

18,- DM

145 öS

18,- sfr

75,- fmk

15900 Lit

23,90 hfl

DOS EXTRA Nr. 19 - 1992

DOS EXTRA

Sonderausgabe DOS EXTRA

Nr. **19** 1992

ISSN 0934-2842

DMV-Verlag

Eine Publikation der
DOS INTERNATIONAL

Raytracer zum Abtippen

Entwerfen Sie eigene Welten

Grafik selbstgemacht

Vektorgrafik in Echtzeit
Animation mit Quick Basic
Superschnelle Fraktale

Sprites

So programmiert
man sie auf dem PC

EGA- und VGA-Karten

Register auf einen Blick

Computergrafik
von A bis Z

Von allem das Beste zu Ihrem Vorteil

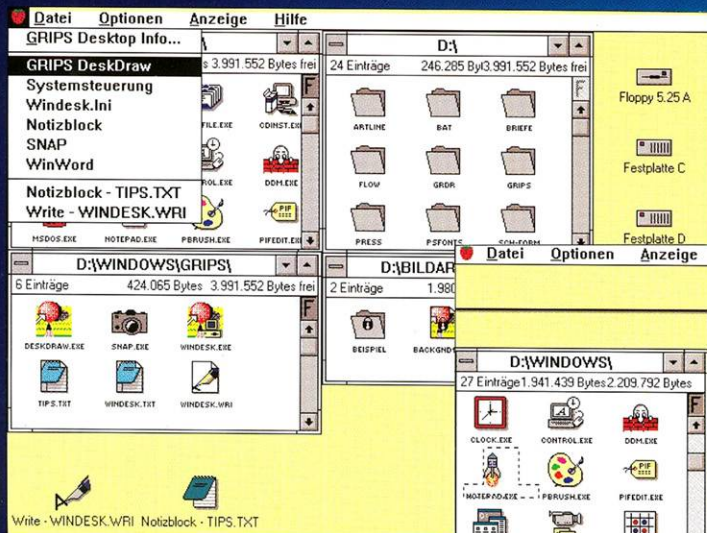
DeskDraw

Grips

Desktop

Die maßgeschneiderte Benutzeroberfläche für Windows

Windows für den Rechner - Grips für den Anwender



Ergonomisches Grundkonzept

Zur arbeitsgerechten Gestaltung gehören leichte Bedienbarkeit, kurze Wege und eine schnelle Übersicht. Deshalb zeigt GRIPS zum Beispiel alles Arbeitsmaterial sichtbar platziert auf dem Desktop oder Sinnbilder statt textlicher Beschreibung und ermöglicht Multitasking auf dem Bildschirm



Funktionale Ausstattung

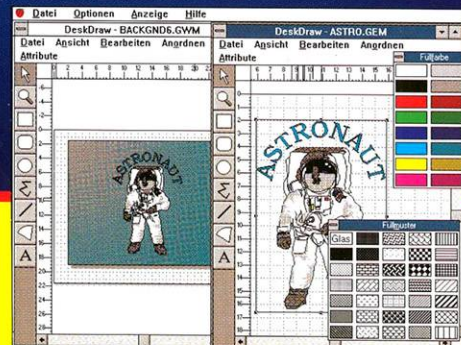
Natürlich bleiben alle Windows-Funktionen erhalten, aber zum komfortablen Arbeiten braucht man mehr, zum Beispiel: Laufwerke und Festplatten als Icons auf dem Bildschirm, ein Papierkorb zum schnellen Löschen, das Schnellstart-Menü (Erdbeere) für die häufigsten Anwendungen, Sortier- und Anzeigefilter für jedes Verzeichnis extra, Archivieren und Komprimierung (bis 50%) auf Knopfdruck, Einladen durch Schieben der Datei auf das Programm-Icon

Persönliche Anpassung

Einfache Voreinstellungen sind die Basis für eine individuelle Konfiguration. Das Layout orientiert sich an modernen Gesichtspunkten und unterstützt in der Farbgebung alle Windows-Optionen. Die Platzierung der Icons und Fenster ist variabel. Die Anzahl der gleichzeitig geöffneten Fenster läßt sich voreinstellen. Die Zuordnung von Icons zu Funktionen ist frei wählbar. Neue Icons können aufgenommen werden. Mit den Einträgen im Erdbeer-Menü lassen sich auch Dokumente mit Ihren Programmen verbinden.

Integriertes DeskDraw

Das vektororientierte Zeichenpaket hat gegenüber Pixel-Malprogrammen den Vorteil, daß Objekte verlustfrei vergrößert und verkleinert werden können. Kanten und Treppentufen gehören der Vergangenheit an, da jedes Objekt in der maximalen Druckerauflösung ausgegeben wird. Außerdem: freiplatzierbare Menüs und Bildaustausch mit Artline, DMV-Atelier, Corel Draw, MicroGrafX Designer u.a.



PCdirekt

2/92:
"Cleveres Windows-Utility mit ausgezeichnetem Preis-Leistungs-Verhältnis"

PCWOCHEN

38/91
"Neue Generation für modulare Software"

Computer
PERSÖNLICH

1/92
"... der wohl vielversprechendste Ansatz auf dem Weg zu einer ergonomischen PC-Oberfläche"

DOS
INTERNATIONAL
DAS MAGAZIN FÜR AKTIVE PC-ANWENDER

12/91
"... Macintosh-Feeling ..."

Win-DOS

2/92
DeskDraw: "... schafft den Brückenschlag zwischen GEM- und Windows-Programmen"

PC
PROFESSIONELL

2/92
"Im Endeffekt wird Micro-soft wieder abkupfern und diese Ideen in die über-nächste Windows-Version einbauen."

Grips-Desktop und Grips-Deskdraw DM 199,—*

HexEdit Editor für Windows-Programme

DM 99,—*

DMV-Verlag • Postfach 250 • W-3440 Eschwege

DMV
Daten- und Medienverlag

* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.

Computer-Kunst

Malerei, Grafik und Bildhauerei sind wohl in allen Kulturen, die je Geschichte schrieben und noch schreiben werden, tief verwurzelt. Schon immer ist es die Kunst gewesen, die in uns das Schöpferische zu Tage fördert, den Geist belebt und die Fantasie beflügelt.

Mit der schnell wachsenden Leistung moderner Computer, mit immer schnelleren Grafikkarten und noch mehr MIPS und FLOPS steuern wir einer Zukunft entgegen, in der Begriffe wie »virtual reality« und »cyberspace« sehr viel Gewicht haben werden. Dieser Entwicklung möchten wir mit dem vorliegenden Heft Rechnung tragen.

Ein Highlight dieser Ausgabe ist der Raytracer, der Ihnen die Konstruktion eigener Welten aus geometrischen Objekten mit Licht, Schatten, Spiegelung und Transparenz erlaubt. Zaubern Sie mit dem Textur-Malprogramm Oberflächenstrukturen, und bauen Sie mit dem 3D-Editor eigene Szenarien zusammen.

Einen weiteren Weg in die unendlichen Tiefen der Computergrafik weist Ihnen der Fraktal-Explorer. Wandeln Sie auf Mandelbrots Spuren, und erleben Sie in Sekundenschnelle die fantastischen Strukturen fraktaler Welten.

Ein Gefühl von Geschwindigkeit wird Ihnen auch der Vektor-Generator vermitteln. Drehen Sie sich in Echtzeit um Ihre selbst entworfenen Drahtmodelle. Vektor-Mania zeigt, wie's geht.

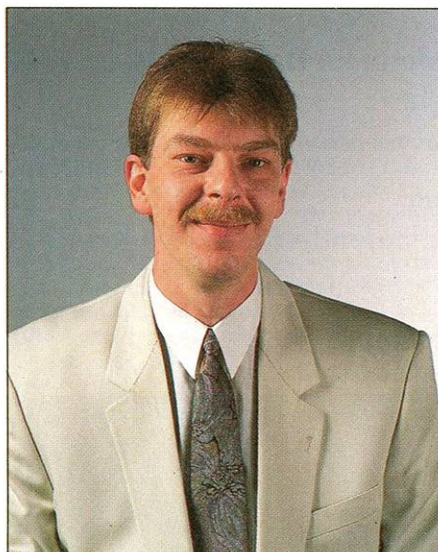
Sie mögen schnelle Grafik? Dann geht es Ihnen wie mir. Lesen Sie, wie es Ihnen gelingt, daß animierte Sprites über einen sich bewegendem Hintergrund flitzen – und das in Turbo Pascal!

Freunde der OOP werden ebenfalls auf ihre Kosten kommen. In zwei Beiträgen werden wir Ihnen die Grundlagen der zwei- und dreidimensionalen Computergrafik vorführen.

Eine Referenz der wichtigsten Register der EGA- und VGA-Karten, die Animations-Programmierung in Quick Basic sowie natürlich die obligatorischen Tips & Tricks runden das Programm ab. Bleibt mir nur noch, Ihnen ein paar erlebnisreiche Stunden bei der Lektüre dieses Hefts zu wünschen.

Ihr

Reinhold Müller



INHALT

DOS EXTRA
19/92

GRUNDLAGEN

2D-Grafik	6
Schritt für Schritt	
3D-Grafik	18
Grafik-Objekte	
Bildmanipulationen mit Quick Basic	40
Grafik ohne Grenzen	
»PUT« und »GET«	42
Grafikbefehle in Basic	
Utilities für Quick Basic	48
Basic und die Grafikwelt	
VGA-Farbpalette im Detail	52
Bunt gemischt	
Auto-Grabber	62
Der Film am laufenden Band	
Grafik-Funktionen	76
Zoom mit Perspektive	

— TIPS & TRICKS ZUR GRAFIK —

Der VGA-monochrome Emulator	80
Linien zeichnen Muster	80
Betrachten von PCX-Dateien	81
Diashow im Speicher	81
Drehende Spirale	82
Extrem schnelle Fraktaldarstellung	85



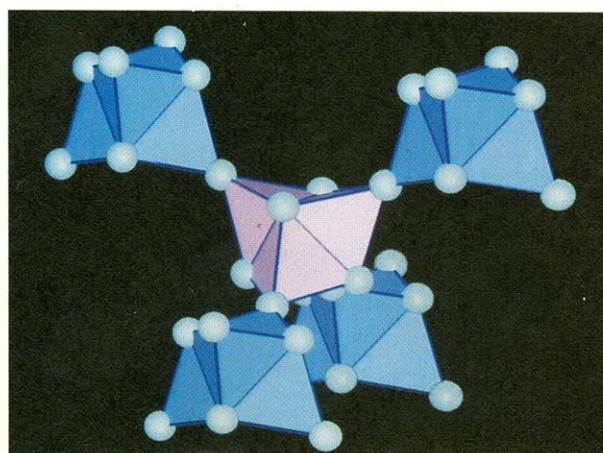
Eine Sammlung objektorientierter Routinen in Turbo Pascal 6.0 führt Sie systematisch in die Darstellung der 2D-Grafik ein. Zaubern Sie Moiré-Muster und versehen Sie Flächen mit Ornamentik.

Seite 6

Fliegender Teppich	85
Farben mit ihren Attributen	86
Atari-Bilder am PC	87
Abrollende Kreise	88
Zweiter Mischer aller Farben	88
Kurven in drei Dimensionen	90
VGA-Grafikkarte im	
Geschwindigkeitsrausch	90
Einfache Hintergrundgrafik	99
Fraktal zum Zoomen	99
Die Dateizerteil-Maschine	100

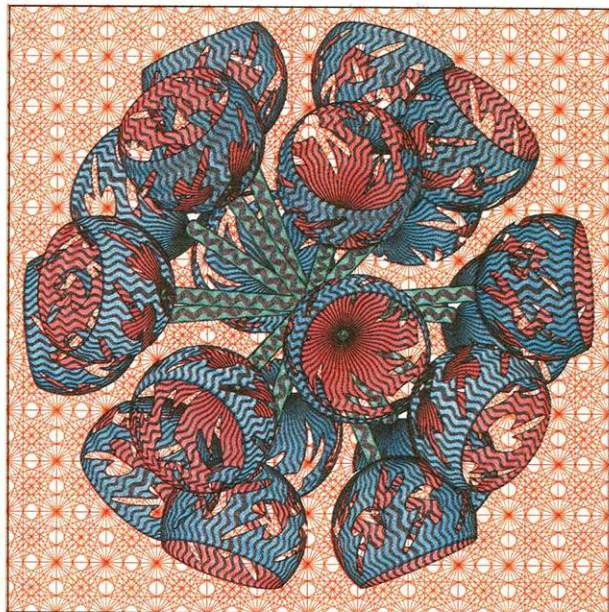
Echtzeit heißt das Zauberwort. Lernen sie die Definition von Drahtmodellen kennen, ihre Drehung um beliebige Raumachsen sowie die perspektivische Transformation auf zwei Dimensionen. Hohe Geschwindigkeit erreichen Sie durch Verwendung von Tabellen, Makros und anderen Tricks auf Assemblerebene.

Seite 116



Hier erfahren Sie die Grundlagen der 3D-Grafik. Konstruieren Sie Szenarien aus platonischen Körpern.

Seite 18



PROGRAMMIERUNG

Scrolling 102

Trickkiste für VGA-Karten

Animation 110

Den Bildern Beine gemacht

Drahtgittermodelle 116

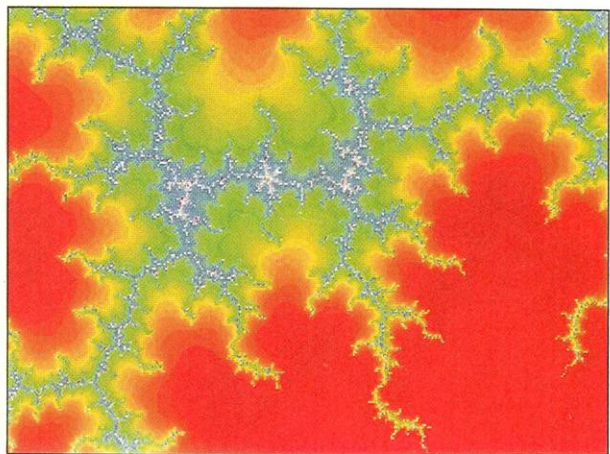
Vektor-Mania

Fraktal-Explorer 120

Apfelmännchen wird entdeckt

Raytracing in C 126

Schöne neue Welt



Ein starkes Gespann aus C und Assembler führt Sie in die Welt der fraktalen Grafiken.

Seite 120



Die Spriteprogrammierung in all ihren Facetten vermittelt Ihnen dieser Beitrag. Angefangen bei den verschiedenen Methoden der Ganzseiten-Animation bis zur mehrphasigen Sprite-Programmierung. In Verbindung mit der »Trickkiste für VGA-Karten« steuern Sie animierte Spielfiguren über scrollenden Hintergrund und halten dabei einen Teil des Bildschirms als Statusanzeige fest.

Seite 110

REFERENZ

Assembler/Turbo Pascal 158

EGA/VGA-Internia

RUBRIKEN

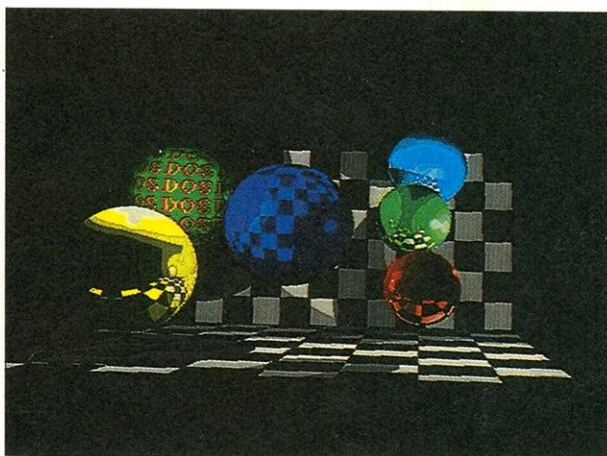
Editorial 3

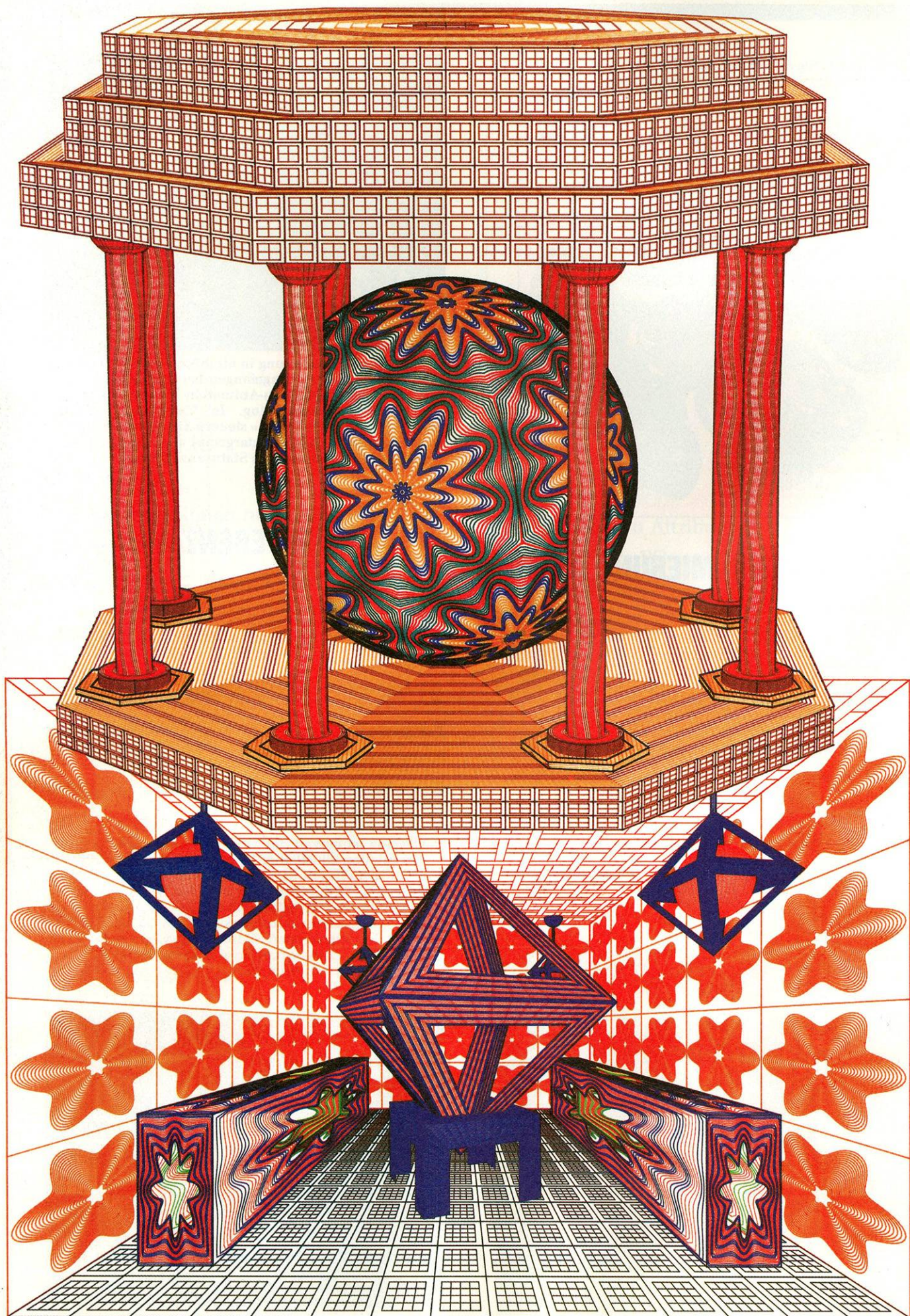
Computer-Kunst

Impressum 166

Raytracing ist derzeit die Methode erster Wahl, wenn es um Simulation künstlicher Welten geht. Wir vermitteln Ihnen die Grundlagen dieses Verfahrens und stellen einen Raytracer in Turbo C vor. Der eingebaute CAD-Editor erleichtert Ihnen den Aufbau Ihrer Szenarien und erspart Ihnen somit das Erlernen einer komplizierten Script-Sprache. Mit dem integrierten Malprogramm erzeugen Sie beliebige Oberflächenstrukturen wie beispielsweise Schrift auf Kugeln oder Schachbretter.

Seite 126





2D-Grafik

Schritt für Schritt

Seit einiger Zeit erlaubt Turbo Pascal die objektorientierte Programmierung (OOP). Zwar gibt es inzwischen schon eine ganze Reihe von Büchern zu diesem Thema, doch der systematische Einsatz bei nichttrivialen geometrischen Problemen scheint noch nicht sehr verbreitet zu sein.

Haben Sie schon einmal versucht, Grafiken objektorientiert zu programmieren? Nein? Dann werden Sie Augen machen, wie elegant sich auch kompliziertere Probleme der zwei- und dreidimensionalen Grafik in dieser neuen Programmieretechnik darstellen lassen.

Zunächst werden Sie sich mit Routinen beschäftigen, mit denen Sie bei beliebiger Auflösung Grafik auf den Bildschirm bringen. Die Strategie des Zeichnens wird gleich so angelegt, daß sie sowohl für die 2D- als auch für die 3D-Grafik gilt. Anschließend lernen Sie die Hierarchie von Objekttypen der 2D-Grafik und deren Anwendung an Hand von einfachen Beispielen kennen. Den Schluß bilden die Objekttypen, mit denen Sie Ornamente in der Ebene erzeugen. Es ist klar, daß Sie hier keine umfassende Einführung in die OOP erwartet. Rufen Sie sich nur dies in Erinnerung: Das reservierte Wort »object« ist ein neuer Typ, am besten vergleichbar mit dem Typ »record«. In einem Objekt dürfen Sie im Unterschied zum »record« nicht nur Daten, sondern auch Methoden (Prozeduren und Funktionen) einbringen. Ein Objekt hat folgende wesentlichen Eigenschaften:

- **Datenkapselung:** Die Daten eines Objekttyps sind in allen Methoden des Objekttyps und nur in diesen bekannt.
 - **Vererbung:** Ein Objekttyp läßt sich als »Nachkomme« eines anderen Objekttyps, des »Vorfahren«, definieren. Dem Nachkommen sind alle Daten und Methoden des Vorfahren bekannt.
 - **Virtuelle Methoden und Polymorphie:** In einem Objekt können virtuelle Methoden definiert werden, deren Adressen nicht bei der Kompilierung, sondern erst während der Laufzeit festgelegt werden. Methoden, die virtuelle Methoden aufrufen, nennt man polymorph.
- Die objektorientierte Programmierung ersetzt nicht die bisherige modulare Programmierung, ergänzt diese jedoch auf ideale Art und Weise.

Grundbegriffe der 2D-Grafik

Grundlegend für die gesamte 2D-Grafik ist das 2D-Weltkoordinatensystem »WCS2D« (2D-Benutzer-Koordinatensystem). In diesem definieren Sie das 2D-Weltintervall »WI2D«, innerhalb dessen alle geometrischen Objekte festgelegt werden (Bild 1). Wie alle zweidimensionalen Intervalle definieren Sie es durch den linken unteren und rechten oberen Eckpunkt. Um auf dem Bildschirm beliebige Bildschirm-Intervalle bei beliebigen Auflösungen festzulegen, geben Sie ein skaliertes Bildschirmintervall »IScrScal« (Interval Screen Scaled) vor, indem Sie die Breite und Höhe des Weltintervalls in je 10 Teilintervalle unterteilen (Bild 2). Diese Skalierung wird dann automatisch umgerechnet. In der Unit »u1.pas« (Listing 1) finden Sie einige grundlegende Typen

Wer
wo
wann?

Bei Zuschriften bitten wir folgendes zu beachten:
Haben Sie Fragen, Anregungen oder Programme, die Sie der DOS-Redaktion zukommen lassen möchten, adressieren Sie wie folgt:

Redaktion
»DOS International«
Gruberstraße 46c
8011 Poing
Tel. (0 81 21) 769-0

★ ★ ★

Haben Sie Fragen, die zum Beispiel Abonnements, Kleinanzeigen oder Data-box-Bestellungen betreffen, wenden Sie sich bitte an den Verlag:

DMV-Verlag
DOS International
Postfach 250
3440 Eschwege
Tel. (0 56 51) 8 09-0
Telefax (0 56 51) 8 09-3 33

NEU!

80386-40 64KB

nur DM 2949,-

4 MB RAM, 105 MB Festplatte
AT-BUS, 1,2 MB und 1,44 MB
Diskettenlaufwerke, 2 seriell/1
parallel/1 game, 1 MB 16 bit
non-interlaced Grafikkarte, 14"
FarbMultiScan non-interlaced und
strahlungsarm 0,28 mm, Desktop-
Gehäuse, MF102, Softwarepaket

80486-33 256KB

nur DM 3549,-

GENIUS GM 6 DM 29,-
SuperMouse II DM 49,-
Microsoft Mouse DM 79,-
GeniTrak GK-T320 DM 145,-
GeniScan GS4500 DM 249,-

TRIDENT 1 MB 16 bit DM 169,-
SpeedStar HiColor 1 MB 16 bit
32767 Farben DM 349,-

CM 1402 MultiScan
1024*768 0,28 mm DM 598,-
CM 1402VA (wie 1402, aber
strahlungsarm) DM 698,-
CM 1448 (wie 1402, aber
non-interlaced) DM 698,-
NYTECH 1448VA (wie
1402VA+1448) DM 798,-
ADI MicroScan 3E-LR 14" MPR
II/non-interlaced (s. Test c't
1/92) DM 1198,-
MAG MX 17H 17", 0,26 mm,
1280*1024, PC-PROF.
Editor's Choice DM 1998,-

HÄNDLERANFRAGEN
ERWÜNSCHT!

Alle Preise zzgl. Fracht/
Verpackung/Versicherung per
UPS-BAR/EC-Nachnahme. Irrtum und
Lieferbarkeit vorbehalten.

C&P

Arnd von Wedemeyer
Bödekerstraße 74 W-3000 Hannover 1
Telefon 0511-38837-18/-20/-21 Fax 38837-23
BTX-Leitseite WEDEMEYER#

und Methoden. Das Programm erklärt sich selbst; nur wo es notwendig scheint, sind Kommentare eingefügt. Mit dem Programm »cs2d.pas« (Listing 2) zeichnen Sie das 2D-Weltkoordinatensystem im gewählten Bildschirmintervall (Bild 3). Verändern Sie einmal zum besseren Verständnis die Definitionen für das 2D-Weltintervall »WI2D«, das skalierte Bildschirmintervall »IScrScal« und das Intervall »I2D«.

Kurven sind das A und O der Grafik

Eine Kurve ist die Abbildung eines Parameter-Intervalls »I1D« in das 2D-Weltintervall, das heißt, einem Punkt T des Parameterintervalls »I1D« wird ein Punkt P im 2D-Weltintervall zugeordnet (Bild 4).

Da Sie natürlich nicht alle Punkte einer Kurve berechnen können, unterteilen Sie das Parameterintervall in $LL+1$ Punkte $T(L)$ ($L=0,1,\dots,LL$). Diese Unterteilung wird durch die Abbildung auf die Kurve übertragen und erzeugt dort $LL+1$ Teilpunkte $P(L)$. Die Kurve entsteht dann dadurch, daß Sie zwei aufeinanderfolgende Punkte durch ein Geradenstück verbinden. Wenn Sie die Anzahl LL der Unterteilungspunkte groß genug gewählt haben, erhalten Sie eine genügend glatte Kurve.

Nun sollen zwei aufeinanderfolgende Punkte nur dann verbunden werden, wenn sie noch einem zusätzlichen Kriterium genügen. Sie könnten zum Beispiel fordern, daß die Punkte – in der 2D-Grafik – in einem vorgegebenen Ausschnitt des 2D-Weltintervalls liegen, oder – in der 3D-Grafik – daß sie zu Punkten auf einer Fläche gehören. Dies bedeutet, daß Sie zu jedem Punkt $P(L)$ noch einen Booleschen Parameter »Check(L)« berechnen müssen. Nur wenn $Check(L-1) = Check(L) = True$ gilt, verbinden Sie die beiden Punkte $P(L-1)$ und $P(L)$ durch einen Streckenzug.

Nun erhöht es oft beträchtlich die Anschaulichkeit eines

Bildes, wenn Punkte $P(L)$ mit $Check(L) = False$ nicht miteinander verbunden werden. Hierzu führen wir noch den Parameter »WP« (With Points) ein. Ist dieser gleich True, so wird der Punkt wiedergegeben, aber die Verbindungslinien zu den »Nachbarn« weggelassen.

Nehmen Sie also an, Sie hätten bis zum Punkt $P(L-1)$ gezeichnet oder punktiert. Bezeichnen Sie dessen Check-Parameter $Check(L-1)$ mit »ChkOld«. Jetzt berechnen Sie den neuen Punkt $P(L)$ und den zugehörigen Check-Parameter $Check(L)$, den Sie mit »Chk« abkürzen. Sie werden nur dann eine Gerade vom Punkt $P(L-1)$ zum Punkt $P(L)$ ziehen, wenn beide Check-Parameter True sind. Sonst bewegen Sie nur den Cursor zum Punkt $P(L)$. Ist $WP = True$, setzen Sie den Punkt. Dies geschieht in der Prozedur »Draw« (Listing 3). Es sei nochmals betont, daß diese Prozedur sowohl für die 2D- als auch für die 3D-Grafik gilt.

Da kompliziertere Bilder auf langsamen Rechnern doch einige Zeit brauchen, sollten Sie die Bilder speichern. Dies erreichen Sie mit dem Parameter »ScrFile«. Zum Aufruf der Bilddateien steht die Prozedur »DrawScrF« bereit.

Die Objekthierarchie der 2D-Grafik

Die zum Zeichnen von zweidimensionalen Grafiken nötigen Objekte lassen sich in einer übersichtlichen Struktur anordnen (Bild 5). Bei den einzelnen Objekttypen sind im folgenden nur die Methoden angegeben.

● Der Objekttyp »Curve2DT«, die Prozeduren »TToP« und »Check«:

Die objektorientierte Programmierung erlaubt es, eine beliebige 2D-Kurve mit einer einzigen Prozedur zu zeichnen. Dazu dient der Objekttyp »Curve2DT« (Listing 4), der an der Spitze der Hierarchie steht. Weil er so wichtig ist, bildet er eine eigene Unit. Zum Verständnis dieses Objekttyps sei

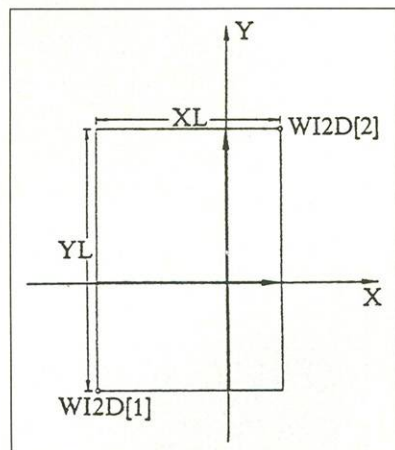


Bild 1. Das zweidimensionale Weltkoordinatensystem »WCS2D«

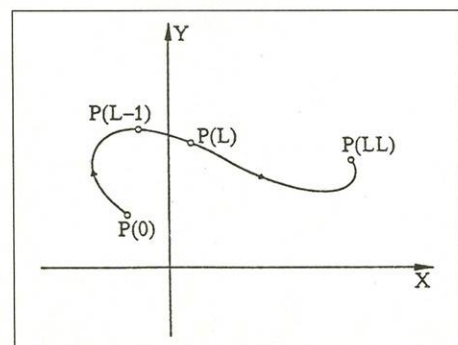


Bild 4. Eine Kurve entsteht durch Abbildung des Parameterintervalls »I1D« auf das 2D-Weltintervall.

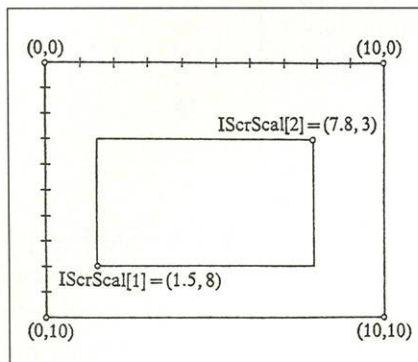


Bild 2. Die Zeichenfläche wird in zehn Teilintervalle aufgeteilt.

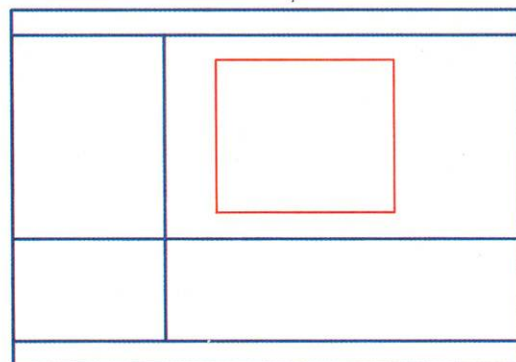


Bild 3. Mit dem Programm »cs2d.pas« zeichnen Sie in dem 2D-Weltkoordinatensystem.

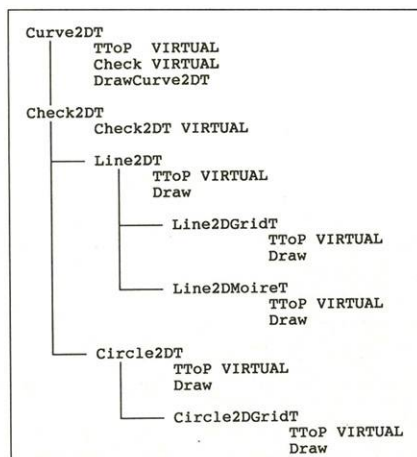


Bild 5. Die Objekthierarchie der 2D-Grafik

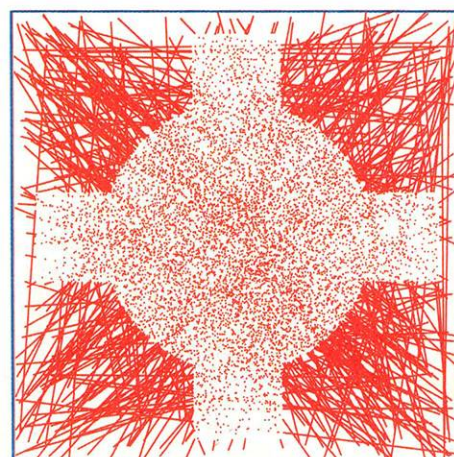


Bild 6. Mit dem Programm »line1.pas« erzeugen Sie Zufallslinien.

daran erinnert, daß Sie zu jedem Unterteilungspunkt T des Parameterintervalls »I1D« den Punkt P und den Parameter »Chk« berechnen müssen. Die entscheidende Idee dieser Unit ist es, diesen Vorgang in zwei verschiedene Prozeduren aufzuteilen: Mit der Prozedur »TToP« berechnen Sie zu dem Parameterwert T den Punkt P, mit einer Prozedur »Check« den Parameter »Chk«.

Damit Sie mit »Curve2DT« alle zweidimensionalen Kurven zeichnen können, definieren Sie die beiden Prozeduren »TToP« und »Check« als virtuelle Methoden, Sie werden also erst zur Laufzeit kompiliert. Die Prozedur »DrawCurve2D« dagegen ist eine sogenannte statische Prozedur. Sie greift auf die beiden Prozeduren »TToP« und »Check« zu, obwohl diese bei ihrer Kompilierung noch gar nicht implementiert sind. Dadurch wird sie zu einer polymorphen Prozedur.

Die entscheidende Prozedur »DrawCurve2D« ist jetzt leicht zu verstehen. Nacheinander werden in der L-Schleife die Parameterwerte T erzeugt. Zu jedem T wird durch die Prozedur »TToP« der zugehörige Kurvenpunkt P, zu diesem wiederum durch die Prozedur »Check« der Parameter Chk berechnet. Dann wird die Prozedur »Draw« aufgerufen. Zusätzlich eingebaut wurde ein Interpolationsverfahren. Sie versuchen natürlich, eine Kurve mit möglichst wenigen »Stützstellen« zu zeichnen. Es kann aber nun sein, daß zwar wenige Punkte genügen, um eine »schöne« Kurve zu konstruieren, daß jedoch häßliche Lücken entstehen, wenn für

zwei aufeinanderfolgende Punkte die Check-Parameter verschieden sind. Sie könnten zwar die Zahl der Punkte insgesamt erhöhen, doch dies ist für die »unproblematischen« Teile der Kurve gar nicht nötig. Es genügt, die Kurve zwischen zwei solchen kritischen Punkten mit einer Anzahl von Zwischenpunkten, die Sie ebenfalls noch festlegen können, zu interpolieren.

• Der Objekttyp »Check2DT«:

Ebenso wie die Berechnung der Kurvenpunkte wird das Zeichnen einer Kurve in zwei Schritte zerlegt. Mit dem Objekttyp »Check2DT« (Listing 5) legen Sie fest, in welchen Teilen des zweidimensionalen Weltintervalls »WI2D« gezeichnet werden soll. Sie schneiden sozusagen aus dem 2D-Weltintervall Gebiete heraus.

Als Beispiel dienen hier zwei Rechtecke und ein Kreis. Mit den Prozeduren »InInterval2D« und »InCircle« stellen Sie fest, ob ein Punkt P innerhalb oder außerhalb des betreffenden Gebiets liegt. Die nötigen mengentheoretischen Verknüpfungen wie Durchschnitt (AND) und Vereinigung (OR) sind in einer CASE-Abfrage formuliert. Deren einzelne Fälle initialisieren Sie in den noch zu definierenden Zeichen-Objekttypen mit dem Parameter »KOfCut« (kind of cut = Art des Ausschneidens).

• Zeichnen von Geraden und Kreisen:

Jetzt können Sie Objekttypen erzeugen, in denen Sie die virtuelle Prozedur »TToP« und die Zeichenprozedur »Draw« definieren. Damit das Verfahren transparent wird, seien nur

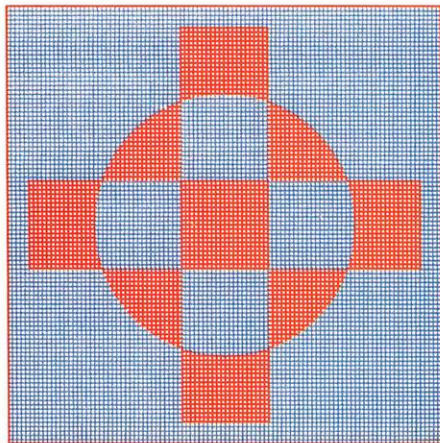


Bild 7. Mit dem Programm »line2.pas« zeichnen Sie bei eingeschalteter Interpolation gitterförmige Linien. Die Interpolationsschritte werden mit »NumbIntPol: = 4« festgelegt. Mit »wp: = False« erhalten Sie durchgezogene Linien.

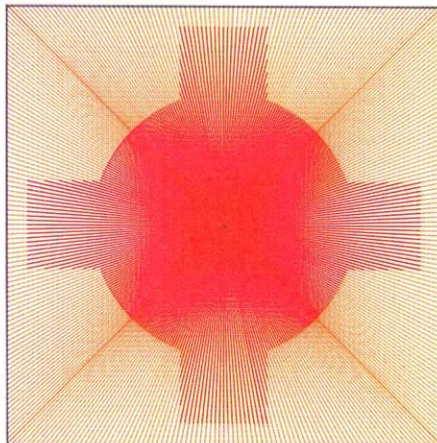


Bild 8. »moire1.pas« zeigt, wie Sie Moiré-Muster zeichnen.

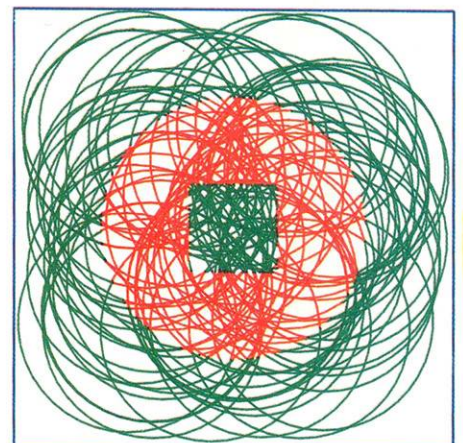


Bild 9. Mit »circle1.pas« zeichnen Sie Kreise.

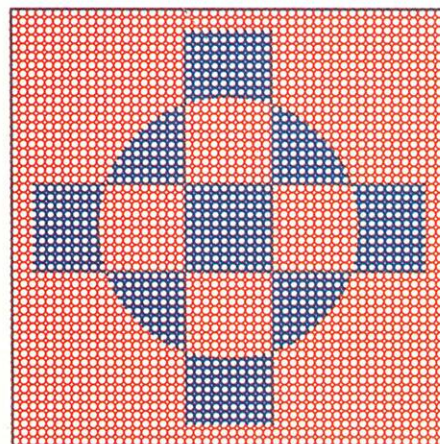


Bild 10. »circle2.pas« zeichnet Kreisgitter.

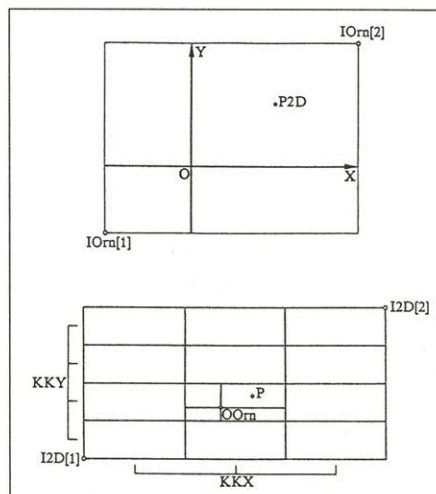


Bild 11. Ornamentik entsteht durch Verknüpfung eines Musters per Translation.

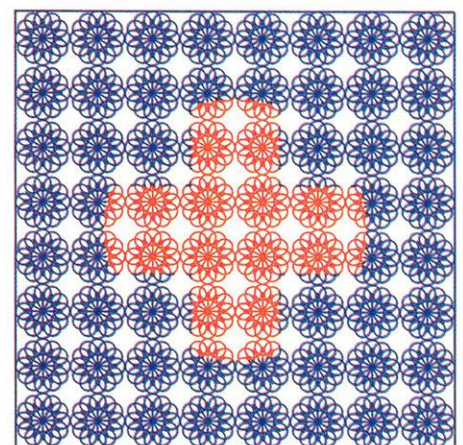


Bild 12. Das Programm »orn1.pas« erzeugt in Zusammenarbeit mit der Unit »uorn.pas« Ornamentik auf Ihren zweidimensionalen Objekten. Dazu bilden Sie ein Ornamentikmotiv, das Sie in einem Intervall erzeugt haben, mehrfach auf ein Intervall »I2D« ab.

Objekttypen für Geraden (»Line2DT«) und Kreise (»Circle2DT«) definiert (Listing 6). Mit diesen beiden Typen lassen sich weitere Objekttypen definieren: »Line2DGridT«, »Line2DMoireT« und »Circle2DGridT«. Da diese Objekttypen virtuelle Prozeduren enthalten, müssen sie mit einem Konstruktor initialisiert werden.

Die folgenden Demonstrationsprogramme verwenden die vorgestellten Objekttypen:

- Mit dem ersten Programm »line1.pas« (Listing 7) zeichnen Sie Zufallslinien (Bild 6). Mit »IntPol=True« schalten Sie die Interpolation hinzu; die Anzahl der Interpolationsschritte wird durch »NumbIntPol« festgelegt.

- Das Programm »Line2« (Listing 8): Hiermit zeichnen Sie gitterförmige Linien (Bild 7). Zu diesem Zweck initialisieren Sie in den zwei Instanzen des Objekttyps »Line2DT« jeweils zwei komplementäre Gebiete.

- Das Programm »Moire1«: Mit diesem Programm (Listing 9) zeichnen Sie Linien, die ein Moiré-Muster erzeugen (Bild 8). »Moire1« demonstriert Ihnen zugleich die Wirkung der Interpolation: Mit Interpolation genügen LL = 12 Punkte; wenn Sie aber die Interpolation abschalten, benötigen Sie mehr als 100 Punkte, um dieselbe Qualität zu erreichen.

- Das Programm »Circle1« setzt den Objekttyp »Circle2DT« zum Zeichnen von Kreisen ein (Listing 10). Zuerst zeichnen Sie Zufallskreise (Bild 9).

- Mit dem Programm »Circle2« zeichnen Sie Kreisgitter (Bild 10, Listing 11).

- Etwas 2D-Handwerkszeug: In der Unit »u2dgeom.pas«

(Listing 12) sind einige selbsterklärende Typen und Prozeduren zusammengestellt, die Sie im folgenden sowohl im Rahmen der 2D- als auch 3D-Grafik benötigen.

- Ornamentik: Ornamente erzeugen Sie mit dem Objekttyp »Orn2DT« (Listing 13). In diesem sind jene Parameter und Methoden integriert, mit denen Sie ein Verzierungsmotiv, das Sie in einem Intervall »IOrn« generieren, mehrfach in ein anderes Intervall »I2D« abbilden können. Das Intervall »I2D« unterteilen Sie in KX*KKY Teilintervalle (Bild 11).

Anschließend sind noch zwei Objekttypen angegeben, mit denen Sie Geraden und Kreise mit Ornamenten verknüpfen.

- Das Programm »Orn1«: In diesem Programm (Listing 14) wenden Sie nun die Objekttypen des vorangehenden Abschnitts an (Bild 12).

Zum Schluß sei noch angedeutet, wie Sie auch schon in der 2D-Grafik sehr anspruchsvolle Geometrie betreiben können. In den Bänden »Computergrafik 1 und 2« werden hierzu Familien von Sternkurven, Polygonen und Blättern definiert. Ihre Ornamente können Sie aus diesen Elementen zusammenbauen. Sehr schöne Motive erhalten Sie, wenn Sie Elemente »übereinanderschachteln«. Mit Check-Abfragen legen Sie fest, welche Elemente sichtbar sein und welche vom »Vordergrund« verdeckt werden sollen.

(Prof. Dr. Kurt Endl/bm)

Literatur:

Kurt und Robert Endl: Computergrafik 1, Würfel-Verlag 1989, 448 Seiten, ISBN 3-92321007, 78 Mark

Kurt Endl: Computergrafik 2, Würfel-Verlag 1991, 528 Seiten, ISBN 3-92321005, 88 Mark

```
[2W2] 1: {Programm: ul.pas
[737] 2: Funktion: Bereitstellung von Funktionen
[HU4] 3: zur Erzeugung von 2D-Grafik
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[AA1] 8: UNIT U1;
[608] 9: {$IFDEF CPU87}{$N+}{$ELSE}{$N-}{$ENDIF}
[SK1] 10: INTERFACE
[ZN2] 11: USES Crt, Graph;
[MT7] 12: {$IFDEF CPU87}TYPE EXTENDED=REAL;{$ENDIF}
[ST0] 13: TYPE
[PY1] 14: Pt1D = RECORD
[LY1] 15: X: EXTENDED;
[ZN0] 16: END;
[3A5] 17: Interval1D = ARRAY[1..2] OF Pt1D;
[UR2] 18: Pt2DInt = RECORD
[M52] 19: X,Y: INTEGER;
[ZN0] 20: END;
[WK6] 21: Interval2DInt = ARRAY[1..2] OF Pt2DInt;
[0Y5] 22: Pt2DIntA12 = ARRAY[1..12] OF Pt2DInt;
[KR2] 23: PtScr = Pt2DInt;
[SX4] 24: IntervalScr = Interval2DInt;
[UY1] 25: Pt2D = RECORD
[RC2] 26: X,Y: EXTENDED;
[ZN0] 27: END;
[MN4] 28: Pt2DA12 = ARRAY[0..12] OF Pt2D;
[9A5] 29: Interval2D = ARRAY[1..2] OF Pt2D;
[NB2] 30: PtPol = RECORD
[5L2] 31: R,Phi: EXTENDED;
[ZN0] 32: END;
[4S1] 33: Vt2D = Pt2D;
[0H2] 34: CONST Eps3=0.001;
[VP1] 35: Eps4=0.0001;
[4U1] 36: Eps5=0.00001;
[8P3] 37: VAR MaxXScr,MaxYScr,
[0D4] 38: NRes,NumbIntPol: Integer;
[DX2] 39: ScrFac: EXTENDED;
[KU3] 40: ScrO: Pt2D; ScrU: Vt2D;
[2B4] 41: IScrScal,WI2D: Interval2D;
[MA4] 42: IScr,IScr2D: IntervalScr;
[9L1] 43: O2D: Pt2D;
[5L4] 44: WP,ScrFile,IntPol: BOOLEAN;
[LF3] 45: ScrF: FILE OF INTEGER;
[A05] 46: PROCEDURE InitializeGraphic;
[PY2] 47: PROCEDURE DefPt2D
[SE4] 48: (X,Y: EXTENDED; VAR P: Pt2D);
[3P2] 49: PROCEDURE DefI2D
[8M3] 50: (X1,Y1,X2,Y2: EXTENDED;
[6F3] 51: VAR I2D: Interval2D);
[9P6] 52: PROCEDURE ParameterTransformationScr;
[126] 53: PROCEDURE TransformationIScrScalToIScr
[HV3] 54: (IScrScal: Interval2D);
[6W3] 55: PROCEDURE Parameter2D;
```

```
[HD4] 56: PROCEDURE Transformation
[GD4] 57: (P: Pt2D; VAR PScr: PtScr);
[7A3] 58: PROCEDURE PutPtScr
[2C4] 59: (PScr: PtScr; Col: INTEGER);
[C96] 60: PROCEDURE MoveToPtScr(PScr: PtScr);
[KU3] 61: PROCEDURE DrawToPtScr
[2C4] 62: (PScr: PtScr; Col: INTEGER);
[R63] 63: PROCEDURE DrawIScr
[TM5] 64: (IScr: IntervalScr; Col: INTEGER);
[2Y2] 65: PROCEDURE DrawI2D
[PW4] 66: (I2D: Interval2D; Col: INTEGER);
[6E6] 67: PROCEDURE DrawCS2D(Col1,Col2: INTEGER);
[NS2] 68: PROCEDURE Stop;
[V34] 69: PROCEDURE CloseGraphic;
[6S3] 70: FUNCTION InInterval2D
[AP5] 71: (I2D: Interval2D; P: Pt2D): BOOLEAN;
[L03] 72: FUNCTION InCircle
[Z65] 73: (M,P: Pt2D; R: EXTENDED): BOOLEAN;
[3N3] 74: PROCEDURE PartitionX
[FD5] 75: (I2D: Interval2D; NX,NNX: INTEGER;
[ES2] 76: VAR X: EXTENDED);
[4N3] 77: PROCEDURE PartitionY
[HD5] 78: (I2D: Interval2D; NY,NNY: INTEGER;
[GS2] 79: VAR Y: EXTENDED);
[XG3] 80: PROCEDURE RandomPt2D
[KW4] 81: (I2D: Interval2D; VAR P: Pt2D);
[000] 82:
[4L2] 83: IMPLEMENTATION
[000] 84:
[PY2] 85: PROCEDURE DefPt2D
[SE4] 86: (X,Y: EXTENDED; VAR P: Pt2D);
[MU0] 87: BEGIN
[BM2] 88: P.X:=X; P.Y:=Y;
[ZN0] 89: END;
[3P2] 90: PROCEDURE DefI2D
[8M3] 91: (X1,Y1,X2,Y2: EXTENDED;
[6F3] 92: VAR I2D: Interval2D);
[MU0] 93: BEGIN
[XB4] 94: I2D[1].X:=X1; I2D[2].X:=X2;
[7C4] 95: I2D[1].Y:=Y1; I2D[2].Y:=Y2;
[ZN0] 96: END;
[A05] 97: PROCEDURE InitializeGraphic;
[357] 98: {Berechnet in Abhängigkeit der gewählten
[KL6] 99: Auflösung NRes die Bildschirmparameter
[006] 100: und initialisiert den Grafikmodus}
[RN0] 101: VAR
[AZ4] 102: Gd,Gm,ErrorCode,Mode: INTEGER;
[7X2] 103: XAsp,YAsp: Word;
[MU0] 104: BEGIN
[T02] 105: CASE NRes OF
[T35] 106: 0: BEGIN Gd:=0; END; {Detect}
[N05] 107: 1: BEGIN Gd:=1; Gm:=0; END; {CGA}
[515] 108: 2: BEGIN Gd:=3; Gm:=1; END; {EGA}
[E35] 109: 3: BEGIN Gd:=9; Gm:=2; END; {VGA}
[ZN0] 110: END;
```



```

[H53] 111: InitGraph(Gd,Gm, '');
[U74] 112: ErrorCode:=GraphResult;
[765] 113: If ErrorCode <> GrOk THEN BEGIN
[DB1] 114:   ClrScr;
[202] 115:   GotoXY(20,6);
[4T2] 116:   CASE ErrorCode OF
[1S4] 117:     -3: WriteLn ('.BGI not found');
[TC5] 118:     -5: WriteLn ('Not enough memory',
[23] 119:       'to load BGI-file');
[ZR4] 120:     -1,-2,-4,-10: WriteLn ('Error in',
[0T4] 121:       'Graphic Initizialization');
[ZN0] 122:   END;
[T41] 123:   HALT(1);
[ZN0] 124: END;
[3K4] 125: GetAspectRatio(XAsp,YAsp);
[D43] 126: ScrFac:=XAsp/YAsp;
[PC6] 127: MaxXScr:=GetMaxX; MaxYScr:=GetMaxY;
[ZN0] 128: END;
[126] 129: PROCEDURE TransformationIScrScalToIScr
[HV3] 130: (IScrScal: Interval2D);
[S17] 131: {Transformiert das skalierte Bildschirm-
[156] 132:  intervall IScrScal in das Bildschirm-
[6Z2] 133:  intervall IScr};
[123] 134: VAR MXS,MYS: EXTENDED;
[MU0] 135: BEGIN
[2B2] 136:   MXS:=MaxXScr;
[7B2] 137:   MYS:=MaxYScr;
[EP6] 138:   IScr[1].X:=TRUNC(IScrScal[1].X/10*MXS);
[MP6] 139:   IScr[1].Y:=TRUNC(IScrScal[1].Y/10*MYS);
[NP6] 140:   IScr[2].X:=TRUNC(IScrScal[2].X/10*MXS);
[VP6] 141:   IScr[2].Y:=TRUNC(IScrScal[2].Y/10*MYS);
[ZN0] 142: END;
[7A3] 143: PROCEDURE PutPtScr
[2C4] 144: (PScr: PtScr; Col: INTEGER);
[MU0] 145: BEGIN
[MV4] 146:   PutPixel(PScr.X,PScr.Y,Col);
[ZN0] 147: END;
[C96] 148: PROCEDURE MoveToPtScr(PScr: PtScr);
[MU0] 149: BEGIN
[5R3] 150:   MoveTo(PScr.X,PScr.Y);
[ZN0] 151: END;
[KU3] 152: PROCEDURE DrawToPtScr
[2C4] 153: (PScr: PtScr; Col: INTEGER);
[MU0] 154: BEGIN
[WD2] 155:   SetColor(Col);
[HN3] 156:   LineTo(PScr.X,PScr.Y);
[ZN0] 157: END;
[R63] 158: PROCEDURE DrawIScr
[TM5] 159: (IScr: IntervalScr; Col: INTEGER);
[C04] 160: VAR PScr3,PScr4: PtScr;
[MU0] 161: BEGIN
[1U6] 162:   PScr3.X:=IScr[2].X; PScr3.Y:=IScr[1].Y;
[5U6] 163:   PScr4.X:=IScr[1].X; PScr4.Y:=IScr[2].Y;
[HV3] 164:   MoveToPtScr(PScr3);
[W04] 165:   DrawToPtScr(PScr3,Col);
[SF4] 166:   DrawToPtScr(PScr4,Col);
[Y04] 167:   DrawToPtScr(PScr4,Col);
[NF4] 168:   DrawToPtScr(PScr3,Col);
[ZN0] 169: END;
[9P6] 170: PROCEDURE ParameterTransformationScr;
[UT5] 171: {Berechnet aus den Intervallen WI2D
[286] 172:  und IScr das Bild des Ursprungs Scr.O
[GM6] 173:  und den Vektor ScrU für die Multipli-
[NF6] 174:  kationsfaktoren in X- und Y-Richtung};
[ZE5] 175: VAR XL,YL,XScrL,YScrL: EXTENDED;
[MU0] 176: BEGIN
[464] 177:   XL:=WI2D[2].X-WI2D[1].X;
[C64] 178:   YL:=WI2D[2].Y-WI2D[1].Y;
[HS4] 179:   XScrL:=IScr[2].X-IScr[1].X;
[TS4] 180:   YScrL:=IScr[1].Y-IScr[2].Y;
[CS5] 181:   IF (XL/YL<=XScrL/YScrL*ScrFac) THEN
[MU0] 182:     BEGIN
[CZ2] 183:       ScrU.Y:=YScrL/YL;
[3T3] 184:       ScrU.X:=ScrU.Y*ScrFac;
[Y24] 185:       IScr2D[1].X:=IScr[1].X+
[1X4] 186:         TRUNC((XScrL-ScrU.X*XL)/2+0.5);
[554] 187:       IScr2D[1].Y:=IScr[1].Y;
[B34] 188:       IScr2D[2].X:=IScr[2].X-
[1X4] 189:         TRUNC((XScrL-ScrU.X*XL)/2+0.5);
[A54] 190:       IScr2D[2].Y:=IScr[2].Y;
[3D1] 191:     END ELSE
[MU0] 192:     BEGIN
[3Z2] 193:       ScrU.X:=XScrL/XL;
[FS3] 194:       ScrU.Y:=ScrU.X*ScrFac;
[Y44] 195:       IScr2D[1].X:=IScr[1].X;
[D34] 196:       IScr2D[1].Y:=IScr[1].Y-
[UX4] 197:         TRUNC((YScrL-ScrU.Y*YL)/2+0.5);
[354] 198:       IScr2D[2].X:=IScr[2].X;
[A34] 199:       IScr2D[2].Y:=IScr[2].Y+
[UX4] 200:         TRUNC((YScrL-ScrU.Y*YL)/2+0.5);
[ZN0] 201:     END;
[MM5] 202:     ScrO.X:=IScr2D[1].X+TRUNC(ScrU.X
[5M2] 203:       *(-WI2D[1].X)+0.5);
[YM5] 204:     ScrO.Y:=IScr2D[1].Y-TRUNC(ScrU.Y
[6M2] 205:       *(-WI2D[1].Y)+0.5);
[ZN0] 206: END;
[6W3] 207: PROCEDURE Parameter2D;
[MU0] 208: BEGIN

```

```

[747] 209:   TransformationIScrScalToIScr(IScrScal);
[435] 210:   ParameterTransformationScr;
[ZN0] 211: END;
[HD4] 212: PROCEDURE Transformation
[GD4] 213: (P: Pt2D; VAR PScr: PtScr);
[2U6] 214: {Berechnet zu einem Punkt P im Benutzer-
[Z26] 215:  koordinatensystem den Bildschirmpunkt}
[MU0] 216: BEGIN
[B36] 217:   PScr.X:=TRUNC(ScrO.X+ScrU.X*P.X+0.5);
[V36] 218:   PScr.Y:=TRUNC(ScrO.Y-ScrU.Y*P.Y+0.5);
[ZN0] 219: END;
[5X4] 220: PROCEDURE MoveToPt2D(P:Pt2D);
[8T2] 221: VAR PScr: PtScr;
[MU0] 222: BEGIN
[A04] 223:   Transformation(P, PScr);
[693] 224:   MoveToPtScr(PScr);
[ZN0] 225: END;
[037] 226: PROCEDURE DrawToPt2D(P:Pt2D; Col:INTEGER);
[8T2] 227: VAR PScr: PtScr;
[MU0] 228: BEGIN
[A04] 229:   Transformation(P, PScr);
[4W3] 230:   DrawToPtScr(PScr,Col);
[ZN0] 231: END;
[2Y2] 232: PROCEDURE DrawI2D
[PW4] 233: (I2D: Interval2D; Col: INTEGER);
[5V3] 234: VAR IScr: IntervalScr;
[MU0] 235: BEGIN
[VC5] 236:   Transformation(I2D[1], IScr[1]);
[YC5] 237:   Transformation(I2D[2], IScr[2]);
[Y73] 238:   DrawIScr(IScr,Col);
[ZN0] 239: END;
[6E6] 240: PROCEDURE DrawCS2D(Col1,Col2: INTEGER);
[YV1] 241: VAR P: Pt2D;
[MU0] 242: BEGIN
[983] 243:   DrawIScr(IScr,Col1);
[9P3] 244:   DrawIScr(IScr2D,Col1);
[UP5] 245:   P.X:=WI2D[1].X; P.Y:=0; {X-Axis}
[GB2] 246:   MoveToPt2D(P);
[5W3] 247:   P.X:=WI2D[2].X; P.Y:=0;
[P43] 248:   DrawToPt2D(P,Col2);
[1R5] 249:   P.X:=0; P.Y:=WI2D[1].Y; {Y-Axis}
[GB2] 250:   MoveToPt2D(P);
[8W3] 251:   P.X:=0; P.Y:=WI2D[2].Y;
[P43] 252:   DrawToPt2D(P,Col2);
[ZN0] 253: END;
[NS2] 254: PROCEDURE Stop;
[MU0] 255: BEGIN
[R04] 256:   IF Keypressed THEN HALT;
[ZN0] 257: END;
[V34] 258: PROCEDURE CloseGraphic;
[5V3] 259: VAR IScr: IntervalScr;
[MU0] 260: BEGIN
[2T3] 261:   IScr[1].X:=MaxXScr-10;
[YG3] 262:   IScr[1].Y:=MaxYScr;
[VG3] 263:   IScr[2].X:=MaxXScr;
[BT3] 264:   IScr[2].Y:=MaxYScr-10;
[FS2] 265:   DrawIScr(IScr,3);
[Y44] 266:   Repeat Until Keypressed;
[G25] 267:   IF Keypressed THEN CloseGraph;
[ZN0] 268: END;
[VE6] 269: FUNCTION InInterval2D(I2D: Interval2D;
[4R2] 270:   P: Pt2D): BOOLEAN;
[2M6] 271: {Prüft, ob ein Punkt P in einem vorgege-
[YN4] 272:  benen 2D-Intervall I2D liegt}
[MU0] 273: BEGIN
[7D3] 274:   InInterval2D:=TRUE;
[BC3] 275:   IF (P.X<I2D[1].X-Eps3)
[VD3] 276:     OR (P.X>I2D[2].X+Eps3)
[0E3] 277:     OR (P.Y<I2D[1].Y-Eps3)
[0E3] 278:     OR (P.Y>I2D[2].Y+Eps3)
[A94] 279:   THEN InInterval2D:=FALSE;
[ZN0] 280: END;
[KM4] 281: FUNCTION InCircle(M,P: Pt2D;
[4G3] 282:   R: EXTENDED): BOOLEAN;
[926] 283: {Prüft, ob ein Punkt P in einem Kreis
[X37] 284:  mit dem Mittelpunkt M und Radius R liegt}
[MU0] 285: BEGIN
[5R2] 286:   InCircle:=TRUE;
[9G5] 287:   IF SQR(SQR(M.X-P.X)+SQR(M.Y-P.Y))
[ZS4] 288:     >R+Eps3 THEN InCircle:=FALSE;
[ZN0] 289: END;
[X86] 290: PROCEDURE PartitionX(I2D: Interval2D;
[HS5] 291:   NX,NNX: INTEGER; VAR X: EXTENDED);
[YS6] 292: {Zerlegt das X-Intervall von I2D in NNX
[H06] 293:  gleiche Teile und berechnet für NX
[TB5] 294:  (0<=NX<=NNX) die X-Koordinate des
[PB4] 295:  NX-ten Teilungspunktes}
[MU0] 296: BEGIN
[UD6] 297:   X:=I2D[1].X+NX/NNX*(I2D[2].X-I2D[1].X);
[ZN0] 298: END;
[Y86] 299: PROCEDURE PartitionY(I2D: Interval2D;
[US5] 300:   NY,NNY: INTEGER; VAR Y: EXTENDED);
[5T6] 301: {Zerlegt das Y-Intervall von I2D in NNY
[M06] 302:  gleiche Teile und berechnet für NY
[3C5] 303:  (0<=NY<=NNY) die Y-Koordinate des

```

Listing 1. Die Unit »u1.pas« stellt grundlegende Funktionen zur Erzeugung von 2D-Grafik zur Verfügung.


```
[TB4] 304: NY-ten Teilungspunktes]
[MU0] 305: BEGIN
[9E6] 306: Y:=I2D[1].Y+NY/NNY*(I2D[2].Y-I2D[1].Y);
[ZN0] 307: END;
[XG3] 308: PROCEDURE RandomPt2D
[KW4] 309: (I2D: Interval2D; VAR P: Pt2D);
[E37] 310: {Berechnet Zufallspunkt im Intervall I2D}
[MU0] 311: BEGIN
[CS6] 312: P.X:=I2D[1].X+RANDOM*(I2D[2].X-I2D[1].X);
[TS6] 313: P.Y:=I2D[1].Y+RANDOM*(I2D[2].Y-I2D[1].Y);
[ZN0] 314: END;
[MU0] 315: BEGIN
[YN2] 316: ScrFile:=FALSE;
[1K2] 317: DefPt2D(0,0,0,0,0,0,0,0);
[KN0] 318: END.
```

Listing 1. Die Unit »u1.pas« stellt grundlegende Funktionen zur Erzeugung von 2D-Grafik zur Verfügung.

```
[6C3] 1: {Programm: cs2d.pas
[K36] 2: Funktion: Demonstriert das Zeichnen
[0B4] 3: im Weltkoordinatensystem
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[N62] 8: PROGRAM CS2D;
[H63] 9: USES Crt, Graph, U1;
[C23] 10: VAR I2D: Interval2D;
[MU0] 11: BEGIN
[A86] 12: NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 13: InitializeGraphic;
[BW4] 14: IntPol:=FALSE; NumbIntPol:=4;
[SD3] 15: DefI2D(-3,-4,10,10, WI2D);
[3S3] 16: DefI2D(1,7,7,1, IScrScal);
[F32] 17: Parameter2D;
[C12] 18: DrawCS2D(1,2);
[6T2] 19: DefI2D(2,2,6,3, I2D);
[CB2] 20: DrawI2D(I2D,4);
[P92] 21: CloseGraphic;
[KN0] 22: END.
```

Listing 2. »cs2d.pas« erzeugt Bild 3.

```
[BF3] 1: {Programm: udraw.pas
[H65] 2: Funktion: Zeichenroutinen für
[LK2] 3: 2D- und 3D-Grafik
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[012] 8: UNIT UDRAW;
[SK1] 9: INTERFACE
[H63] 10: USES Crt, Graph, U1;
[4F2] 11: PROCEDURE Draw
[CR4] 12: (P: Pt2D; ChkOld,Chk: BOOLEAN;
[SE2] 13: L,Col: INTEGER);
[H25] 14: PROCEDURE DrawScrF(S: STRING);
[000] 15:
[4L2] 16: IMPLEMENTATION
[000] 17:
[4F2] 18: PROCEDURE Draw
[RN4] 19: (P:Pt2D; ChkOld,Chk: BOOLEAN;
[SE2] 20: L,Col: INTEGER);
[PW2] 21: VAR Nub: INTEGER;
[H22] 22: PScr: PtScr;
[MU0] 23: BEGIN
[RH4] 24: IF ScrFile=FALSE THEN BEGIN
[A04] 25: Transformation(P, PScr);
[RG2] 26: If L=0 THEN BEGIN
[184] 27: MoveToPtScr(PScr); EXIT;
[ZN0] 28: END;
[SC3] 29: IF ChkOld AND CHK THEN
[LP3] 30: DrawToPtScr(PScr,Col)
[9F6] 31: ELSE IF NOT WP THEN MoveToPtScr(PScr)
[4X4] 32: ELSE BEGIN MoveToPtScr(PScr);
[E34] 33: PutPtScr(PScr,Col); END;
[ZN0] 34: END;
[TA4] 35: IF ScrFile=TRUE THEN BEGIN
[A04] 36: Transformation(P, PScr);
[RG2] 37: If L=0 THEN BEGIN
[ZM1] 38: Nub:=Col;
[7A5] 39: Write (ScrF, Nub,PScr.X,PScr.Y);
[184] 40: MoveToPtScr(PScr); EXIT;
[ZN0] 41: END;
[B94] 42: IF ChkOld AND CHK THEN BEGIN
[YG1] 43: Nub:=Col;
[V85] 44: Write(ScrF, Nub,PScr.X,PScr.Y);
[LP3] 45: DrawToPtScr(PScr,Col)
```

```
[4M0] 46: END
[864] 47: ELSE IF NOT WP THEN BEGIN
[ZM1] 48: Nub:=-Col;
[V85] 49: Write(ScrF, Nub,PScr.X,PScr.Y);
[N33] 50: MoveToPtScr(PScr)
[3D1] 51: END ELSE
[MU0] 52: BEGIN
[693] 53: MoveToPtScr(PScr);
[FC3] 54: PutPtScr(PScr,Col);
[ZN0] 55: END;
[ZN0] 56: END;
[ZN0] 57: END;
[000] 58:
[H25] 59: PROCEDURE DrawScrF(S: STRING);
[A64] 60: VAR ScrF: FILE OF INTEGER;
[AW2] 61: Number: INTEGER;
[H22] 62: PScr: PtScr;
[MU0] 63: BEGIN
[8K2] 64: Assign(ScrF,S);
[802] 65: Reset(ScrF);
[7D4] 66: WHILE NOT EOF(ScrF) DO BEGIN
[DB5] 67: READ(ScrF, Number,PScr.X,PScr.Y);
[YN2] 68: CASE Number<0 OF
[X64] 69: TRUE: MoveToPtScr(PScr);
[NE5] 70: FALSE: DrawToPtScr(PScr,Number);
[ZN0] 71: END;
[ZN0] 72: END;
[ZV1] 73: Close(ScrF);
[ZN0] 74: END;
[KN0] 75: END.
```

Listing 3. Die Unit »udraw.pas« benötigen Sie auch für die 3D-Grafik.

```
[Z44] 1: {Programm: ucurve2d.pas
[135] 2: Funktion: Objekt zum Zeichnen
[VA4] 3: zweidimensionaler Kurven
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[HE2] 8: UNIT UCURVE2D;
[SK1] 9: INTERFACE
[PE4] 10: USES Crt, Graph, U1, UDRAW;
[ST0] 11: TYPE
[EP2] 12: Curve2DT = OBJECT
[DH2] 13: PROCEDURE TToP
[4Z5] 14: (T: EXTENDED; VAR P: Pt2D); VIRTUAL;
[EP2] 15: PROCEDURE Check
[DT5] 16: (P: Pt2D; VAR Chk: BOOLEAN); VIRTUAL;
[TN3] 17: PROCEDURE DrawCurve2D
[0H5] 18: (IID: Interval1D; LL,Col: INTEGER);
[ZN0] 19: END;
[000] 20:
[4L2] 21: IMPLEMENTATION
[000] 22:
[D54] 23: PROCEDURE Curve2DT.TToP
[9E4] 24: (T: EXTENDED; VAR P: Pt2D);
[FK1] 25: BEGIN END;
[X34] 26: PROCEDURE Curve2DT.Check
[884] 27: (P: Pt2D; VAR Chk: BOOLEAN);
[FK1] 28: BEGIN END;
[V95] 29: PROCEDURE Curve2DT.DrawCurve2D
[0H5] 30: (IID: Interval1D; LL,Col: INTEGER);
[0Y2] 31: VAR L,CN: INTEGER;
[X94] 32: Step,T,T1,T2,TI: EXTENDED;
[314] 33: ChkOld,Chk,ChkI: BOOLEAN;
[HT1] 34: P,PI: Pt2D;
[MU0] 35: BEGIN
[AY4] 36: Step:=1/LL*(IID[2].X-IID[1].X);
[BX2] 37: T:=IID[1].X-Step;
[FB2] 38: ChkOld:=FALSE;
[FS3] 39: FOR L:=0 TO LL DO BEGIN
[LP1] 40: T:=T+Step;
[LU1] 41: TToP(T, P);
[652] 42: Check(P, Chk);
[UT5] 43: IF (L>0) AND IntPol AND (ChkOld<>Chk)
[ZN1] 44: THEN BEGIN
[HV2] 45: T1:=T-Step; T2:=T;
[6X0] 46: CN:=0;
[DS4] 47: WHILE CN<NumbIntPol DO BEGIN
[902] 48: TI:=(T1+T2)/2;
[Z52] 49: TToP(TI, PI);
[8D2] 50: Check(PI, ChkI);
[0H4] 51: IF (ChkOld=ChkI) THEN T1:=TI
[K02] 52: ELSE T2:=TI;
[SA1] 53: CN:=CN+1;
[ZN0] 54: END;
[502] 55: ChkI:=TRUE;
[F84] 56: IF ChkOld THEN TToP(T1, PI)
[5X2] 57: ELSE TToP(T2, PI);
```



```
[ZU4] 58: Draw(PI,ChkOld,ChkI,LL+1,Col);
[B52] 59: ChkOld:=ChkI;
[ZN0] 60: END;
[X04] 61: Draw(P,ChkOld,Chk,L,Col);
[712] 62: ChkOld:=Chk;
[ZN0] 63: END;
[ZN0] 64: END;
[KN0] 65: END.
```

Listing 4. Das Objekt »Curve2dt« wird in der Unit »ucurve2d.pas« definiert.

```
[X14] 1: {Programm: ucheck2d.pas
[F06] 2: Funktion: Ausschneiden von Gebieten
[364] 3: aus dem 2D-Weltintervall
[G64] 4: Sprache: Turbo Pascal 6.0
[GL3] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[9C2] 8: UNIT UCHECK2D;
[SK1] 9: INTERFACE
[7W4] 10: USES Crt, Graph, U1, UCURVE2D;
[ST0] 11: TYPE
[TA4] 12: Check2DT = OBJECT(Curve2DT)
[XV2] 13: KofCut: INTEGER;
[DC2] 14: {P1,P2,M: Pt2D;
[162] 15: R: EXTENDED;}
[5N2] 16: I1D: Interval1D;
[BN2] 17: I2D: Interval2D;
[EP2] 18: PROCEDURE Check
[DT5] 19: (P: Pt2D; VAR Chk: BOOLEAN); VIRTUAL;
[ZN0] 20: END;
[000] 21:
[4L2] 22: IMPLEMENTATION
[N14] 23: PROCEDURE Check2DT.Check
[884] 24: (P: Pt2D; VAR Chk: BOOLEAN);
[U65] 25: VAR I2Dchk1,I2Dchk2: Interval2D;
[X63] 26: I1,I2,IC,BV: BOOLEAN;
[MU0] 27: BEGIN
[9S3] 28: DefI2D(-9,-2,9,2, I2Dchk1);
[PR3] 29: DefI2D(-2,-9,2,9, I2Dchk2);
[SE4] 30: I1:=InInterval2D(I2Dchk1,P);
[WE4] 31: I2:=InInterval2D(I2Dchk2,P);
[7L3] 32: IC:=InCircle(O2D,P, 6);
[1Y1] 33: Chk:=FALSE;
[2F2] 34: CASE KofCut OF
[M02] 35: 0: Chk:=TRUE;
[0C5] 36: 1: IF I1 OR I2 OR IC THEN Chk:=TRUE;
[GF3] 37: 2: IF NOT (I1 OR I2 OR IC)
[8M2] 38: THEN Chk:=TRUE;
[MT5] 39: 3: IF I1 AND I2 AND IC THEN Chk:=TRUE;
[4V3] 40: 4: IF NOT (I1 AND I2 AND IC)
[8M2] 41: THEN Chk:=TRUE;
[0V5] 42: 5: IF (I1 OR I2) AND IC THEN Chk:=TRUE;
[TS3] 43: 6: IF NOT ((I1 OR I2) AND IC)
[8M2] 44: THEN Chk:=TRUE;
[VS5] 45: 7: IF (I1 AND I2) OR IC THEN Chk:=TRUE;
[CN3] 46: 8: IF NOT ((I1 AND I2) OR IC)
[8M2] 47: THEN Chk:=TRUE;
[HT3] 48: 9: IF (I1 AND I2) OR NOT IC
[8M2] 49: THEN Chk:=TRUE;
[BP4] 50: 10: IF NOT ((I1 AND I2) OR NOT IC)
[8M2] 51: THEN Chk:=TRUE;
[KL2] 52: 11: IF (I1 AND I2) OR
[7X3] 53: (NOT (I1 OR I2) AND IC) OR
[WC3] 54: (NOT IC AND (I1 OR I2))
[8M2] 55: THEN Chk:=TRUE;
[ZM3] 56: 12: IF NOT ((I1 AND I2) OR
[7X3] 57: (NOT (I1 OR I2) AND IC) OR
[5E3] 58: (NOT IC AND (I1 OR I2)))
[8M2] 59: THEN Chk:=TRUE;
[ZN0] 60: END;
[ZN0] 61: END;
[KN0] 62: END.
```

Listing 5. Das Objekt »check2dt« in der Unit »ucheck2d.pas« stellt fest, in welchen Teilen des Weltintervalls gezeichnet werden soll.

```
[VR3] 1: {Programm: udraw2d.pas
[TT5] 2: Funktion: Objekte zum Zeichnen von
[EY2] 3: Linien und Kreisen
[G64] 4: Sprache: Turbo Pascal 6.0
[GL3] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[P52] 8: UNIT UDRAW2D;
[SK1] 9: INTERFACE
```

```
[ZT4] 10: USES Crt, Graph, U1, UCHECK2D;
[ST0] 11: TYPE
[Z54] 12: Line2DT = OBJECT(Check2DT)
[YM1] 13: P1,P2: Pt2D;
[5V6] 14: CONSTRUCTOR Init(KofCutInit: INTEGER);
[DH2] 15: PROCEDURE TToP
[4Z5] 16: (T: EXTENDED; VAR P: Pt2D); VIRTUAL;
[4F2] 17: PROCEDURE Draw
[AA6] 18: (P1Init,P2Init: Pt2D; LL,Col: INTEGER);
[ZN0] 19: END;
[GG4] 20: Circle2DT = OBJECT(Check2DT)
[7Y1] 21: R: EXTENDED;
[151] 22: M: Pt2D;
[5V6] 23: CONSTRUCTOR Init(KofCutInit: INTEGER);
[DH2] 24: PROCEDURE TToP
[4Z5] 25: (T: EXTENDED; VAR P: Pt2D); VIRTUAL;
[4F2] 26: PROCEDURE Draw
[M05] 27: (Minit: Pt2D; Rinit: EXTENDED;
[EU2] 28: LL,Col: INTEGER);
[ZN0] 29: END;
[LR4] 30: Line2DGridT = OBJECT(Line2DT)
[413] 31: CONSTRUCTOR Init
[MN3] 32: (KofCutInit: INTEGER;
[ZG3] 33: I2DInit: Interval2D);
[FF6] 34: PROCEDURE Draw(NNX,NNY,LL,Col: INTEGER);
[ZN0] 35: END;
[5U4] 36: Line2DMoireT = OBJECT(Line2DT)
[413] 37: CONSTRUCTOR Init
[MN3] 38: (KofCutInit: INTEGER;
[ZG3] 39: I2DInit: Interval2D);
[FF6] 40: PROCEDURE Draw(NNX,NNY,LL,Col: INTEGER);
[ZN0] 41: END;
[LA5] 42: Circle2DGridT = OBJECT(Circle2DT)
[413] 43: CONSTRUCTOR Init
[MN3] 44: (KofCutInit: INTEGER;
[ZG3] 45: I2DInit: Interval2D);
[FF6] 46: PROCEDURE Draw(NNX,NNY,LL,Col: INTEGER);
[ZN0] 47: END;
[AD3] 48: VAR Line2D: Line2DT;
[593] 49: Circle2D: Circle2DT;
[B34] 50: Line2DGrid: Line2DGridT;
[XP4] 51: Circle2DGrid: Circle2DGridT;
[BF4] 52: Line2DMoire: Line2DMoireT;
[000] 53:
[4L2] 54: IMPLEMENTATION
[000] 55:
[2E4] 56: CONSTRUCTOR Line2DT.Init
[1U3] 57: (KofCutInit: INTEGER);
[MU0] 58: BEGIN
[TG3] 59: KofCut:=KofCutInit;
[ZN0] 60: END;
[FU3] 61: PROCEDURE Line2DT.TToP
[9E4] 62: (T: EXTENDED; VAR P: Pt2D);
[MU0] 63: BEGIN
[CR3] 64: P.X:=P1.X+T*(P2.X-P1.X);
[RR3] 65: P.Y:=P1.Y+T*(P2.Y-P1.Y);
[ZN0] 66: END;
[6S3] 67: PROCEDURE Line2DT.Draw
[AA6] 68: (P1Init,P2Init: Pt2D; LL,Col: INTEGER);
[MU0] 69: BEGIN
[9U3] 70: I1D[1].X:=0; I1D[2].X:=1;
[RV1] 71: P1:=P1Init;
[XV1] 72: P2:=P2Init;
[5W3] 73: DrawCurve2D(I1D,LL,Col);
[ZN0] 74: END;
[1P4] 75: CONSTRUCTOR Circle2DT.Init
[1U3] 76: (KofCutInit: INTEGER);
[MU0] 77: BEGIN
[TG3] 78: KofCut:=KofCutInit;
[ZN0] 79: END;
[G84] 80: PROCEDURE Circle2DT.TToP
[9E4] 81: (T: EXTENDED; VAR P: Pt2D);
[MU0] 82: BEGIN
[UR2] 83: P.X:=M.X+R*COS(T);
[MR2] 84: P.Y:=M.Y+R*SIN(T);
[ZN0] 85: END;
[T54] 86: PROCEDURE Circle2DT.Draw
[M05] 87: (Minit: Pt2D; Rinit: EXTENDED;
[EU2] 88: LL,Col: INTEGER);
[MU0] 89: BEGIN
[RB4] 90: I1D[1].X:=0; I1D[2].X:=2*PI;
[LF1] 91: M:=Minit;
[ZG1] 92: R:=Rinit;
[5W3] 93: DrawCurve2D(I1D,LL,Col);
[ZN0] 94: END;
[E45] 95: CONSTRUCTOR Line2DGridT.Init
[F57] 96: (KofCutInit: INTEGER; I2DInit: Interval2D);
[MU0] 97: BEGIN
[TG3] 98: KofCut:=KofCutInit;
[X12] 99: I2D:=I2DInit;
[ZN0] 100: END;
[KF4] 101: PROCEDURE Line2DGridT.Draw
[1C4] 102: (NNX,NNY,LL,Col: INTEGER);
[183] 103: VAR NX,NY: INTEGER;
[MU0] 104: BEGIN
```

Listing 6. Die Unit »udraw2d.pas« enthält Objekte zum Zeichnen von Linien und Kreisen.


```
[2B4] 105: FOR NX:=1 TO NNX-1 DO BEGIN
[4L4] 106:   PartitionX(I2D,NX,NNX, P1.X);
[KE2] 107:   P1.Y:=I2D[1].Y;
[6L4] 108:   PartitionX(I2D,NX,NNX, P2.X);
[TE2] 109:   P2.Y:=I2D[2].Y;
[FA4] 110:   Line2DT.Draw(P1,P2,LL,Col);
[4X0] 111:   Stop;
[ZN0] 112: END;
[6B4] 113: FOR NY:=1 TO NNY-1 DO BEGIN
[FL4] 114:   PartitionY(I2D,NY,NNY, P1.Y);
[EE2] 115:   P1.X:=I2D[1].X;
[HL4] 116:   PartitionY(I2D,NY,NNY, P2.Y);
[NE2] 117:   P2.X:=I2D[2].X;
[FA4] 118:   Line2DT.Draw(P1,P2,LL,Col);
[4X0] 119:   Stop;
[ZN0] 120: END;
[ZN0] 121: END;
[395] 122: CONSTRUCTOR Line2DMoireT.Init
[277] 123: (KofCutInit:INTEGER;I2DInit: Interval2D);
[MU0] 124: BEGIN
[3G3] 125:   KofCut:=KofCutInit;
[X12] 126:   I2D:=I2DInit;
[ZN0] 127: END;
[DT4] 128: PROCEDURE Line2DMoireT.Draw
[1C4] 129:   (NNX,NNY,LL,Col: INTEGER);
[183] 130: VAR NX,NY: INTEGER;
[MU0] 131: BEGIN
[Z34] 132:   FOR NX:=0 TO NNX DO BEGIN
[4L4] 133:     PartitionX(I2D,NX,NNX, P1.X);
[KE2] 134:     P1.Y:=I2D[1].Y;
[C85] 135:     PartitionX(I2D,NX,NNX, P2.X);
[TE2] 136:     P2.Y:=I2D[2].Y;
[FA4] 137:     Line2DT.Draw(P1,P2,LL,Col);
[4X0] 138:     Stop;
[ZN0] 139:   END;
[344] 140:   FOR NY:=0 TO NNY DO BEGIN
[FL4] 141:     PartitionY(I2D,NY,NNY, P1.Y);
[EE2] 142:     P1.X:=I2D[1].X;
[U85] 143:     PartitionY(I2D,NNY,NNY, P2.Y);
[NE2] 144:     P2.X:=I2D[2].X;
[FA4] 145:     Line2DT.Draw(P1,P2,LL,Col);
[4X0] 146:     Stop;
[ZN0] 147:   END;
[ZN0] 148: END;
[DD5] 149: CONSTRUCTOR Circle2DGridT.Init
[P57] 150: (KofCutInit:INTEGER; I2DInit:Interval2D);
[MU0] 151: BEGIN
[3G3] 152:   KofCut:=KofCutInit;
[X12] 153:   I2D:=I2DInit;
[ZN0] 154: END;
[5W4] 155: PROCEDURE Circle2DGridT.Draw
[1C4] 156:   (NNX,NNY,LL,Col: INTEGER);
[183] 157: VAR NX,NY: INTEGER;
[N93] 158:   X,Y,LX,LY: EXTENDED;
[751] 159:   P: Pt2D;
[MU0] 160: BEGIN
[9M4] 161:   LX:=(I2D[2].X-I2D[1].X)/NNX;
[NM4] 162:   LY:=(I2D[2].Y-I2D[1].Y)/NNY;
[6W4] 163:   R:=LX/2; IF LY<LX THEN R:=LY/2;
[0B4] 164:   FOR NX:=0 TO NNX-1 DO BEGIN
[444] 165:     PartitionX(I2D,NX,NNX, X);
[SV1] 166:     M.X:=X+LX/2;
[4B4] 167:     FOR NY:=0 TO NNY-1 DO BEGIN
[C44] 168:       PartitionY(I2D,NY,NNY, Y);
[1W1] 169:       M.Y:=Y+LY/2;
[794] 170:       Circle2DT.Draw(M,R,LL,Col);
[4X0] 171:       Stop;
[ZN0] 172:     END;
[ZN0] 173:   END;
[ZN0] 174: END;
[KN0] 175: END.
```

Listing 6. Die Unit »udraw2d.pas« enthält Objekte zum Zeichnen von Linien und Kreisen.

```
[AE3] 1: {Programm: line1.pas
[X36] 2: Funktion: Demonstriert das Zeichnen
[0B4] 3: im Weltkoordinatensystem
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[RE2] 8: PROGRAM LINE1;
[CL4] 9: USES Crt, Graph, U1, UDRAW2D;
[GP2] 10: PROCEDURE Draw;
[1U2] 11: VAR I,N: INTEGER;
[YM1] 12:   P1,P2: Pt2D;
[BN2] 13:   I2D: Interval2D;
[MU0] 14: BEGIN
[GL1] 15:   WP:=TRUE;
[5M3] 16:   FOR I:=0 TO 12 DO BEGIN
[9G2] 17:     Line2D.Init(I);
[SC2] 18:     DrawI2D(WI2D,1);
[4P3] 19:   FOR N:=1 TO 200 DO BEGIN
```

```
[T43] 20:     RandomPt2D(WI2D,P1);
[W43] 21:     RandomPt2D(WI2D,P2);
[BT3] 22:     Line2D.Draw(P1,P2,12,I+1);
[ZN0] 23:   END;
[TN1] 24:   Delay(2000);
[VW3] 25:   IF I<12 THEN ClearDevice;
[4X0] 26:   Stop;
[ZN0] 27: END;
[ZN0] 28: END;
[585] 29: {-----}
[MU0] 30: BEGIN
[A86] 31:   NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 32:   InitializeGraphic;
[BW4] 33:   IntPol:=FALSE; NumbIntPol:=4;
[VM3] 34:   DefI2D(-10,-10,10,10, WI2D);
[BS3] 35:   DefI2D(1,9,9,1, IScrScal);
[F32] 36:   Parameter2D;
[B12] 37:   DrawCS2D(1,1);
[FU0] 38:   Draw;
[P92] 39:   CloseGraphic;
[KN0] 40: END.
```

Listing 7. »line1.pas« erzeugt Zufallslinien.

```
[BE3] 1: {Programm: line2.pas
[ZN6] 2: Funktion: Zeichnen von gitterförmigen
[831] 3: Linien
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[TE2] 8: PROGRAM LINE2;
[CL4] 9: USES Crt, Graph, U1, UDRAW2D;
[GP2] 10: PROCEDURE Draw;
[ZH2] 11: VAR I: INTEGER;
[MU0] 12: BEGIN
[GP1] 13:   WP:=FALSE;
[3F3] 14:   FOR I:=0 TO 5 DO BEGIN
[SC2] 15:     DrawI2D(WI2D,1);
[7D4] 16:     Line2DGrid.Init(2*I+1,WI2D);
[CD4] 17:     Line2DGrid.Draw(80,60,18,I+1);
[9D4] 18:     Line2DGrid.Init(2*I+2,WI2D);
[DD4] 19:     Line2DGrid.Draw(80,60,18,I+2);
[TN1] 20:     Delay(2000);
[KS3] 21:     IF I<5 THEN ClearDevice;
[ZN0] 22:   END;
[ZN0] 23: END;
[585] 24: {-----}
[MU0] 25: BEGIN
[A86] 26:   NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 27:   InitializeGraphic;
[5W4] 28:   IntPol:=TRUE; NumbIntPol:=4;
[VM3] 29:   DefI2D(-10,-10,10,10, WI2D);
[0Z3] 30:   DefI2D(0,10,10,0, IScrScal);
[F32] 31:   Parameter2D;
[FU0] 32:   Draw;
[P92] 33:   CloseGraphic;
[KN0] 34: END.
```

Listing 8. Gitterförmige Linien zeichnen Sie mit »line2.pas«.

```
[0M3] 1: {Programm: moirel.pas
[W96] 2: Funktion: Zeichnen von Moiré-Mustern
[G64] 3: Sprache: Turbo Pascal 6.0
[G13] 4: Autor: Kurt Endl
[585] 5: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 6:
[UP2] 7: PROGRAM MOIRE1;
[CL4] 8: USES Crt, Graph, U1, UDRAW2D;
[GP2] 9: PROCEDURE Draw;
[ZH2] 10: VAR I: INTEGER;
[MU0] 11: BEGIN
[GL1] 12:   WP:=TRUE;
[3F3] 13:   FOR I:=0 TO 5 DO BEGIN
[SC2] 14:     DrawI2D(WI2D,1);
[ZF4] 15:     Line2DMoire.Init(2*I+1,WI2D);
[YR4] 16:     Line2DMoire.Draw(30,30,18,I+1);
[ZG4] 17:     Line2DMoire.Init(2*I+2,WI2D);
[0S4] 18:     Line2DMoire.Draw(30,30,18,I+2);
[TN1] 19:     Delay(2000);
[KS3] 20:     IF I<5 THEN ClearDevice;
[ZN0] 21:   END;
[ZN0] 22: END;
[585] 23: {-----}
[MU0] 24: BEGIN
[A86] 25:   NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 26:   InitializeGraphic;
[5W4] 27:   IntPol:=TRUE; NumbIntPol:=4;
[VM3] 28:   DefI2D(-10,-10,10,10, WI2D);
[0Z3] 29:   DefI2D(0,10,10,0, IScrScal);
```



```
[F32] 30: Parameter2D;
[FU0] 31: Draw;
[P92] 32: CloseGraphic;
[KN0] 33: END.
```

Listing 9. Moiré-Muster zeichnen Sie mit »moire1.pas«.

```
[XP3] 1: {Programm: circle1.pas
[YB6] 2: Funktion: Zeichnen von Zufallskreisen
[G64] 3: Sprache: Turbo Pascal 6.0
[G13] 4: Autor: Kurt Endl
[585] 5: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 6:
[6T2] 7: PROGRAM CIRCLE1;
[CL4] 8: USES Crt, Graph, U1, UDRAW2D;
[GP2] 9: PROCEDURE Draw;
[N94] 10: CONST Col1=1; Col2=2; Col3=3;
[1U2] 11: VAR I,N: INTEGER;
[7Y1] 12: R: EXTENDED;
[151] 13: M: Pt2D;
[BN2] 14: I2D: Interval2D;
[MU0] 15: BEGIN
[GP1] 16: WP:=FALSE;
[533] 17: DefI2D(-5,-5,5,5, I2D);
[3F3] 18: FOR I:=0 TO 5 DO BEGIN
[SC2] 19: DrawI2D(WI2D,1);
[UR2] 20: Circle2D.Init(I);
[2P3] 21: FOR N:=1 TO 100 DO BEGIN
[5U2] 22: RandomPt2D(I2D,M);
[HR0] 23: R:=5;
[Z43] 24: Circle2D.Init(2*I+1);
[BY3] 25: Circle2D.Draw(M,R,22,I+1);
[353] 26: Circle2D.Init(2*I+2);
[CY3] 27: Circle2D.Draw(M,R,22,I+2);
[ZN0] 28: END;
[TN1] 29: Delay(2000);
[KS3] 30: IF I<5 THEN ClearDevice;
[4X0] 31: Stop;
[ZN0] 32: END;
[ZN0] 33: END;
[585] 34: {-----}
[MU0] 35: BEGIN
[A86] 36: NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 37: InitializeGraphic;
[BW4] 38: IntPol:=FALSE; NumbIntPol:=4;
[VM3] 39: DefI2D(-10,-10,10,10, WI2D);
[0Z3] 40: DefI2D(0,10,10,0, IScrScal);
[F32] 41: Parameter2D;
[FU0] 42: Draw;
[P92] 43: CloseGraphic;
[KN0] 44: END.
```

Listing 10. Zufallskreise zeichnen Sie mit »circle1.pas«.

```
[0R3] 1: {Programm: circle2.pas
[H56] 2: Funktion: Zeichnen von Kreisgittern
[G64] 3: Sprache: Turbo Pascal 6.0
[G13] 4: Autor: Kurt Endl
[585] 5: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 6:
[AT2] 7: PROGRAM CIRCLE2;
[CL4] 8: USES Crt, Graph, U1, UDRAW2D;
[GP2] 9: PROCEDURE Draw;
[ZH2] 10: VAR I: INTEGER;
[MU0] 11: BEGIN
[GP1] 12: WP:=FALSE;
[3F3] 13: FOR I:=0 TO 5 DO BEGIN
[SC2] 14: DrawI2D(WI2D,1);
[GN4] 15: Circle2DGrid.Init(2*I+1,WI2D);
[RN4] 16: Circle2DGrid.Draw(22,22,12,I+1);
[MN4] 17: Circle2DGrid.Init(2*I+2,WI2D);
[UN4] 18: Circle2DGrid.Draw(22,22,12,I+2);
[TN1] 19: Delay(2000);
[KS3] 20: IF I<5 THEN ClearDevice;
[ZN0] 21: END;
[ZN0] 22: END;
[585] 23: {-----}
[MU0] 24: BEGIN
[A86] 25: NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 26: InitializeGraphic;
[5W4] 27: IntPol:=TRUE; NumbIntPol:=4;
[VM3] 28: DefI2D(-10,-10,10,10, WI2D);
[0Z3] 29: DefI2D(0,10,10,0, IScrScal);
[F32] 30: Parameter2D;
[FU0] 31: Draw;
[P92] 32: CloseGraphic;
[KN0] 33: END.
```

Listing 11. Das Programm »circle2.pas« erzeugt Kreisgitter.

```
[8S3] 1: {Programm: u2dgeom.pas
[A47] 2: Funktion: Werkzeuge für 2D- und 3D-Grafik
[G64] 3: Sprache: Turbo Pascal 6.0
[G13] 4: Autor: Kurt Endl
[585] 5: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 6:
[LS2] 7: UNIT U2DGEOM;
[SK1] 8: INTERFACE
[H63] 9: USES Crt, Graph, U1;
[ST0] 10: TYPE
[NB2] 11: PtPol = RECORD
[5L2] 12: R,Phi: EXTENDED;
[ZN0] 13: END;
[NB2] 14: FUNCTION Null
[D75] 15: (Z,Epsilon: EXTENDED): BOOLEAN;
[UG3] 16: PROCEDURE CartPolar
[7D4] 17: (P: Pt2D; VAR PPol: PtPol);
[223] 18: PROCEDURE SumVt2D
[474] 19: (V1,V2: Vt2D; VAR SV: Vt2D);
[H44] 20: PROCEDURE DifferenceVt2D
[854] 21: (V1,V2: Vt2D; VAR DV: Vt2D);
[DK5] 22: PROCEDURE LinearCombinationVt2D
[WS4] 23: (S1,S2: EXTENDED; V1,V2: Vt2D;
[R72] 24: VAR LC: Vt2D);
[3Y4] 25: PROCEDURE ScalarProductVt2D
[RK4] 26: (V1,V2: Vt2D; VAR SP: EXTENDED);
[244] 27: PROCEDURE DirectionVt2D
[1P6] 28: (Dgr:BOOLEAN;R,Phi:EXTENDED; VAR V:Vt2D);
[000] 29:
[4L2] 30: IMPLEMENTATION
[000] 31:
[9F7] 32: FUNCTION Null(Z,Epsilon:EXTENDED):BOOLEAN;
[MU0] 33: BEGIN
[7M3] 34: Null:=ABS(Z)<Epsilon;
[ZN0] 35: END;
[357] 36: PROCEDURE SumVt2D(V1,V2:Vt2D;VAR SV:Vt2D);
[MU0] 37: BEGIN
[VK2] 38: SV.X:=V1.X+V2.X;
[3L2] 39: SV.Y:=V1.Y+V2.Y;
[ZN0] 40: END;
[H44] 41: PROCEDURE DifferenceVt2D
[S14] 42: (V1,V2:Vt2D; VAR DV:Vt2D);
[MU0] 43: BEGIN
[5K2] 44: DV.X:=V1.X-V2.X;
[DK2] 45: DV.Y:=V1.Y-V2.Y;
[ZN0] 46: END;
[DK5] 47: PROCEDURE LinearCombinationVt2D
[WS4] 48: (S1,S2: EXTENDED; V1,V2: Vt2D;
[R72] 49: VAR LC: Vt2D);
[MU0] 50: BEGIN
[XD3] 51: LC.X:=S1*V1.X+S2*V2.X;
[2E3] 52: LC.Y:=S1*V1.Y+S2*V2.Y;
[ZN0] 53: END;
[3Y4] 54: PROCEDURE ScalarProductVt2D
[0W4] 55: (V1,V2:Vt2D; VAR SP:EXTENDED);
[MU0] 56: BEGIN
[2S3] 57: SP:=V1.X*V2.X+V1.Y*V2.Y;
[ZN0] 58: END;
[244] 59: PROCEDURE DirectionVt2D
[HY4] 60: (Dgr: BOOLEAN; R,Phi: EXTENDED;
[MZ1] 61: VAR V: Vt2D);
[SW5] 62: {Berechnet den 2D-Vektor der Länge
[063] 63: R in Richtung Phi}
[MU0] 64: BEGIN
[WB4] 65: IF Dgr THEN Phi:=Phi*PI/180;
[WF2] 66: V.X:=R*COS(Phi);
[NF2] 67: V.Y:=R*SIN(Phi);
[ZN0] 68: END;
[M64] 69: PROCEDURE CosSinToAngle
[MH5] 70: (C,S: EXTENDED; VAR Ang:EXTENDED);
[PD6] 71: {Berechnet den Winkel, der durch seinen
[4C5] 72: Cosinus und Sinus gegeben ist}
[MU0] 73: BEGIN
[6R3] 74: IF Null(C,Eps5) AND (S<0)
[ND2] 75: THEN Ang:=-PI/2;
[4W3] 76: IF Null(C,Eps5) AND (S>=0)
[SF2] 77: THEN Ang:=PI/2;
[RY2] 78: IF NOT Null(C,Eps5)
[4G3] 79: THEN Ang:=ARCTAN(S/C);
[VB5] 80: IF (C<0) AND (S<0) THEN Ang:=Ang-PI;
[DD5] 81: IF (C<0) AND (S>=0) THEN Ang:=Ang+PI;
[ZN0] 82: END;
[9R4] 83: PROCEDURE CartPolar(P: Pt2D;
[LX2] 84: VAR PPol: PtPol);
[LU6] 85: {Berechnet zu einem in kartischen Koordi-
[216] 86: naten gegebenen Punkt dessen Polar-
[T82] 87: koordinaten}
[MU0] 88: BEGIN
[7T5] 89: PPol.R:=SQRT(SQR(P.X)+SQR(P.Y));
[ZH5] 90: CosSinToAngle(P.X,P.Y, PPol.Phi);
[R22] 91: IF PPol.Phi<0
[2V4] 92: THEN PPol.Phi:=PPol.Phi+2*PI;
[ZN0] 93: END;
[KN0] 94: END.
```

Listing 12. Die Unit »u2dgeom.pas« enthält Werkzeuge für die 2D- und die 3D-Grafik.

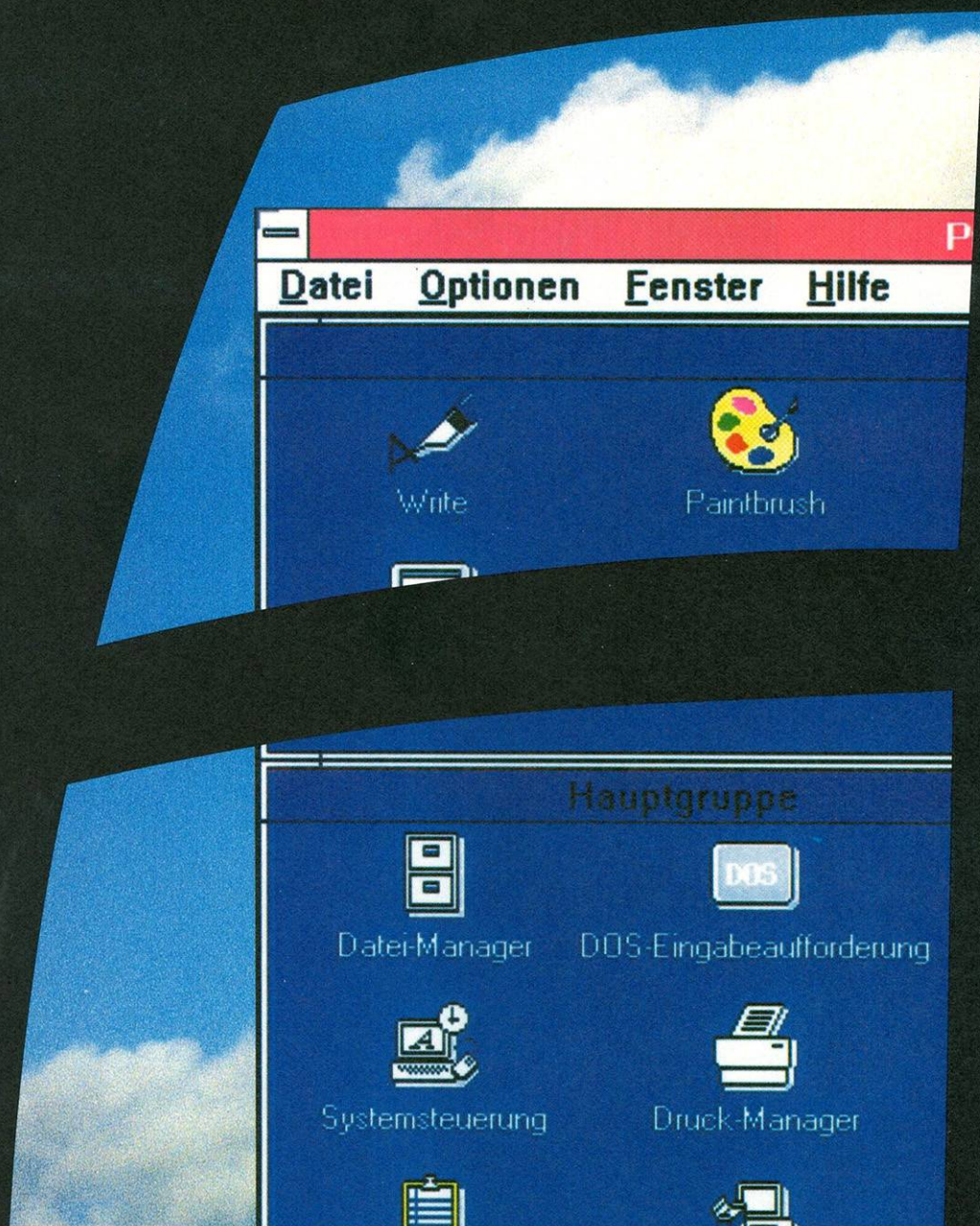

```
[SG3] 1: {Programm: uorn.pas
[FH5] 2: Funktion: Objekte zur Erzeugung
[6L2] 3: von Ornamentik
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[6T1] 8: UNIT UORN;
[SK1] 9: INTERFACE
[SL4] 10: USES Crt, Graph, U1, UDRAW2D;
[ST0] 11: TYPE
[1L2] 12: Orn2DT = OBJECT
[RC3] 13: IOrn, I2D: Interval2D;
[UZ2] 14: KX, KY: INTEGER;
[N36] 15: LIOrnX, LIOrnY, LI2DX, LI2DY: EXTENDED;
[9Y1] 16: UOrn: Vt2D;
[VF2] 17: PROCEDURE Init
[Y15] 18: (IOrnInit, I2DInit: Interval2D;
[7N4] 19: KXInit, KYInit: INTEGER);
[P74] 20: PROCEDURE CalculateOOrn
[HD5] 21: (KX, KY: INTEGER; VAR OOrn: Pt2D);
[X35] 22: PROCEDURE TransformationOrn
[HK4] 23: (OOrn, P: Pt2D; VAR P2D: Pt2D);
[PT5] 24: PROCEDURE TransformationOrnBack
[7L4] 25: (OOrn, P2D: Pt2D; VAR P: Pt2D);
[ZN0] 26: END;
[593] 27: Line2DOrnT = OBJECT
[P72] 28: Orn2D: Orn2DT;
[LL5] 29: PROCEDURE Init(Orn2DInit: Orn2DT);
[665] 30: PROCEDURE Draw(Line2D: Line2DT;
[CF4] 31: P1, P2: Pt2D; LL, Col: INTEGER);
[ZN0] 32: END;
[6D3] 33: Circle2DOrnT = OBJECT
[P72] 34: Orn2D: Orn2DT;
[LL5] 35: PROCEDURE Init(Orn2DInit: Orn2DT);
[GX5] 36: PROCEDURE Draw(Circle2D: Circle2DT;
[A76] 37: M: Pt2D; R: EXTENDED; LL, Col: INTEGER);
[ZN0] 38: END;
[DY2] 39: VAR Orn2D: Orn2DT;
[AR3] 40: Line2DOrn: Line2DOrnT;
[SF4] 41: Circle2DOrn: Circle2DOrnT;
[000] 42:
[4L2] 43: IMPLEMENTATION
[000] 44:
[ZL3] 45: PROCEDURE Orn2DT.Init
[Y15] 46: (IOrnInit, I2DInit: Interval2D;
[7N4] 47: KXInit, KYInit: INTEGER);
[MU0] 48: BEGIN
[DT2] 49: IOrn:=IOrnInit;
[X12] 50: I2D:=I2DInit;
[UA2] 51: KX:=KXInit;
[ZA2] 52: KY:=KYInit;
[935] 53: LIOrnX:=(IOrn[2].X-IOrn[1].X);
[G35] 54: LIOrnY:=(IOrn[2].Y-IOrn[1].Y);
[ZZ4] 55: LI2DX:=(I2D[2].X-I2D[1].X)/KX;
[705] 56: LI2DY:=(I2D[2].Y-I2D[1].Y)/KY;
[5M3] 57: UOrn.X:=LI2DX/LIOrnX;
[BM3] 58: UOrn.Y:=LI2DY/LIOrnY;
[ZN0] 59: END;
[U85] 60: PROCEDURE Orn2DT.CalculateOOrn
[HD5] 61: (KX, KY: INTEGER; VAR OOrn: Pt2D);
[MU0] 62: BEGIN
[2G4] 63: OOrn.X:=I2D[1].X+(KX-1)*LI2DX
[BK3] 64: +UOrn.X*(-IOrn[1].X);
[9G4] 65: OOrn.Y:=I2D[1].Y+(KY-1)*LI2DY
[LK3] 66: +UOrn.Y*(-IOrn[1].Y);
[ZN0] 67: END;
[L36] 68: PROCEDURE Orn2DT.TransformationOrn
[6H5] 69: (OOrn: Pt2D; P: Pt2D; VAR P2D: Pt2D);
[MU0] 70: BEGIN
[B24] 71: P2D.X:=OOrn.X+UOrn.X*P.X;
[H24] 72: P2D.Y:=OOrn.Y+UOrn.Y*P.Y;
[ZN0] 73: END;
[000] 74:
[3T6] 75: PROCEDURE Orn2DT.TransformationOrnBack
[7L4] 76: (OOrn, P2D: Pt2D; VAR P: Pt2D);
[MU0] 77: BEGIN
[WM4] 78: P.X:=(P2D.X-OOrn.X)/UOrn.X;
[AN4] 79: P.Y:=(P2D.Y-OOrn.Y)/UOrn.Y;
[ZN0] 80: END;
[BB4] 81: PROCEDURE Line2DOrnT.Init
[CB3] 82: (Orn2DInit: Orn2DT);
[MU0] 83: BEGIN
[VV2] 84: Orn2D:=Orn2DInit;
[ZN0] 85: END;
[AB4] 86: PROCEDURE Line2DOrnT.Draw
[6N4] 87: (Line2D: Line2DT; P1, P2: Pt2D;
[EU2] 88: LL, Col: INTEGER);
[S73] 89: VAR KX, KY: INTEGER;
[413] 90: OOrn, P11, P22: Pt2D;
[MU0] 91: BEGIN
[463] 92: WITH Orn2D DO BEGIN
[634] 93: FOR KX:=1 TO KX DO BEGIN
[A34] 94: FOR KY:=1 TO KY DO BEGIN
[P64] 95: CalculateOOrn(KX, KY, OOrn);
[755] 96: TransformationOrn(OOrn, P1, P11);
[065] 97: TransformationOrn(OOrn, P2, P22);
[494] 98: Line2D.Draw(P11, P22, LL, Col);
[ZN0] 99: END;
[ZN0] 100: END;
```

```
[ZN0] 101: END;
[ZN0] 102: END;
[GU4] 103: PROCEDURE Circle2DOrnT.Init
[CB3] 104: (Orn2DInit: Orn2DT);
[MU0] 105: BEGIN
[VV2] 106: Orn2D:=Orn2DInit;
[ZN0] 107: END;
[7U4] 108: PROCEDURE Circle2DOrnT.Draw
[BP4] 109: (Circle2D: Circle2DT; M: Pt2D;
[LW4] 110: R: EXTENDED; LL, Col: INTEGER);
[S73] 111: VAR KX, KY: INTEGER;
[WM2] 112: ROrn: EXTENDED;
[413] 113: OOrn, P11, P22: Pt2D;
[MU0] 114: BEGIN
[463] 115: WITH Orn2D DO BEGIN
[634] 116: FOR KX:=1 TO KX DO BEGIN
[A34] 117: FOR KY:=1 TO KY DO BEGIN
[P64] 118: CalculateOOrn(KX, KY, OOrn);
[Y85] 119: TransformationOrn(OOrn, M, P11);
[TP2] 120: IF UOrn.X<UOrn.Y
[XC3] 121: THEN ROrn:=R*UOrn.X
[TF3] 122: ELSE ROrn:=R*UOrn.Y;
[415] 123: Circle2D.Draw(P11, ROrn, LL, Col);
[ZN0] 124: END;
[ZN0] 125: END;
[ZN0] 126: END;
[ZN0] 127: END;
[MU0] 128: BEGIN
[1K2] 129: DefPt2D(0, 0, O2D);
[KN0] 130: END.
```

Listing 13. Objekte zur Erzeugung von Ornamenten enthält die Unit »uorn.pas«.

```
[SC3] 1: {Programm: orn1.pas
[0K5] 2: Funktion: Anwendung der Objekte
[UZ1] 3: in uorn.pas
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[7A2] 8: PROGRAM ORN1;
[Z17] 9: USES Crt, Graph, U1, UDRAW2D, U2DGEOM, UORN;
[GP2] 10: PROCEDURE Draw;
[WW1] 11: CONST NN:=12;
[XN4] 12: VAR KX, KY, I, J, N: INTEGER;
[5L2] 13: R, Phi: EXTENDED;
[MN1] 14: P1, M: Pt2D;
[RC3] 15: IOrn, I2D: Interval2D;
[MU0] 16: BEGIN
[GP1] 17: WP:=FALSE;
[7P3] 18: DefI2D(-10, -10, 10, 10, IOrn);
[WK2] 19: KX:=7; KY:=7;
[E35] 20: {Anzahl der Teilintervalle in
[WD3] 21: X- bzw. Y-Richtung}
[405] 22: Orn2D.Init(IOrn, WI2D, KX, KY);
[0L3] 23: Line2DOrn.Init(Orn2D);
[F14] 24: Circle2DOrn.Init(Orn2D);
[3F3] 25: FOR I:=0 TO 5 DO BEGIN
[SC2] 26: DrawI2D(WI2D, I);
[0F3] 27: FOR J:=1 TO 2 DO BEGIN
[W13] 28: Line2D.Init(2*I+J);
[YS3] 29: FOR N:=1 TO NN DO BEGIN
[RD2] 30: Phi:=N/NN*2*PI;
[S25] 31: DirectionVt2D(FALSE, 7, Phi, P1);
[UN4] 32: Line2DOrn.Draw(Line2D, O2D, P1,
[U11] 33: 6, I+J);
[ZN0] 34: END;
[383] 35: Circle2D.Init(2*I+J);
[YS3] 36: FOR N:=1 TO NN DO BEGIN
[RD2] 37: Phi:=N/NN*2*PI;
[FW4] 38: DirectionVt2D(FALSE, 7, Phi, M);
[XB4] 39: Circle2DOrn.Draw(Circle2D,
[UG1] 40: M, 3, 12, I+J);
[ZN0] 41: END;
[ZN0] 42: END;
[XN1] 43: Delay(3000);
[KS3] 44: IF I<5 THEN ClearDevice;
[4X0] 45: Stop;
[ZN0] 46: END;
[ZN0] 47: END;
[MU0] 48: BEGIN
[A86] 49: NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 50: InitializeGraphic;
[5W4] 51: IntPol:=TRUE; NumbIntPol:=4;
[VM3] 52: DefI2D(-10, -10, 10, 10, WI2D);
[0Z3] 53: DefI2D(0, 10, 10, 0, IScrScal);
[F32] 54: Parameter2D;
[SC2] 55: DrawI2D(WI2D, 1);
[FU0] 56: Draw;
[P92] 57: CloseGraphic;
[KN0] 58: END.
```

Listing 14. Demonstration der Funktionen aus der Unit »uorn.pas«



Das Fenster zur Windows-Welt

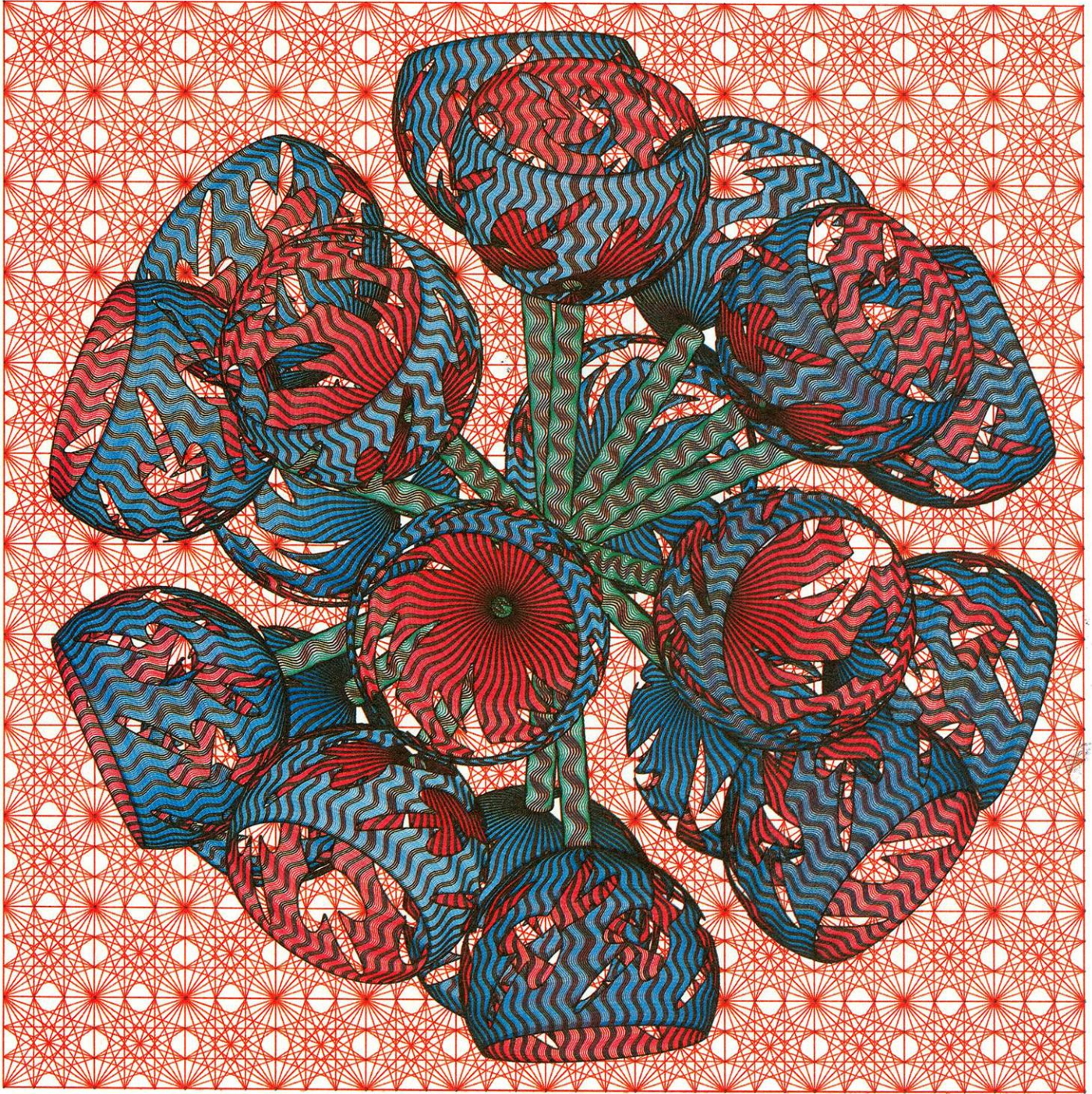
Das neue Monatsmagazin WinDOS aus der DOS-Redaktion bietet Ihnen qualifizierte und umfassende Informationen zum Thema Windows:

- aktuelle Berichterstattung über Neuheiten und Neuigkeiten auf dem Windows-Markt – wertvolle Informationsquelle für jeden Computeranwender
- fundierte Testberichte über brandheiße Windows-Hard- und -Software – unverzichtbar als Entscheidungshilfe
- Werkstätten und Kurse zu Windows und den wichtigsten Windows-Applikationen – notwendig, um Windows noch besser zu verstehen und zu beherrschen
- Grundlagenwissen und professionelle Kniffe für diejenigen, die unter Windows programmieren – unabdingbar, um Windows optimal auszureizen
- und eine Fülle von Tips, Tricks, Tabellen und Hintergrundinformationen – eine wahre Fundgrube für jeden Anwender und Programmierer.



Eine Publikation aus dem Hause





3D-Grafik

Grafik-Objekte

Mit dieser Einführung in die dreidimensionale Computergrafik erhalten Sie eine komplette Darstellung des objektorientierten Zugangs zur 3D-Grafik, inklusive ihrer Objekthierarchie. Damit sind Sie in der Lage, mit einfachen Mitteln relativ komplexe Objekte zu erzeugen.

Nach einer Einführung in die allgemeine freie Perspektive lernen Sie die Darstellung beliebig vieler, auch aufgeschnittener Kugeln, Kegel und Zylinder und das Zeichnen von geschwungenen Meridianen und Breitenkreisen kennen. Wie schon im Beitrag zur 2D-Grafik können Sie auch jetzt wieder die Interpolation und die Speicherung hinzuschalten.

Grundsätzlich ist zu der hier vorgestellten Software zu bemerken: Im Gegensatz zu den häufig beim CAD üblichen Methoden der Flächengenerierung werden hier alle 3D-Objekte analytisch dargestellt. Dies erlaubt im weiteren Ausbau insbesondere das beliebige Aufschneiden der Flächen, ferner können Sie Ornamente, Schrift und ähnliches anbringen.

● Grundbegriffe der 3D-Grafik:

Die gesamte 3D-Grafik baut auf dem 3D-Weltkoordinatensystem »WCD3D« (3D-Benutzerkoordinatensystem) auf. In diesem definieren Sie das 3D-Weltintervall »WI3D«, innerhalb

dessen Sie alle geometrischen Objekte plazieren. Wie alle 3D-Intervalle definieren Sie es durch den vorderen linken unteren und hinteren rechten oberen Eckpunkt (Bild 1).

Neben kartesischen Koordinaten werden auch sphärische (Kugelkoordinaten Bild 2) und zylindrische Koordinaten (Zylinderkoordinaten Bild 3) verwendet.

Die Unit »u3dgeom.pas« (Listing 1) enthält die benötigten Typen, Variablen und Methoden für die 3D-Grafik.

● **Die allgemeine Perspektive:** In der 3D-Grafik zeichnen Sie von einem räumlichen Objekt ein Bild. Dazu benötigen Sie eine Abbildung, welche den Raum in eine Ebene projiziert. Sie erzeugen hier eine volle Perspektive, im Gegensatz zu vielen Darstellungen, die sich auf eine Orthogonal-Projektion beschränken.

Die Definition einer Perspektive ist denkbar einfach. Sie benötigen einen Projektionspunkt »C« und eine Projektionsebene. Diese geben Sie durch einen Punkt »O« vor, den Normalenvektor »UZ« der Ebene und einen Vektor »YUP«, der die y-Achse in der Projektionsebene festlegt. Denn die Ebene hat ihr eigenes Koordinatensystem, das durch den Normalenvektor »UZ« allein noch nicht festgelegt ist (Bild 4).

Das 3D-Weltintervall WI3D wird dann in die Ebene projiziert (Bild 5). Das kleinste 2D-Intervall, das alle Bildpunkte der Ecken enthält, definiert das 2D-Weltintervall, das Sie dann, wie in der 2D-Grafik, auf dem Bildschirm abbilden. In der Unit »u3d.pas« (Listing 2) sind alle Programmteile aufgeführt, die Sie zur Erzeugung der Perspektive und der Projektion des 3D-Weltintervalls benötigen. Mit dem Programm »cs3d.pas« (Listing 3) zeichnen Sie das 3D-Weltintervall perspektivisch.

Die Perspektive definieren Sie in sphärischen Koordinaten. Wie Sie sich die Lage eines Punktes PSph = (R,Phi,Theta) hierbei vorstellen, entnehmen Sie Bild 6. Zur Veranschaulichung der Tiefenwirkung ist der »Fußboden« mit »Kacheln« versehen (Bild 7).

Bild 8 zeigt Ihnen die Objekthierarchie der 3D-Grafik. Auch hier sind bei den Objekttypen wieder nur die Methoden angeführt.

Die Logik des Objekttyps »curve3dt« (»ucurve3d.pas«, Listing 4) folgt völlig der des Objekttyps »curve2dt«. Zum Verständnis sollten Sie den Beitrag zur 2D-Grafik in dieser Ausgabe lesen. Nur die Köpfe der beiden Prozeduren »ttop« und »check« haben sich etwas verändert. Das ist nicht weiter verwunderlich, denn in diesen beiden Prozeduren steckt die ganze 3D-Grafik, also insbesondere die Perspektive.

In die Prozedur »ttop« geben Sie nach wie vor den Parameterwert »t« ein. Zu diesem wird dann der 3D-Punkt »p3d« berechnet, der anschließend in den Punkt »p« im 2D-Weltkoordinatensystem projiziert wird. Ausgegeben wird der Sehstrahl »prray« (projection ray), das heißt die Gerade vom Projektionszentrum »C« zum 3D-Punkt »p3d« und die Entfernung »distn« (distance).

In die Prozedur »check« geben Sie den Projektionsstrahl »prray« und die Entfernung »distn« ein. Ausgegeben wird mit dem Parameter »chk« die Information, ob der Punkt »p3d« sichtbar ist oder nicht.

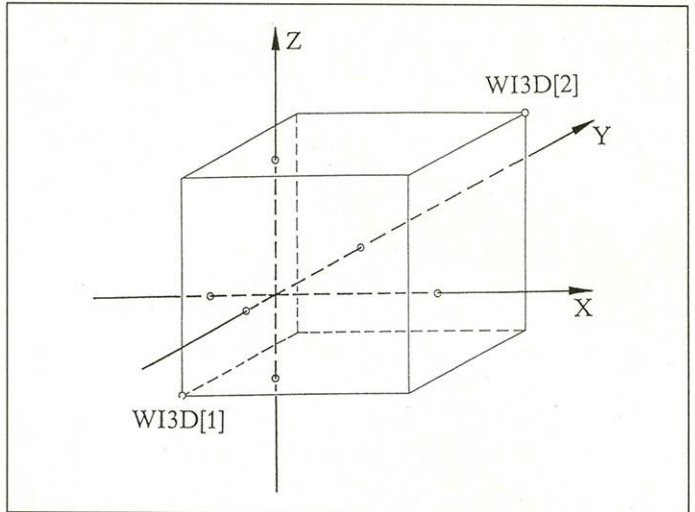


Bild 1. Das 3D-Weltkoordinatensystem mit dem 3D-Weltintervall WI3D

● **Rotationsflächen:** Die Körper Kugel, Kegel und Zylinder werden mit dem gemeinsamen Begriff Rotationsflächen bezeichnet, wobei sie sich voneinander durch den Parameter »kofrs« (Kind Of Rotation Surface = Art der Rotationsfläche) unterscheiden. Eine Rotationsfläche entsteht, wenn Sie eine Kurve um eine feste Achse drehen. So entsteht eine Kugel durch Drehung eines Halbkreises, ein Kegel durch Drehung einer Geraden, welche die Achse schneidet, und ein Zylinder durch Drehung einer zur Achse parallelen Geraden.

Dies legt nahe, jeder Rotationsfläche ein eigenes 3D-Koordinatensystem zuzuordnen. Hierbei wählen Sie als z-Achse die Drehachse. Bezeichnen Sie dann den Drehwinkel mit »U«, die Höhe auf der z-Achse mit »V«, so sind die maximalen Parametergebiete festgelegt (Bild 9). Da Sie in diesem Fall nur Teile dieser Flächen zeichnen können, wählen Sie für jede Rotationsfläche ein Parameterintervall »IUV« im maximalen Parametergebiet.

Die Konturlinie beschreiben Sie mit der Funktion:

$$D(V) = A \cdot V^2 + B$$

Die Parameter »A« und »B« haben folgende Bedeutung:

KOfRS=1 (Kugel): $A=-1$; $B=R \times R$;

KOfRS=2 (Kegel): $A=\tan^2(\alpha)$; $B=0$;

KOfRS=3 (Zylinder): $A=0$; $B=R \times R$;

In der Unit »ugs.pas« (Listing 5) ist unter anderem der Objekttyp »rst« (rs: rotation surface) definiert, zusammen mit allen Methoden, die zur Behandlung der Unit notwendig sind. Die Gesamtzahl aller zu definierenden Flächen ist mit »mnrs« bezeichnet.

Analog dem Vorgehen in der 2D-Grafik definieren wir jetzt den Objekttyp, welcher die Sichtbarkeit der Rotationsflächen regelt (Unit »ugscheck.pas«, Listing 6). Das ist ein wenig komplizierter, da Sie beliebig viele solcher Flächen zugleich darstellen möchten. Jede der Flächen kann aufge-

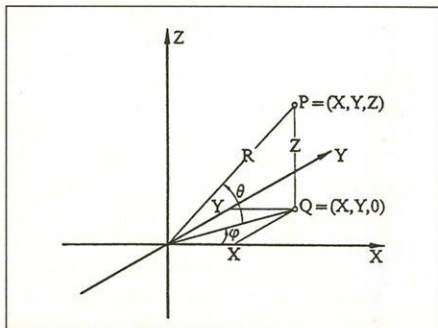


Bild 2. Das sphärische Koordinatensystem (Polarsystem)

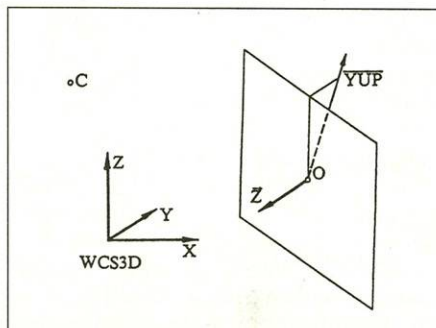


Bild 3. Zylinderkoordinaten eignen sich besonders für Rotationsflächen.

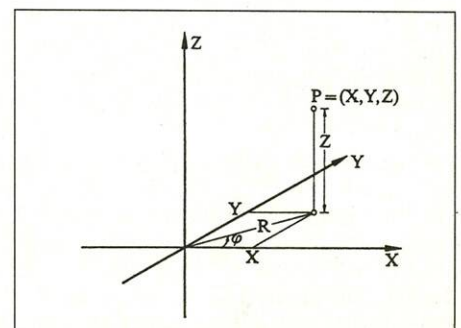


Bild 4. Die Definition einer Perspektive

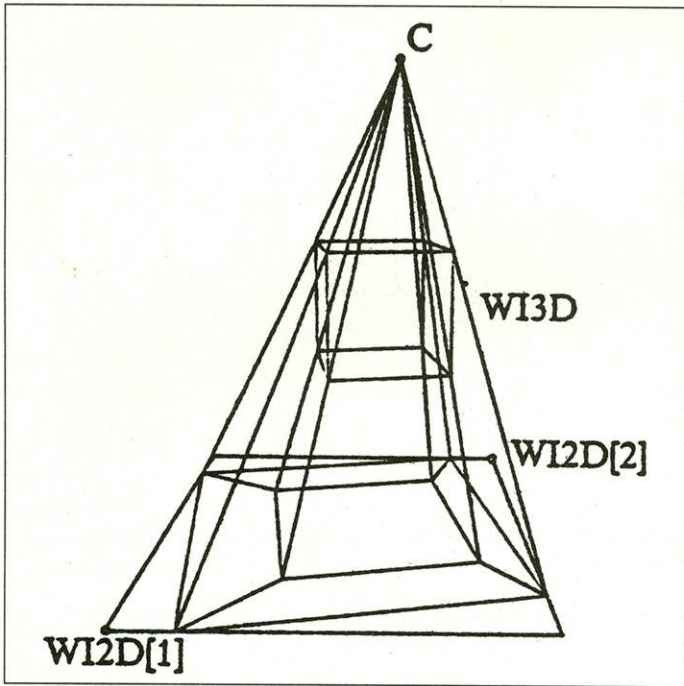


Bild 5. Das 3D-Weltintervall in der Projektionsebene

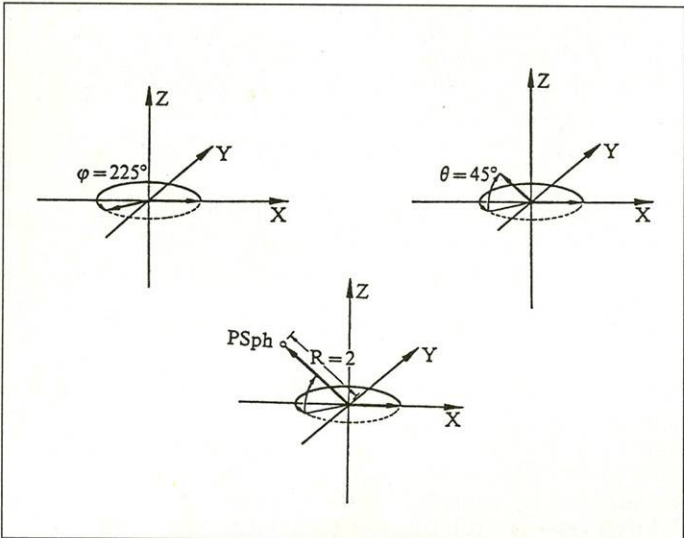


Bild 6. Die Definition eines Punktes in sphärischen Koordinaten

schnitten sein, so daß ein Punkt, der »hinten« liegt, sichtbar sein kann. Außerdem verdecken sich die Flächen im allgemeinen.

Es ist zweckmäßig, die Sichtbarkeit eines Punktes in zwei Probleme aufzuspalten:

1. Sie zeichnen nacheinander immer nur auf einer Fläche, sagen wir auf der n-ten Fläche. Dann benötigen Sie zuerst eine Prozedur, die Ihnen sagt, ob ein Punkt dieser n-ten Fläche bezüglich dieser n-ten Fläche sichtbar ist oder nicht. Diese Prozedur ist mit »rsvisibility« (visibility = Sichtbarkeit) bezeichnet.
2. Anschließend entscheiden Sie, ob ein bezüglich der n-ten Fläche sichtbarer Punkt nicht von einer der anderen Flächen verdeckt wird. Diese Aufgabe, bei der also der Sehstrahl mit allen anderen Flächen geschnitten werden muß, erledigen Sie mit der Prozedur »rshidden« (hidden = verdeckt). Diese beiden Prozeduren sind in dem Objekttyp »rsvisidd« enthalten. Beide Informationen werden dann in der eigentlichen Check-Prozedur des Objekttyps »gscheck« verarbeitet.

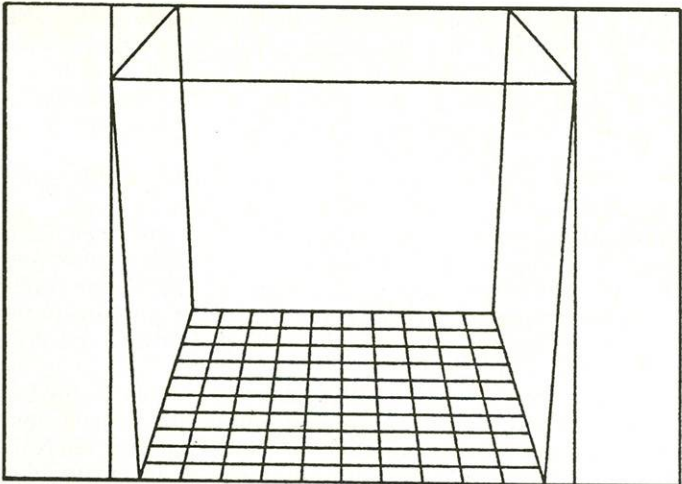


Bild 7. Die Tiefenwirkung der verwendeten Perspektive

● **Meridiane:** In der Unit »umer.pas« (Listing 7) ist der Objekttyp zum Zeichnen von Meridianen definiert. Einen Meridian erhalten Sie, indem Sie den Parameter »U« konstant halten und »V« das V-Intervall durchlaufen lassen. Um das Resultat etwas interessanter zu gestalten, lassen Sie jedoch »U« in Abhängigkeit von »V« oszillieren, wobei Sie die Oszillationsbreite durch den Faktor »oscfac« und die Oszillationsfrequenz durch den Faktor »oscfr« regeln. Dabei ist, falls die Fläche aufgeschnitten ist, vorgesehen, daß diese Oszillation an den Begrenzungen gedämpft ist. Diese Dämpfung schalten Sie mit dem Parameter »damp« ein und aus.

● **Breitenkreise:** Analog den Meridianen entstehen Breitenkreise dadurch, daß Sie »V« konstant halten und »U« laufen lassen. Ansonsten gilt für die Breitenkreise, die Sie mit »ucirc.pas« (Listing 8) erzeugen, das gleiche wie für das Zeichnen von Meridianen.

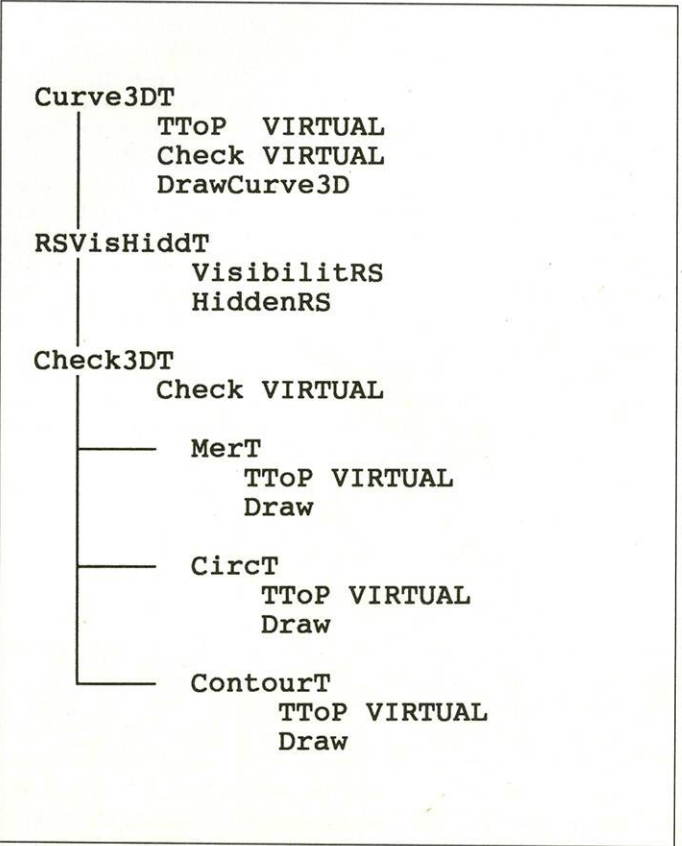


Bild 8. Die Objekthierarchie der 3D-Grafik

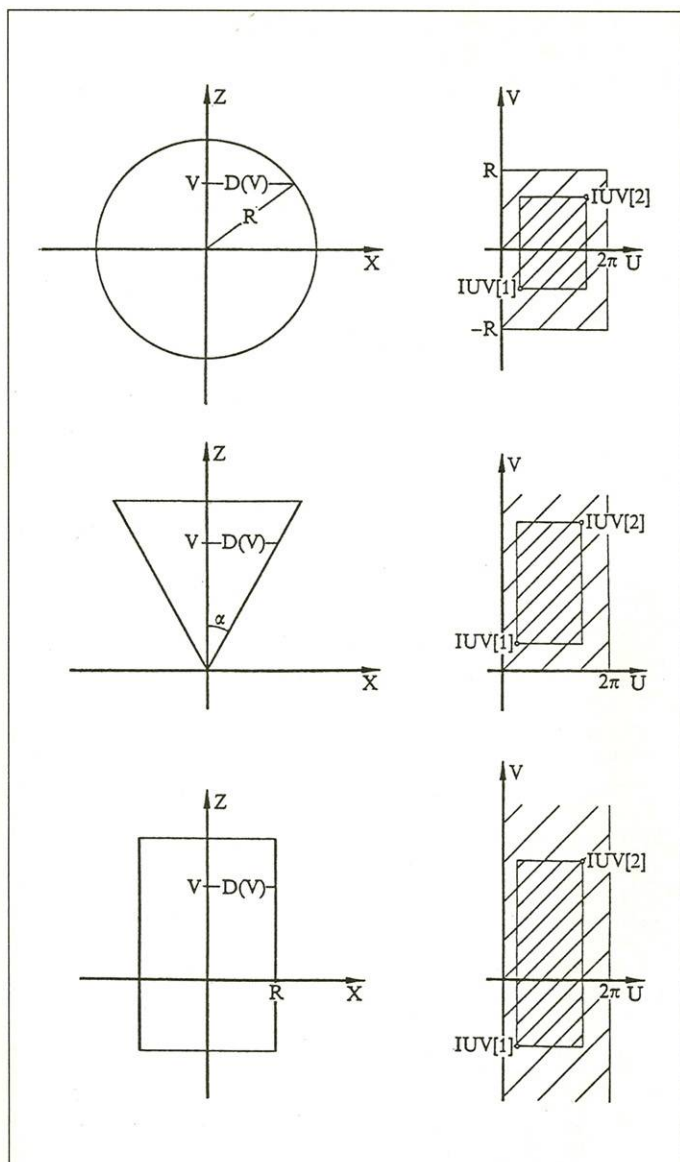


Bild 9. Die Rotationsflächen Kugel, Kegel und Zylinder

● **Konturen:** Das Zeichnen der Konturen entwickeln Sie getrennt für die einzelnen Flächen. Mit »ucontour.pas« (Listing 9) erzeugen Sie drei Objekttypen, die Sie aber anschließend in einem Objekttyp zusammenführen.

Beispiele:

● Mit dem Programm »rs1.pas« (Listing 10) zeichnen Sie Meridiane und Breitenkreise auf den drei Flächen (Bild 10). Folgendes sollten Sie bei der Definition dieser Flächen beachten: Zunächst teilen Sie dem Programm mit, wie viele Flächen Sie definieren möchten. In diesem Beispiel reservieren Sie nacheinander mit der Prozedur »create« drei Plätze auf dem Heap und definieren mit den zugeordneten Zeigern die drei Instanzen des Objekttyps »rst«.

Anschließend rufen Sie in der NRS-Schleife mit der Prozedur »call« die einzelnen Flächen auf und zeichnen mit den Prozeduren »mer.draw«, »circ.draw« und »contour.draw«. Sie bemerken, wie bequem es ist, neben den Zeigern auch noch die Zählvariable »nrs« zur Verfügung zu haben. Diese können Sie vorzüglich dazu benutzen, gewisse Parameter in Abhängigkeit dieses Parameters »nrs« zu definieren.

● Das Programm »rs2.pas« (Listing 11) zeigt ein kompliziertes Beispiel der Definition einer Flächenliste (Bild 11).

● Mit »rs3.pas« (Listing 12) erzeugen Sie eine aufgeschnittene Kugel (Bild 12). Obwohl eigentlich keine Mechanismen zum Aufschneiden definiert sind, erreichen Sie dies



Bild 10. Breitenkreise und Meridiane auf platonischen Körpern

dadurch, daß Sie fünfmal dieselbe Kugel nehmen und für diese lediglich die Parametergebiete ändern. Natürlich können Sie hier für jede der Teilflächen eine andere Farbe wählen oder zum Beispiel mit Meridianen und Breitenkreisen abwechseln.

● »rs4.pas« erzeugt ein kleines Wäldchen aus Kegeln (Bild 13).

In den Programmen »rs1.pas« bis »rs4.pas« (Listings 10 - 13) ist das Speichern der Bilder vorgesehen. Dazu aktivieren

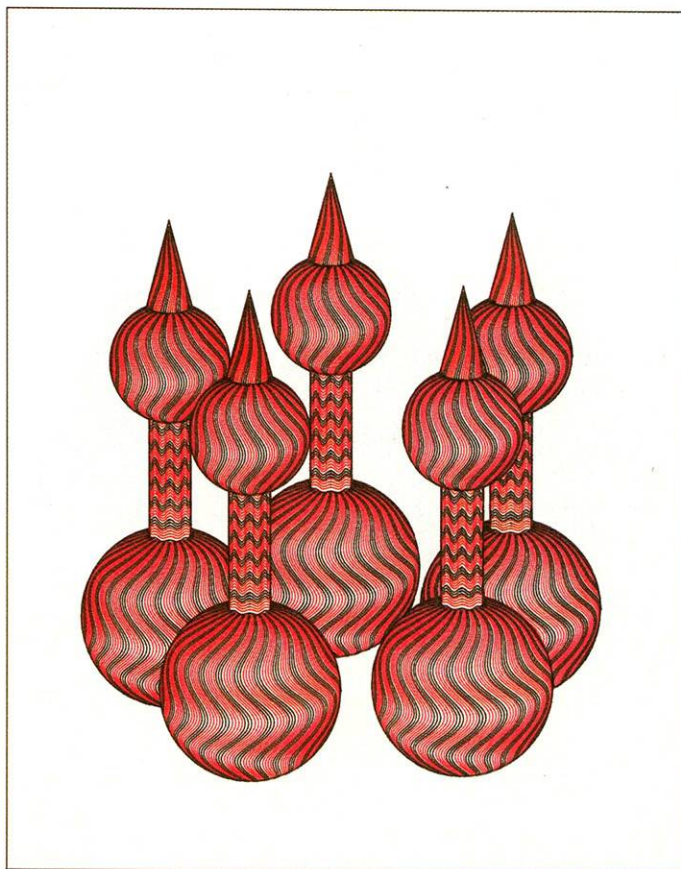


Bild 11. Dieses Objekt erzeugt das Programm »rs2.pas«.

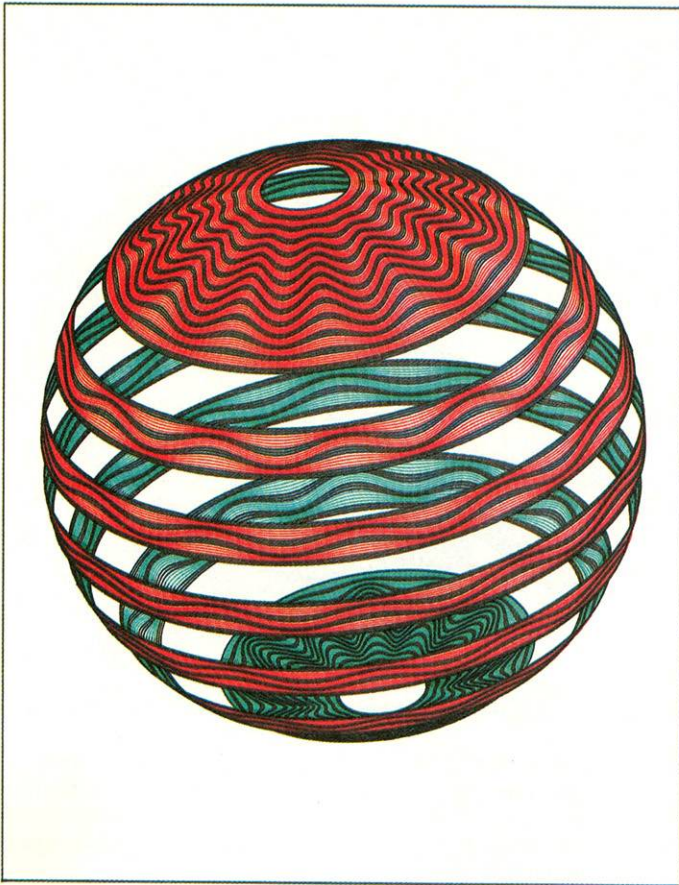


Bild 12. Mit dem Programm »rs3.pas« zeichnen Sie eine aufgeschnittene Kugel, wobei Sie jeder Teilfläche auch eine andere Farbe zuordnen können.

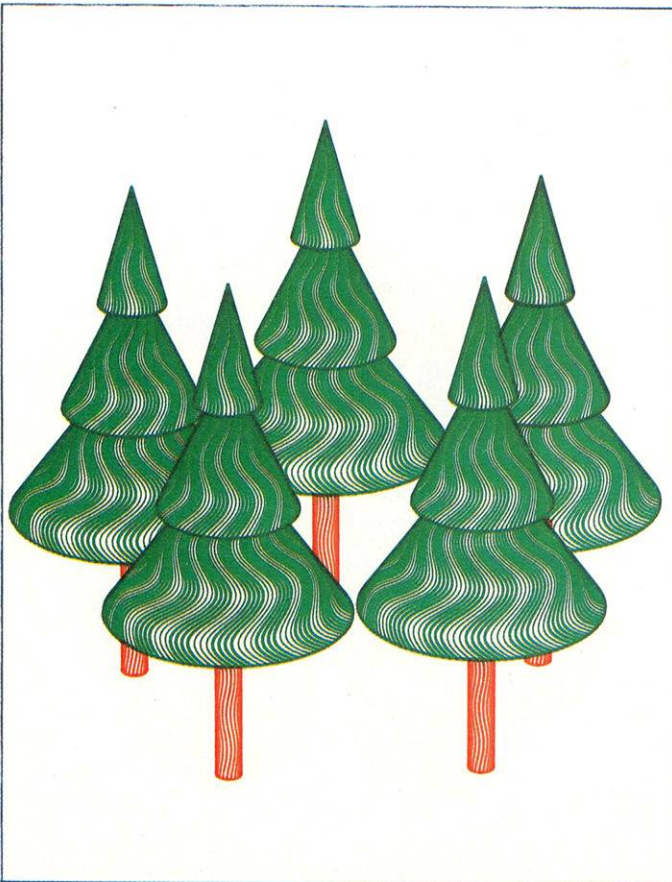


Bild 13. Dieses kleine Wäldchen zeichnen Sie mit »rs4.pas«.

Sie nur die folgenden Zeilen, indem Sie die Kommentarklammern entfernen:

```
{ScrFile:=TRUE;
Assign(ScrF,'RS*.DAT');
Rewrite(ScrF);
Draw;
Close(ScrF);}
```

Die Bilder zeigen Sie dann mit dem Programm »drawsf.pas« (Listing 14) an.

Im übrigen ist die hier vorgestellte Strategie sehr ausbaufähig, ihre objektorientierte Programmierung läßt sich zur Realisierung auch sehr komplexer Geometrie einsetzen.

Die Computergrafiken 1 und 2 zeigen neben den hier vorgestellten Rotationsflächen auch Ebenen und die fünf platonischen Flächen (Tetraeder, Würfel, Oktaeder, Dodekaeder und Ikosaeder). Sie können beliebig viele dieser Flächen in beliebigen Kombinationen bei beliebig kompliziertem Ausschneiden gleichzeitig darstellen und die Flächen mit beliebiger Ornamentik inklusive Schrift versehen.

(Prof. Dr. Kurt Endl/bm)

Literatur:

Kurt Endl, Robert Endl: Computergrafik 1, Würfel-Verlag 1989, 448 Seiten, ISBN 3-923210-95-7, 78 Mark
Kurt Endl: Computergrafik 2, Würfel-Verlag 1991, 528 Seiten, ISBN 3-923210-96-5, 88 Mark

```
[AS3] 1: {Programm: u3dgeom.pas
[Z36] 2: Funktion: Funktionen für 3D-Grafik
[G64] 3: Sprache: Turbo Pascal 6.0
[G13] 4: Autor: Kurt Endl
[585] 5: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 6:
[R52] 7: UNIT U3DGEOM;
[SK1] 8: INTERFACE
[9L4] 9: USES Crt, Graph, U1, U2DGEOM;
[ST0] 10: TYPE
[YR2] 11: Pt3DInt = RECORD
[EK2] 12: X,Y,Z: INTEGER;
[ZN0] 13: END;
[YY1] 14: Pt3D = RECORD
[VM2] 15: X,Y,Z: EXTENDED;
[ZN0] 16: END;
[FA5] 17: Interval3D = ARRAY[1..2] OF Pt3D;
[BS1] 18: Vt3D = Pt3D;
[UB2] 19: PtSph = RECORD
[7R3] 20: R,Phi,Theta: EXTENDED;
[ZN0] 21: END;
[BA2] 22: PtCyl = RECORD
[D13] 23: R,Phi,Z: EXTENDED;
[ZN0] 24: END;
[WS4] 25: CoordinateSystem3D = RECORD
[851] 26: O: Pt3D;
[TH2] 27: UX,UY,UZ: Vt3D;
[ZN0] 28: END;
[GG2] 29: Line3D = RECORD
[851] 30: O: Pt3D;
[U51] 31: U: Vt3D;
[ZN0] 32: END;
[ER4] 33: Plane = CoordinateSystem3D;
[RN0] 34: VAR
[0R2] 35: WI3D: Interval3D;
[7A2] 36: P2DA12: Pt2DA12;
[RY2] 37: PROCEDURE DefPt3D
[525] 38: (X,Y,Z: EXTENDED; VAR P: Pt3D);
[7P2] 39: PROCEDURE DefI3D
[NG4] 40: (X1,Y1,Z1,X2,Y2,Z2: EXTENDED;
[CF3] 41: VAR I3D: Interval3D);
[773] 42: PROCEDURE DefPtSph
[V26] 43: (R,Phi,Theta: EXTENDED; VAR P: PtSph);
[E22] 44: PROCEDURE SphCart
[676] 45: (Dgr:BOOLEAN; PSph:PtSph; VAR P:Pt3D);
[M03] 46: PROCEDURE CartSph
[F44] 47: (P:Pt3D; VAR PSph:PtSph);
[SF3] 48: PROCEDURE CartCylin
[8C4] 49: (P: Pt3D; VAR PCyl: PtCyl);
[544] 50: PROCEDURE DirectionVt3D
[5Z5] 51: (Dgr: BOOLEAN; R,Phi,Theta: EXTENDED;
[RZ1] 52: VAR V: Vt3D);
[HH3] 53: PROCEDURE LengthVt3D
[AB4] 54: (V: Vt3D; VAR L: EXTENDED);
[BD3] 55: PROCEDURE ScaleVt3D
[K96] 56: (V:Vt3D; LG:EXTENDED; VAR ScalV:Vt3D);
[323] 57: PROCEDURE SumVt3D
```



```
[A74] 58: (V1,V2: Vt3D; VAR SV: Vt3D);
[N44] 59: PROCEDURE DifferenceVt3D
[E54] 60: (V1,V2: Vt3D; VAR DV: Vt3D);
[GK5] 61: PROCEDURE LinearCombinationVt3D
[XS4] 62: (S1,S2: EXTENDED; V1,V2: Vt3D;
[V72] 63: VAR LC: Vt3D);
[6Y4] 64: PROCEDURE ScalarProductVt3D
[VX4] 65: (V1,V2: Vt3D; VAR SP: EXTENDED);
[1C4] 66: PROCEDURE VectorProduct
[X74] 67: (V1,V2: Vt3D; VAR V: Vt3D);
[6E3] 68: PROCEDURE TToLine3D
[B76] 69: (Ln3D: Line3D; T: EXTENDED; VAR P: Pt3D);
[LH5] 70: PROCEDURE DefineNewCS3DWithPt3D
[YD5] 71: (ONew: Pt3D; UZNew: Vt3D; C: Pt3D;
[B25] 72: VAR CS3D: CoordinateSystem3D);
[Y84] 73: PROCEDURE World3DToNew3D
[375] 74: (CS3D: CoordinateSystem3D; P: Pt3D;
[5K2] 75: VAR PNew: Pt3D);
[864] 76: PROCEDURE New3DToWorld3D
[AC6] 77: (CS3D: CoordinateSystem3D; PNew: Pt3D;
[5Z1] 78: VAR P: Pt3D);
[E55] 79: PROCEDURE DefinePlaneWithVt3D
[WU4] 80: (O: Pt3D; UZ: Vt3D; YUP: Vt3D;
[AH2] 81: VAR P1: Plane);
[2D3] 82: PROCEDURE PlaneTo3D
[A75] 83: (P1: Plane; Q: Pt2D; VAR P: Pt3D);
[3R5] 84: PROCEDURE IntersectionLinePlane
[HP3] 85: (Ln3D: Line3D; P1: Plane;
[AG3] 86: VAR Intsec: BOOLEAN;
[ØT4] 87: VAR T: EXTENDED; VAR Q: Pt2D);
[ØØØ] 88:
[4L2] 89: IMPLEMENTATION
[ØØØ] 90:
[7P2] 91: PROCEDURE DefI3D
[NG4] 92: (X1,Y1,Z1,X2,Y2,Z2: EXTENDED;
[CF3] 93: VAR I3D: Interval3D);
[MUØ] 94: BEGIN
[1C4] 95: I3D[1].X:=X1; I3D[2].X:=X2;
[BC4] 96: I3D[1].Y:=Y1; I3D[2].Y:=Y2;
[NC4] 97: I3D[1].Z:=Z1; I3D[2].Z:=Z2;
[ZNØ] 98: END;
[VM1] 99: PROCEDURE
[K26] 100: DefPt3D(X,Y,Z: EXTENDED; VAR P: Pt3D);
[MUØ] 101: BEGIN
[BZ3] 102: P.X:=X; P.Y:=Y; P.Z:=Z;
[ZNØ] 103: END;
[773] 104: PROCEDURE DefPtSph
[V26] 105: (R,Phi,Theta: EXTENDED; VAR P: PtSph);
[MY5] 106: {Definiert einen 3D-Punkt in Kugel-
[T82] 107: koordinaten}
[MUØ] 108: BEGIN
[881] 109: P.R:=R;
[DW1] 110: P.Phi:=Phi;
[SN2] 111: P.Theta:=Theta;
[ZNØ] 112: END;
[E22] 113: PROCEDURE SphCart
[NE6] 114: (Dgr: BOOLEAN; PSph: PtSph; VAR P: Pt3D);
[BT6] 115: {Berechnet zu einem in Kugelkoordinaten
[XX5] 116: gegebenen Punkt dessen kartesische
[T82] 117: Koordinaten}
[MUØ] 118: BEGIN
[BM2] 119: IF Dgr THEN BEGIN
[354] 120: PSph.Phi:=PSph.Phi*PI/180;
[VY4] 121: PSph.Theta:=PSph.Theta*PI/180;
[ZNØ] 122: END;
[5B4] 123: P.X:=PSph.R*COS(PSph.Phi)
[3V2] 124: *COS(PSph.Theta);
[3B4] 125: P.Y:=PSph.R*SIN(PSph.Phi)
[3V2] 126: *COS(PSph.Theta);
[7V4] 127: P.Z:=PSph.R*SIN(PSph.Theta);
[ZNØ] 128: END;
[MØ3] 129: PROCEDURE CartSph
[9D4] 130: (P: Pt3D; VAR PSph: PtSph);
[436] 131: {Berechnet zu einem in kartesischen
[K16] 132: Koordinaten gegebenen Punkt dessen
[P53] 133: Kugelkoordinaten}
[2A2] 134: VAR P2D: Pt2D;
[S32] 135: PPol: PtPol;
[MUØ] 136: BEGIN
[RK6] 137: PSph.R:=SQRT(SQR(P.X)+SQR(P.Y)+SQR(P.Z));
[6V1] 138: P2D.X:=P.X;
[BV1] 139: P2D.Y:=P.Y;
[9D3] 140: CartPolar(P2D, PPol);
[KB3] 141: PSph.Phi:=PPol.Phi;
[M65] 142: P2D.X:=SQRT(SQR(P.X)+SQR(P.Y));
[EV1] 143: P2D.Y:=P.Z;
[9D3] 144: CartPolar(P2D, PPol);
[HN3] 145: PSph.Theta:=PPol.Phi;
[ZNØ] 146: END;
[SF3] 147: PROCEDURE CartCylin
[8C4] 148: (P: Pt3D; VAR PCyl: PtCyl);
[436] 149: {Berechnet zu einem in kartesischen
[K16] 150: Koordinaten gegebenen Punkt dessen
[ZS3] 151: Zylinderkoordinaten}
[2A2] 152: VAR P2D: Pt2D;
[S32] 153: PPol: PtPol;
[MUØ] 154: BEGIN
[N32] 155: PCyl.Z:=P.Z;
```

```
[HR3] 156: P2D.X:=P.X; P2D.Y:=P.Y;
[9D3] 157: CartPolar(P2D, PPol);
[6R2] 158: PCyl.R:=PPol.R;
[AB3] 159: PCyl.Phi:=PPol.Phi;
[ZNØ] 160: END;
[544] 161: PROCEDURE DirectionVt3D
[5Z5] 162: (Dgr: BOOLEAN; R,Phi,Theta: EXTENDED;
[RZ1] 163: VAR V: Vt3D);
[WK6] 164: {Erzeugt zu R,Phi,Theta den Vektor}
[MUØ] 165: BEGIN
[BM2] 166: IF Dgr THEN BEGIN
[YU5] 167: Phi:=Phi*PI/180; Theta:=Theta*PI/180;
[ZNØ] 168: END;
[3D4] 169: V.X:=R*COS(Phi)*COS(Theta);
[W4] 170: V.Y:=R*SIN(Phi)*COS(Theta);
[6X2] 171: V.Z:=R*SIN(Theta);
[ZNØ] 172: END;
[HH3] 173: PROCEDURE LengthVt3D
[AB4] 174: (V: Vt3D; VAR L: EXTENDED);
[T36] 175: {Berechnet die Länge L des Vektors V}
[MUØ] 176: BEGIN
[DY5] 177: L:=SQRT(SQR(V.X)+SQR(V.Y)+SQR(V.Z));
[ZNØ] 178: END;
[BD3] 179: PROCEDURE ScaleVt3D
[AD6] 180: (V: Vt3D; LG: EXTENDED; VAR ScalV: Vt3D);
[8Ø7] 181: {Führt den Vektor V in einen Vektor der
[SW2] 182: Länge LG über}
[KM2] 183: VAR L: EXTENDED;
[MUØ] 184: BEGIN
[TE1] 185: ScalV:=V;
[1P2] 186: LengthVt3D(V, L);
[5P4] 187: IF NOT Null(L,Eps4) THEN BEGIN
[GW2] 188: ScalV.X:=V.X/L*LG;
[NW2] 189: ScalV.Y:=V.Y/L*LG;
[UW2] 190: ScalV.Z:=V.Z/L*LG;
[ZNØ] 191: END;
[ZNØ] 192: END;
[323] 193: PROCEDURE SumVt3D
[A74] 194: (V1,V2: Vt3D; VAR SV: Vt3D);
[MUØ] 195: BEGIN
[VK2] 196: SV.X:=V1.X+V2.X;
[3L2] 197: SV.Y:=V1.Y+V2.Y;
[BL2] 198: SV.Z:=V1.Z+V2.Z;
[ZNØ] 199: END;
[N44] 200: PROCEDURE DifferenceVt3D
[E54] 201: (V1,V2: Vt3D; VAR DV: Vt3D);
[MUØ] 202: BEGIN
[5K2] 203: DV.X:=V1.X-V2.X;
[DK2] 204: DV.Y:=V1.Y-V2.Y;
[NK2] 205: DV.Z:=V1.Z-V2.Z;
[ZNØ] 206: END;
[GK5] 207: PROCEDURE LinearCombinationVt3D
[XS4] 208: (S1,S2: EXTENDED; V1,V2: Vt3D;
[V72] 209: VAR LC: Vt3D);
[MUØ] 210: BEGIN
[XD3] 211: LC.X:=S1*V1.X+S2*V2.X;
[2E3] 212: LC.Y:=S1*V1.Y+S2*V2.Y;
[7E3] 213: LC.Z:=S1*V1.Z+S2*V2.Z;
[ZNØ] 214: END;
[6Y4] 215: PROCEDURE ScalarProductVt3D
[VX4] 216: (V1,V2: Vt3D; VAR SP: EXTENDED);
[MUØ] 217: BEGIN
[M75] 218: SP:=V1.X*V2.X+V1.Y*V2.Y+V1.Z*V2.Z;
[ZNØ] 219: END;
[1C4] 220: PROCEDURE VectorProduct
[X74] 221: (V1,V2: Vt3D; VAR V: Vt3D);
[PY2] 222: VAR LV: EXTENDED;
[MUØ] 223: BEGIN
[VØ4] 224: V.X:= V1.Y*V2.Z-V1.Z*V2.Y;
[274] 225: V.Y:= -(V1.X*V2.Z-V1.Z*V2.X);
[TØ4] 226: V.Z:= V1.X*V2.Y-V1.Y*V2.X;
[VV2] 227: LengthVt3D(V, LV);
[C15] 228: IF NOT Null(LV,Eps4) THEN BEGIN
[H66] 229: V.X:=V.X/LV; V.Y:=V.Y/LV; V.Z:=V.Z/LV;
[ZNØ] 230: END;
[4E4] 231: IF Null(LV,Eps4) THEN BEGIN
[4R5] 232: OutTextXY(100,100, 'VecProd failed');
[NN1] 233: Delay(1000);
[DUØ] 234: Halt;
[ZNØ] 235: END;
[ZNØ] 236: END;
[6E3] 237: PROCEDURE TToLine3D
[B76] 238: (Ln3D: Line3D; T: EXTENDED; VAR P: Pt3D);
[Ø47] 239: {Erzeugt auf der Geraden Line3D den Punkt,
[E65] 240: der zum Parameter T gehört}
[MUØ] 241: BEGIN
[FT3] 242: LinearCombinationVt3D
[CE3] 243: (1,T,Ln3D.O,Ln3D.U, P);
[ZNØ] 244: END;
[LH5] 245: PROCEDURE DefineNewCS3DWithPt3D
[YD5] 246: (ONew: Pt3D; UZNew: Vt3D; C: Pt3D;
[B25] 247: VAR CS3D: CoordinateSystem3D);
[2X6] 248: {Führt ein neues Koordinatensystem mit
[1P6] 249: Nullpunkt ONew, Z-Achse UZNew ein, wobei
[HY6] 250: die X-Achse nach dem zusätzlichen Punkt
```

Listing 1. Die wichtigsten Elemente zur Darstellung von 3D-Grafik


```
[ME3] 251: C ausgerichtet ist}
[OL2] 252: VAR ONewC: Vt3D;
[MU0] 253: BEGIN
[EM4] 254:   CS3D.O:=ONew; CS3D.UZ:=UZNew;
[9W4] 255:   DifferenceVt3D(C,ONew, ONewC);
[M46] 256:   VectorProduct(UZNew,ONewC, CS3D.UY);
[OF6] 257:   VectorProduct(CS3D.UY,UZNew, CS3D.UX);
[ZN0] 258: END;
[TH5] 259: PROCEDURE DefineNewCS3DWithVt3D
[NR5] 260:   (ONew: Pt3D; UZNew: Vt3D; YUP: Vt3D;
[B25] 261:   VAR CS3D: CoordinateSystem3D);
[2X6] 262: {Führt ein neues Koordinatensystem mit
[1P6] 263: Nullpunkt ONew, Z-Achse UZNew ein, wobei
[E17] 264: die Y-Achse nach dem zusätzlichen Vektor
[NX3] 265: YUP ausgerichtet ist}
[MU0] 266: BEGIN
[EM4] 267:   CS3D.O:=ONew; CS3D.UZ:=UZNew;
[9Z5] 268:   VectorProduct(YUP,UZNew, CS3D.UX);
[3M6] 269:   VectorProduct(UZNew,CS3D.UX, CS3D.UY);
[ZN0] 270: END;
[Y84] 271: PROCEDURE World3DToNew3D
[3T5] 272:   (CS3D: CoordinateSystem3D; P: Pt3D;
[5K2] 273:   VAR PNew: Pt3D);
[DR6] 274: {Berechnet zu dem im 3D-Weltkoordinaten-
[GN6] 275: system gegebenen Punkt P dessen Koordi-
[UR4] 276: naten im neuen System CS3D}
[UL2] 277: VAR ONewP: Vt3D;
[MU0] 278: BEGIN
[785] 279:   DifferenceVt3D(P,CS3D.O, ONewP);
[P67] 280:   ScalarProductVt3D(ONewP,CS3D.UX, PNew.X);
[Y67] 281:   ScalarProductVt3D(ONewP,CS3D.UY, PNew.Y);
[677] 282:   ScalarProductVt3D(ONewP,CS3D.UZ, PNew.Z);
[ZN0] 283: END;
[864] 284: PROCEDURE New3DToWorld3D
[AC6] 285:   (CS3D: CoordinateSystem3D; PNew: Pt3D;
[5Z1] 286:   VAR P: Pt3D);
[NU6] 287: {Berechnet zu dem im neuen 3D-Koordinaten-
[VA6] 288: system gegebenen Punkt PNew dessen 3D-
[U03] 289: Weltkoordinaten}
[9W2] 290: VAR LC1,LC2: Vt3D;
[MU0] 291: BEGIN
[FT3] 292:   LinearCombinationVt3D
[PR4] 293:   (1,PNew.X,CS3D.O,CS3D.UX, LC1);
[7X3] 294:   LinearCombinationVt3D(
[274] 295:   1,PNew.Y,LC1,CS3D.UY, LC2);
[FT3] 296:   LinearCombinationVt3D
[MS3] 297:   (1,PNew.Z,LC2,CS3D.UZ, P);
[ZN0] 298: END;
[E55] 299: PROCEDURE DefinePlaneWithVt3D
[WU4] 300:   (O: Pt3D; UZ: Vt3D; YUP: Vt3D;
[AH2] 301:   VAR Pl: Plane);
[4F6] 302: {Definiert eine Ebene mit Nullpunkt O,
[YX5] 303: Normalenvektor UZ, wobei die Y-Achse
[LL6] 304: nach dem Vektor YUP ausgerichtet ist}
[MU0] 305: BEGIN
[656] 306:   DefineNewCS3DWithVt3D(O,UZ,YUP, Pl);
[ZN0] 307: END;
[2D3] 308: PROCEDURE PlaneTo3D
[A75] 309:   (Pl: Plane; Q: Pt2D; VAR P: Pt3D);
[HC6] 310: {Rechnet Ebenenkoordinaten in 3D-Welt-
[123] 311: koordinaten um}
[V42] 312: VAR OP: Vt3D;
[MU0] 313: BEGIN
[FT3] 314:   LinearCombinationVt3D
[X04] 315:   (Q.X,Q.Y,Pl.UX,Pl.UY, OP);
[L73] 316:   SumVt3D(Pl.O,OP, P);
[ZN0] 317: END;
[ZE3] 318: PROCEDURE D3ToPlane
[A75] 319:   (Pl: Plane; P: Pt3D; VAR Q: Pt2D);
[D76] 320: {Rechnet Weltkoordinaten in Ebenen-
[123] 321: koordinaten um}
[V42] 322: VAR OP: Vt3D;
[MU0] 323: BEGIN
[4B4] 324:   DifferenceVt3D(P,Pl.O, OP);
[PP5] 325:   ScalarProductVt3D(OP,Pl.UX, Q.X);
[XP5] 326:   ScalarProductVt3D(OP,Pl.UY, Q.Y);
[ZN0] 327: END;
[3R5] 328: PROCEDURE IntersectionLinePlane
[HP3] 329:   (Ln3D: Line3D; Pl: Plane;
[AG3] 330:   VAR Intsec: BOOLEAN;
[ØT4] 331:   VAR T: EXTENDED; VAR Q: Pt2D);
[DX6] 332: {Berechnet den Schnittpunkt der Geraden
[BNS] 333: Ln3D mit der Ebene Plane, wobei der
[FS6] 334: Parameter T der Geraden und der Ebenen-
[AM4] 335: punkt Q ausgegeben werden}
[DA3] 336: VAR SZ,SN: EXTENDED;
[G51] 337:   S: Pt3D;
[LR1] 338:   OOL: Vt3D;
[MU0] 339: BEGIN
[4D2] 340:   Intsec:=FALSE;
[PA5] 341:   DifferenceVt3D(Ln3D.O,Pl.O, OOL);
[TU5] 342:   ScalarProductVt3D(OOL,Pl.UZ, SZ);
[H26] 343:   ScalarProductVt3D(Ln3D.U,Pl.UZ, SN);
[Y05] 344:   IF NOT Null(SN,Eps3) THEN BEGIN
[4A2] 345:     Intsec:=TRUE;
[7M1] 346:     T:=-SZ/SN;
[FT3] 347:     LinearCombinationVt3D
[KE3] 348:     (1,T,Ln3D.O,Ln3D.U, S);
```

```
[E23] 349:   D3ToPlane(Pl,S, Q);
[ZN0] 350: END;
[ZN0] 351: END;
[KN0] 352: END.
```

Listing 1. Die wichtigsten Elemente zur Darstellung von 3D-Grafik

```
[413] 1: {Programm: u3d.pas
[LP6] 2: Funktion: Funktionen für Perspektive
[CL2] 3:   und Projektion
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[ØØØ] 7:
[HE1] 8: UNIT U3D;
[SK1] 9: INTERFACE
[KØ6] 10: USES Crt, Graph, U1, UDRAW2D, U2DGEOM,
[ØC1] 11:   U3DGEOM;
[STØ] 12: TYPE
[1L6] 13: Interval3DInt = ARRAY[1..2] OF Pt3DInt;
[9U5] 14: Pt3DIntA8 = ARRAY[1..8] OF Pt3DInt;
[EV3] 15: ProjectionSph = RECORD
[ØT2] 16:   C,O,Z,YUP: PtSph;
[ZNØ] 17:   END;
[YB3] 18: Projection = RECORD
[G41] 19:   C: Pt3D;
[XM1] 20:   Pl: Plane;
[ZNØ] 21:   END;
[6XØ] 22: CONST
[E83] 23:   Corner: Pt3DIntA8 =
[KG4] 24:   ((X:1; Y:1; Z:1), (X:2; Y:1; Z:1),
[7D4] 25:   (X:2; Y:2; Z:1), (X:1; Y:2; Z:1),
[AD4] 26:   (X:1; Y:1; Z:2), (X:2; Y:1; Z:2),
[TK4] 27:   (X:2; Y:2; Z:2), (X:1; Y:2; Z:2));
[TS2] 28:   Edge: Pt2DIntA12 =
[YK4] 29:   ((X:1;Y:2), (X:4;Y:3), (X:1;Y:4),
[CK4] 30:   (X:2;Y:3), (X:5;Y:6), (X:8;Y:7),
[WE4] 31:   (X:5;Y:8), (X:6;Y:7), (X:1;Y:5),
[XD4] 32:   (X:2;Y:6), (X:3;Y:7), (X:4;Y:8));
[4A2] 33: VAR O3D: Pt3D;
[ØR2] 34:   WI3D: Interval3D;
[6P3] 35:   PrSph: ProjectionSph;
[US2] 36:   Pr: Projection;
[7A2] 37:   P2DA12: Pt2DA12;
[8Ø3] 38: PROCEDURE Project
[D94] 39:   (P: Pt3D; VAR Ln3D: Line3D;
[RØ6] 40:   VAR DistCP: EXTENDED; VAR P2D: Pt2D);
[7W3] 41: PROCEDURE Parameter3D;
[N33] 42: PROCEDURE DrawCS3D
[1P5] 43:   (KKXtiles,KKYtiles,Col: INTEGER);
[AV2] 44: FUNCTION Damping
[GK6] 45:   (Ild: Interval1D; T: EXTENDED): EXTENDED;
[HD4] 46: PROCEDURE SquareEquation
[BMS] 47:   (A,B,C: EXTENDED; VAR NOS: INTEGER;
[183] 48:   VAR T1,T2: EXTENDED);
[ØØØ] 49:
[4L2] 50: IMPLEMENTATION
[ØØØ] 51:
[GZ4] 52: PROCEDURE DefinePerspective;
[K96] 53: {Berechnet zu den in Kugelkoordinaten
[N86] 54: gegebenen Parametern der Perspektive
[ZA6] 55: deren kartesische 3D-Weltkoordinaten}
[EF2] 56: VAR YUP: Pt3D;
[MUØ] 57: BEGIN
[4P4] 58:   SphCart(TRUE,PrSph.C, Pr.C);
[VB5] 59:   SphCart(TRUE,PrSph.O, Pr.Pl.O);
[SG5] 60:   SphCart(TRUE,PrSph.Z, Pr.Pl.UZ);
[645] 61:   SphCart(TRUE,PrSph.YUP, YUP);
[VE3] 62:   DefinePlaneWithVt3D
[KT4] 63:   (Pr.Pl.O,Pr.Pl.UZ,YUP, Pr.Pl);
[ZNØ] 64: END;
[S94] 65: PROCEDURE ProjectionRay
[6B4] 66:   (C,P: Pt3D; VAR Ln3D: Line3D;
[2U3] 67:   VAR DistCP: EXTENDED);
[4R5] 68: {Berechnet die Gerade Ln3D durch die
[3Z5] 69: Punkte C und P und deren Entfernung}
[342] 70: VAR CP: Vt3D;
[MUØ] 71: BEGIN
[RH1] 72:   Ln3D.O:=C;
[CS3] 73:   DifferenceVt3D(P,C, CP);
[MT3] 74:   LengthVt3D(CP, DistCP);
[HN3] 75:   ScaleVt3D(CP,1, Ln3D.U);
[ZNØ] 76: END;
[8Ø3] 77: PROCEDURE Project
[D94] 78:   (P: Pt3D; VAR Ln3D: Line3D;
[RØ6] 79:   VAR DistCP: EXTENDED; VAR P2D: Pt2D);
[ET6] 80: {Projiziert den 3D-Punkt P in den Punkt
[HY6] 81: P2D im 2D-Weltkoordinatensystem und gibt
[486] 82: hierbei den Sehstrahl Ln3D und die Ent-
[HP6] 83: fernung des Punktes P vom Projektions-
[2L2] 84: zentrum C aus}
[AF4] 85: VAR DistToPrPl: EXTENDED;
[ØE1] 86:   DV: Vt3D;
```



```
[LT2] 87: Intsec: BOOLEAN;
[MU0] 88: BEGIN
[VV5] 89: ProjectionRay(Pr.C,P, Ln3D,DistCP);
[ZV3] 90: IntersectionLinePlane
[PZ5] 91: (Ln3D,Pr.Pl, Intsec,DistToPrPl,P2D);
[ZN0] 92: END;
[GA6] 93: PROCEDURE WI3DTOWI2D(VAR P2DA: Pt2DA12);
[977] 94: {Berechnet die Projektionspunkte der Ecken
[H66] 95: des 3D-Weltintervalls und das 2D-Welt-
[P02] 96: intervall}
[9K2] 97: VAR N: INTEGER;
[SL2] 98: Dist: EXTENDED;
[9H3] 99: P: ARRAY[1..8] OF Pt3D;
[7A2] 100: PrRay: Line3D;
[951] 101: Q: Pt2D;
[MU0] 102: BEGIN
[MF3] 103: FOR N:=1 TO 8 DO BEGIN
[515] 104: P[N].X:=WI3D[Corner[N].X].X;
[D15] 105: P[N].Y:=WI3D[Corner[N].Y].Y;
[N15] 106: P[N].Z:=WI3D[Corner[N].Z].Z;
[ZN0] 107: END;
[5T4] 108: Project(P[1],PrRay, Dist,Q);
[RY1] 109: P2DA[1]:=Q;
[B85] 110: WI2D[1].X:=Q.X; WI2D[1].Y:=Q.Y;
[H85] 111: WI2D[2].X:=Q.X; WI2D[2].Y:=Q.Y;
[NF3] 112: FOR N:=2 TO 8 DO BEGIN
[VW4] 113: Project(P[N], PrRay,Dist,Q);
[E12] 114: P2DA[N]:=Q;
[WY5] 115: IF Q.X<WI2D[1].X THEN WI2D[1].X:=Q.X;
[725] 116: IF Q.X>WI2D[2].X THEN WI2D[2].X:=Q.X;
[525] 117: IF Q.Y<WI2D[1].Y THEN WI2D[1].Y:=Q.Y;
[GZ5] 118: IF Q.Y>WI2D[2].Y THEN WI2D[2].Y:=Q.Y;
[ZN0] 119: END;
[ZN0] 120: END;
[7W3] 121: PROCEDURE Parameter3D;
[547] 122: {Berechnet die Transformationsparameter}
[MU0] 123: BEGIN
[747] 124: TransformationIScrScalToIScr(IScrScal);
[183] 125: DefinePerspective;
[313] 126: WI3DTOWI2D(P2DA12);
[435] 127: ParameterTransformationScr;
[ZN0] 128: END;
[3Y2] 129: PROCEDURE DrawI3D
[Z52] 130: (P2DA: Pt2DA12;
[4D5] 131: KKXTiles,KKYTiles,Col: INTEGER);
[VT5] 132: {Zeichnet das 3D-Intervall, dessen
[PK5] 133: Projektionspunkte durch den Array
[L23] 134: P2DA gegeben sind}
[PC5] 135: VAR N,KXTiles,KYTiles: INTEGER;
[BY1] 136: T: EXTENDED;
[YM1] 137: P1,P2: Pt2D;
[T51] 138: V: Vt2D;
[MU0] 139: BEGIN
[RE2] 140: Line2D.Init(0);
[GM3] 141: FOR N:=1 TO 12 DO BEGIN
[YW4] 142: Line2D.Draw(P2DA[Edge[N].X],
[UY3] 143: P2DA[Edge[N].Y],1,Col);
[ZN0] 144: END;
[7X5] 145: FOR KXTiles:=0 TO KKXTiles DO BEGIN
[OL3] 146: T:=KXTiles/KKYTiles;
[TN5] 147: DifferenceVt2D(P2DA[2],P2DA[1], V);
[ET3] 148: LinearCombinationVt2D
[BT2] 149: (1,T,P2DA[1],V, P1);
[1P5] 150: DifferenceVt2D(P2DA[3],P2DA[4], V);
[ET3] 151: LinearCombinationVt2D
[UT2] 152: (1,T,P2DA[4],V, P2);
[MU3] 153: Line2D.Draw(P1,P2,1,Col);
[ZN0] 154: END;
[CK5] 155: FOR KYTiles:=0 TO KKYTiles DO BEGIN
[5L3] 156: T:=KYTiles/KKXTiles;
[XN5] 157: DifferenceVt2D(P2DA[4],P2DA[1], V);
[ET3] 158: LinearCombinationVt2D
[BT2] 159: (1,T,P2DA[1],V, P1);
[XN5] 160: DifferenceVt2D(P2DA[3],P2DA[2], V);
[ET3] 161: LinearCombinationVt2D
[KT2] 162: (1,T,P2DA[2],V, P2);
[MU3] 163: Line2D.Draw(P1,P2,1,Col);
[ZN0] 164: END;
[ZN0] 165: END;
[N33] 166: PROCEDURE DrawCS3D
[1P5] 167: (KKXTiles,KKYTiles,Col: INTEGER);
[107] 168: {Zeichnet das 3D-Weltintervall, zusammen
[ZH5] 169: mit den Bildschirmintervallen}
[MU0] 170: BEGIN
[Y73] 171: DrawIScr(IScr,Col);
[MH3] 172: DrawIScr(IScr2D,Col);
[4C6] 173: DrawI3D(P2DA12,KKXTiles,KKYTiles,Col);
[ZN0] 174: END;
[AV2] 175: FUNCTION Damping
[GK6] 176: (IID: IntervallD; T: EXTENDED): EXTENDED;
[UX5] 177: {Dämpfungsfunktion, welche an den
[SS4] 178: Intervallen von IID 0 ist}
[W13] 179: VAR M,R: EXTENDED;
[MU0] 180: BEGIN
[Z45] 181: M:=SQRT(SQRT(IID[2].X-IID[1].X)/4);
[AK4] 182: Damping:=1/M*SQRT((T-IID[1].X)
[4B2] 183: *(IID[2].X-T));
[ZN0] 184: END;
[HD4] 185: PROCEDURE SquareEquation
[BM5] 186: (A,B,C: EXTENDED; VAR NOS: INTEGER;
```

```
[183] 187: VAR T1,T2: EXTENDED);
[B73] 188: VAR D,TC: EXTENDED;
[MU0] 189: BEGIN
[H83] 190: CASE Null(A,Eps3) OF
[T21] 191: TRUE: BEGIN
[D84] 192: IF Null(B,Eps3) THEN NOS:=0
[443] 193: ELSE BEGIN NOS:=1;
[P73] 194: T1:=-C/B; T2:=T1; END;
[ZN0] 195: END;
[XW1] 196: FALSE: BEGIN
[6R1] 197: D:=B*B-4*A*C;
[A24] 198: IF Null(D,Eps3) THEN BEGIN
[R61] 199: NOS:=1;
[5T2] 200: T1:=-B/2/A; T2:=T1;
[4M0] 201: END
[WP1] 202: ELSE BEGIN
[LG2] 203: IF D>0 THEN BEGIN
[B81] 204: NOS:=2;
[FZ2] 205: T1:=(-B+SQRT(D))/2/A;
[MZ2] 206: T2:=(-B-SQRT(D))/2/A;
[133] 207: IF T1>T2 THEN BEGIN
[7K3] 208: TC:=T1; T1:=T2; T2:=TC;
[ZN0] 209: END;
[4M0] 210: END
[YY1] 211: ELSE NOS:=0;
[ZN0] 212: END;
[ZN0] 213: END;
[ZN0] 214: END;
[ZN0] 215: END;
[MU0] 216: BEGIN
[LB4] 217: O3D.X:=0; O3D.Y:=0; O3D.Z:=0;
[KN0] 218: END.
```

Listing 2. Die Unterprogramme für Perspektive und Projektion

```
[9C3] 1: {Programm: cs3d.pas
[HC6] 2: Funktion: perspektivische Zeichnung
[PM3] 3: des 3D-Weltintervalls
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[T62] 8: PROGRAM CS3D;
[3S6] 9: USES Crt, Graph, U1, U2DGEOM, U3DGEOM, U3D;
[585] 10: {-----}
[MU0] 11: BEGIN
[A86] 12: NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 13: InitializeGraphic;
[BS3] 14: DefI2D(1,9,9,1, IScrScal);
[235] 15: DefI3D(-100,-100,-100,100,100,100, WI3D);
[7F4] 16: DefPtSph(350,270,10, PrSph.C);
[DY3] 17: {Projektionszentrum}
[NM4] 18: DefPtSph(-600,270,10, PrSph.O);
[3U5] 19: {Nullpunkt der Projektionsebene}
[E54] 20: DefPtSph(1,270,10, PrSph.Z);
[WR6] 21: {Normalenvektor der Projektionsebene}
[7B4] 22: DefPtSph(1,0,90, PrSph.YUP);
[0D2] 23: {YUP Vektor}
[K32] 24: Parameter3D;
[F82] 25: DrawCS3D(5,8,3);
[P92] 26: CloseGraphic;
[KN0] 27: END.
```

Listing 3. Die perspektivische Zeichnung des 3D-Weltintervalls

```
[254] 1: {Programm: ucurve3d.pas
[DA7] 2: Funktion: Funktionen für Projektion und
[713] 3: Sichtbarkeitstest
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[KE2] 8: UNIT UCURVE3D;
[SK1] 9: INTERFACE
[UT5] 10: USES Crt, Graph, U1, UDRAW, U3DGEOM;
[ST0] 11: TYPE
[HP2] 12: Curve3DT = OBJECT
[DH2] 13: PROCEDURE TToP
[9A5] 14: (T: EXTENDED; VAR PrRay: Line3D;
[TC3] 15: VAR DistN: EXTENDED;
[1L3] 16: VAR P: Pt2D; VIRTUAL;
[EP2] 17: PROCEDURE Check
[ZA5] 18: (PrRay: Line3D; DistN: EXTENDED;
[XH4] 19: VAR Chk: BOOLEAN); VIRTUAL;
[UN3] 20: PROCEDURE DrawCurve3D
[0H5] 21: (IID: IntervallD; LL,Col: INTEGER);
```

Listing 4. »ucurve3d.pas« enthält Funktionen für Projektion und Sichtbarkeitstest.


```
[ZN0] 22: END;
[000] 23:
[4L2] 24: IMPLEMENTATION
[000] 25:
[E54] 26: PROCEDURE Curve3DT.TToP
[9A5] 27: (T: EXTENDED; VAR PrRay: Line3D;
[MG5] 28: VAR DistN: EXTENDED; VAR P: Pt3D);
[FK1] 29: BEGIN END;
[Y34] 30: PROCEDURE Curve3DT.Check
[ZA5] 31: (PrRay: Line3D; DistN: EXTENDED;
[8Z2] 32: VAR Chk: BOOLEAN);
[FK1] 33: BEGIN END;
[Y95] 34: PROCEDURE Curve3DT.DrawCurve3D
[0H5] 35: (I1D: IntervalD; LL,Col: INTEGER);
[0Y2] 36: VAR L,CN: INTEGER;
[275] 37: Step,T,T1,T2,TI,Dist: EXTENDED;
[314] 38: ChkOld,Chk,ChkI: BOOLEAN;
[HT1] 39: P,PI: Pt3D;
[7A2] 40: PrRay: Line3D;
[MU0] 41: BEGIN
[AY4] 42: Step:=1/LL*(I1D[2].X-I1D[1].X);
[BX2] 43: T:=I1D[1].X-Step;
[FB2] 44: ChkOld:=FALSE;
[FS3] 45: FOR L:=0 TO LL DO BEGIN
[LP1] 46: T:=T+Step;
[ZM3] 47: TToP(T, PrRay,Dist,P);
[SU3] 48: Check(PrRay,Dist, Chk);
[CT3] 49: IF (L>0) AND IntPol THEN
[ND4] 50: IF (ChkOld<>Chk) THEN BEGIN
[HV2] 51: T1:=T-Step; T2:=T;
[6X0] 52: CN:=0;
[DS4] 53: WHILE CN<NumbIntPol DO BEGIN
[902] 54: TI:=(T1+T2)/2;
[S34] 55: TToP(TI, PrRay,Dist,PI);
[BX3] 56: Check(PrRay,Dist, ChkI);
[0H4] 57: IF (ChkOld=ChkI) THEN T1:=TI
[K02] 58: ELSE T2:=TI;
[SA1] 59: CN:=CN+1;
[ZN0] 60: END;
[502] 61: ChkI:=TRUE;
[8D1] 62: IF ChkOld
[FT4] 63: THEN TToP(T1, PrRay,Dist,PI)
[3Y4] 64: ELSE TToP(T2, PrRay,Dist,PI);
[ZU4] 65: Draw(PI,ChkOld,ChkI,LL+1,Col);
[B52] 66: ChkOld:=ChkI;
[ZN0] 67: END;
[X04] 68: Draw(P,ChkOld,Chk,L,Col);
[712] 69: ChkOld:=Chk;
[ZN0] 70: END;
[ZN0] 71: END;
[KN0] 72: END.
```

Listing 4. »ucurve3d.pas« enthält Funktionen für Projektion und Sichtbarkeitstest.

```
[4E2] 38: PROCEDURE Call
[225] 39: (NRS: INTEGER; VAR RSP: RSPtr);
[ZN0] 40: END;
[DU3] 41: FUNCTION InIntervalUV
[H26] 42: (IUV: IntervalUV; Q: PtUV): BOOLEAN;
[1V2] 43: PROCEDURE DefIUV
[BS5] 44: (Dgr: BOOLEAN; U1,U2,V1,V2: EXTENDED;
[8U3] 45: VAR IUV: IntervalUV);
[0N3] 46: PROCEDURE PartitionU
[5R5] 47: (IUV: IntervalUV; NU,NNU: INTEGER;
[8S2] 48: VAR U: EXTENDED);
[1N3] 49: PROCEDURE PartitionV
[7R5] 50: (IUV: IntervalUV; NV,NNV: INTEGER;
[AS2] 51: VAR V: EXTENDED);
[HY2] 52: VAR NRS: INTEGER;
[DE1] 53: RS: RST;
[L32] 54: RSP: RSPtr;
[000] 55:
[4L2] 56: IMPLEMENTATION
[000] 57:
[863] 58: PROCEDURE RST.Init
[CK6] 59: (KofRSInit: INTEGER; O: Pt3D; UZ: Vt3D;
[EK4] 60: Dgr: BOOLEAN; Prm: EXTENDED;
[5V3] 61: InitIUV: IntervalUV);
[D17] 62: {O: Nullpunkt des 3D-Koordinatensystems
[7G5] 63: UZ: Einheitsvektor der Drehachse
[105] 64: Prm: Parameter der RS, im Falle:
[G84] 65: KofRS=1: Radius der Kugel
[966] 66: KofRS=2: Oeffnungswinkel des Kegels,
[PK3] 67: in Grad, wenn Dgr=TRUE
[HD5] 68: KofRS=3: Radius des Zylinders}
[MU0] 69: BEGIN
[NZ2] 70: KofRS:=KofRSInit;
[VM6] 71: DefineNewCS3DWithPt3D(O,UZ,Pr.C, CS3D);
[N32] 72: CASE KofRS OF
[GT4] 73: 1: BEGIN A:=-1; B:=SQR(Prm); END;
[551] 74: 2: BEGIN
[PE4] 75: IF Dgr THEN Prm:=Prm*PI/180;
[HW4] 76: A:=SQR(SIN(Prm)/COS(Prm)); B:=0;
[ZN0] 77: END;
[SR4] 78: 3: BEGIN A:=0; B:=SQR(Prm); END;
[ZN0] 79: END;
[WC2] 80: IUV:=InitIUV;
[Y62] 81: {KofCut:=1;}
[ZN0] 82: END;
[3W6] 83: FUNCTION RST.RSF(V: EXTENDED): EXTENDED;
[NE2] 84: {Drehkurve}
[MU0] 85: BEGIN
[9F4] 86: RSF:=SQRT(A*SQR(V)+B+Eps3);
[ZN0] 87: END;
[2E7] 88: PROCEDURE RST.UVToRS(Q: PtUV; VAR P: Pt3D);
[7E5] 89: {Berechnet zum Parameterpunkt Q
[705] 90: den 3D-Punkt P auf der Fläche}
[6A2] 91: VAR P3D: Pt3D;
[MU0] 92: BEGIN
[PK3] 93: P3D.X:=RSF(Q.V)*COS(Q.U);
[KX3] 94: P3D.Y:=RSF(Q.V)*SIN(Q.U);
[8V1] 95: P3D.Z:=Q.V;
[DH4] 96: New3DToWorld3D(CS3D,P3D, P);
[ZN0] 97: END;
[FB7] 98: PROCEDURE RST.RSToUV(P: Pt3D; VAR Q: PtUV);
[6A2] 99: VAR P3D: Pt3D;
[X32] 100: PCyl: PtCyl;
[4L6] 101: {Berechnet zum 3D-Punkt P auf der Fläche
[CT3] 102: den Parameterpunkt Q}
[MU0] 103: BEGIN
[W94] 104: World3DToNew3D(CS3D,P, P3D);
[KD3] 105: CartCylin(P3D, PCyl);
[PB2] 106: Q.U:=PCyl.Phi;
[K22] 107: Q.V:=PCyl.Z;
[ZN0] 108: END;
[ZT5] 109: PROCEDURE RST.IntersectionLineRS
[925] 110: (Ln3D: Line3D; VAR NOS: INTEGER;
[X43] 111: VAR T1,T2: EXTENDED;
[DM2] 112: VAR Q1,Q2: PtUV);
[9C5] 113: {Schneidet die Gerade Ln3D mit der
[C47] 114: Fläche; NOS ist Zahl der Schnittpunkte,
[PN6] 115: T1,T2 sind die Parameterwerte der Schnitt-
[9S5] 116: punkte auf der Geraden, Q1,Q2 sind die
[F25] 117: Schnittpunkte auf der Fläche}
[PD5] 118: VAR AA,BB,CC,D,E,F,G,LDV: EXTENDED;
[FL1] 119: P3D: Pt3D;
[LR1] 120: OOL: Vt3D;
[MU0] 121: BEGIN
[Z36] 122: ScalarProductVt3D(Ln3D.U,CS3D.UZ, E);
[H55] 123: DifferenceVt3D(Ln3D.O,CS3D.O, OOL);
[0F3] 124: LengthVt3D(OOL, LDV);
[XH5] 125: ScalarProductVt3D(OOL,CS3D.UZ, F);
[EE5] 126: ScalarProductVt3D(OOL,Ln3D.U, G);
[031] 127: D:=1+A;
[NA2] 128: AA:=1-SQR(E)*D;
[E22] 129: BB:=2*(G-E*F*D);
[M23] 130: CC:=-B-F*F*D+LDV*LDV;
[ST5] 131: SquareEquation(AA,BB,CC, NOS,T1,T2);
[E12] 132: IF NOS>0 THEN
[MU0] 133: BEGIN
[AL3] 134: TToLine3D(Ln3D,T1, P3D);
[EF2] 135: RSToUV(P3D, Q1);
```



```
[CL3] 136: TToLine3D(Ln3D,T2, P3D);
[HF2] 137: RSToUV(P3D, Q2);
[ZN0] 138: END;
[S33] 139: IF NOS=0 THEN BEGIN
[1F3] 140: Q1.U:=10000; Q1.V:=100000;
[8F3] 141: Q2.U:=10000; Q2.V:=100000;
[ZN0] 142: END;
[ZN0] 143: END;
[G94] 144: PROCEDURE RST.InFrontRS
[6S3] 145: (DistN,T1,T2: EXTENDED;
[UL3] 146: VAR InFrnt: BOOLEAN);
[PR6] 147: {Prüft, ob ein Punkt mit der Entfernung
[1A7] 148: DistN vom Projektionszentrum vorne liegt}
[MU0] 149: BEGIN
[UK3] 150: IF Null(DistN-T1,Eps3)
[F96] 151: THEN InFrnt:=TRUE ELSE InFrnt:=FALSE;
[ZN0] 152: END;
[G76] 153: FUNCTION RST.InIUUV(Q: PtUV): BOOLEAN;
[CZ5] 154: {Prüft, ob der Punkt P im Parameter-
[9H3] 155: intervall IUUV liegt}
[MU0] 156: BEGIN
[XY4] 157: InIUUV:=InIntervalUV(IUUV,Q);
[ZN0] 158: END;
[8G3] 159: PROCEDURE RST.Create
[225] 160: (NRS: INTEGER; VAR RSP: RSPtr);
[596] 161: {Reserviert für die NRS-te Fläche
[FA3] 162: Platz auf dem Heap}
[Z83] 163: VAR RSPNew: RSPtr;
[MU0] 164: BEGIN
[Z33] 165: IF NRS=1 THEN BEGIN
[0S1] 166: New(RSP1);
[AG3] 167: RSP1^.NextRSP:=NIL;
[PT1] 168: RSP:=RSP1;
[4M0] 169: END
[WP1] 170: ELSE BEGIN
[842] 171: New(RSPNew);
[0Z3] 172: RSPNew^.NextRSP:=NIL;
[SU3] 173: RSP^.NextRSP:=RSPNew;
[592] 174: RSP:=RSPNew;
[ZN0] 175: END;
[ZN0] 176: END;
[H43] 177: PROCEDURE RST.Call
[225] 178: (NRS: INTEGER; VAR RSP: RSPtr);
[GA6] 179: {Holt zur NRS-ten Fläche den Zeiger}
[9K2] 180: VAR N: INTEGER;
[MU0] 181: BEGIN
[NU3] 182: IF NRS=1 THEN RSP:=RSP1;
[343] 183: IF NRS>1 THEN BEGIN
[PT1] 184: RSP:=RSP1;
[H74] 185: FOR N:=1 TO NRS-1 DO BEGIN
[YB3] 186: RSP:=RSP^.NextRSP;
[ZN0] 187: END;
[ZN0] 188: END;
[ZN0] 189: END;
[DU3] 190: FUNCTION InIntervalUV
[H26] 191: (IUUV: IntervalUV; Q: PtUV): BOOLEAN;
[PL5] 192: {Prüft ob der Parameterpunkt Q im
[P45] 193: Parameterintervall IUUV liegt}
[MU0] 194: BEGIN
[5K3] 195: InIntervalUV:=TRUE;
[HH3] 196: IF (Q.U<IUUV[1].U-Eps4)
[1L3] 197: OR (Q.U>IUUV[2].U+Eps4)
[6L3] 198: OR (Q.V<IUUV[1].V-Eps4)
[6L3] 199: OR (Q.V>IUUV[2].V+Eps4)
[HB4] 200: THEN InIntervalUV:=FALSE;
[ZN0] 201: END;
[1V2] 202: PROCEDURE DefIUUV
[BS5] 203: (Dgr: BOOLEAN; U1,U2,V1,V2: EXTENDED;
[8U3] 204: VAR IUUV: IntervalUV);
[MU0] 205: BEGIN
[BM2] 206: IF Dgr THEN BEGIN
[P83] 207: IUUV[1].U:=U1*PI/180;
[T83] 208: IUUV[2].U:=U2*PI/180;
[4M0] 209: END
[WP1] 210: ELSE BEGIN
[RK4] 211: IUUV[1].U:=U1; IUUV[2].U:=U2;
[ZN0] 212: END;
[1L4] 213: IUUV[1].V:=V1; IUUV[2].V:=V2;
[ZN0] 214: END;
[0N3] 215: PROCEDURE PartitionU
[5R5] 216: (IUUV: IntervalUV; NU,NUU: INTEGER;
[8S2] 217: VAR U: EXTENDED);
[3V5] 218: {Wie Prozedur PartitionX in U1}
[MU0] 219: BEGIN
[SU6] 220: U:=IUUV[1].U+NU/NUU*(IUUV[2].U-IUUV[1].U);
[ZN0] 221: END;
[1N3] 222: PROCEDURE PartitionV
[7R5] 223: (IUUV: IntervalUV; NV,NNV: INTEGER;
[AS2] 224: VAR V: EXTENDED);
[4V5] 225: {Wie Prozedur PartitionY in U1}
[MU0] 226: BEGIN
[7V6] 227: V:=IUUV[1].V+NV/NNV*(IUUV[2].V-IUUV[1].V);
[ZN0] 228: END;
[KN0] 229: END.
```

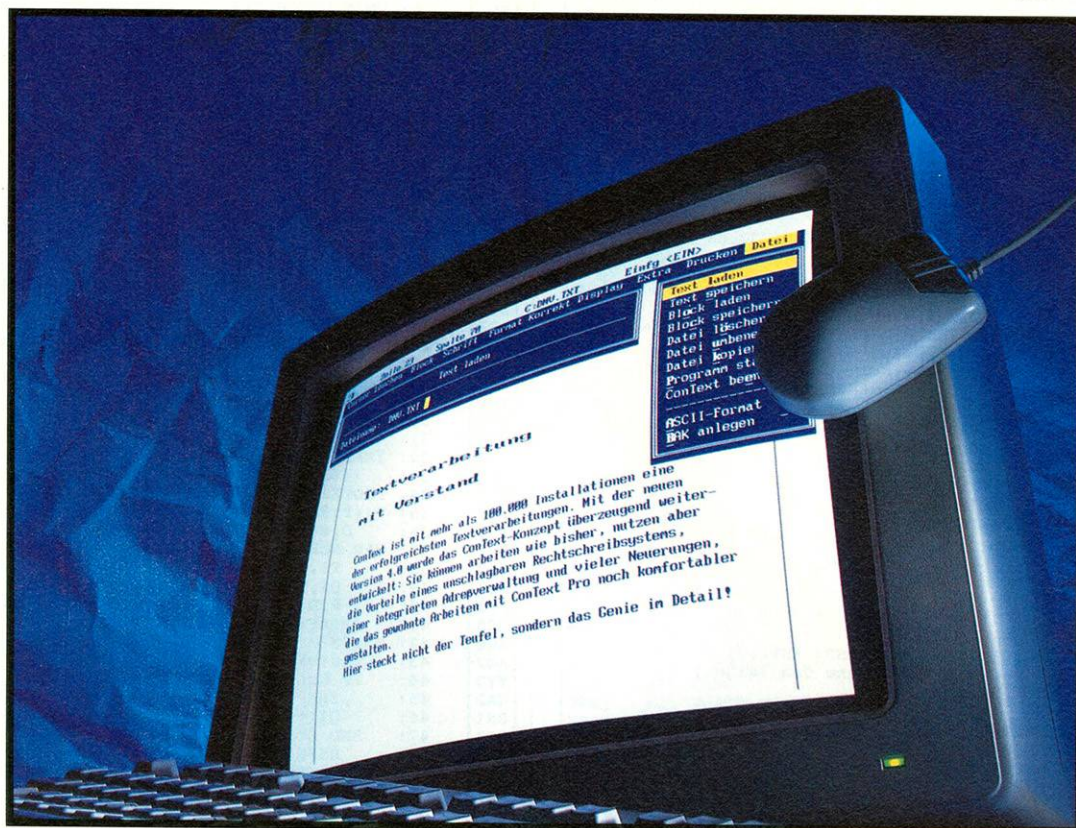
Listing 5. Die Unit »ugs.pas« stellt das Objekt zum Zeichnen von Rotationsflächen bereit.

```
[654] 1: {Programm: ugscheck.pas
[BF5] 2: Funktion: Rotationsflächen auf
[VG3] 3: Sichtbarkeit prüfen
[G64] 4: Sprache: Turbo Pascal 6.0
[GL3] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[PD2] 8: UNIT UGSCHECK;
[SK1] 9: INTERFACE
[2A5] 10: USES Crt, Graph, U1, U3DGEOM, U3D,
[EA2] 11: UCURVE3D, UGS;
[ST0] 12: TYPE
[BK4] 13: RSVisHiddT = OBJECT(Curve3DT)
[RY2] 14: RSPN,RSPJ: RSPtr;
[U23] 15: DrawFrnt: INTEGER;
[N22] 16: QCont: PtUV;
[EA2] 17: Cont: BOOLEAN;
[714] 18: PROCEDURE VisibilityRS
[AK5] 19: (PrRay: Line3D; RSPNinit: RSPtr;
[SS5] 20: DistN: EXTENDED; VAR Vis: BOOLEAN);
[R53] 21: PROCEDURE HiddenRS
[Y07] 22: (PrRay:Line3D; RSPNinit,RSPJinit:RSPtr;
[X55] 23: DistN: EXTENDED; VAR Hidd: BOOLEAN);
[ZN0] 24: END;
[RS4] 25: GSCheckT = OBJECT(RSVisHiddT)
[1A6] 26: KofCheck,KofDraw,NOS,Branch: INTEGER;
[EP2] 27: PROCEDURE Check
[ZA5] 28: (PrRay: Line3D; DistN: EXTENDED;
[XH4] 29: VAR Chk: BOOLEAN); VIRTUAL;
[ZN0] 30: END;
[000] 31:
[4L2] 32: IMPLEMENTATION
[000] 33:
[UZ5] 34: PROCEDURE RSVisHiddT.VisibilityRS
[AK5] 35: (PrRay: Line3D; RSPNinit: RSPtr;
[SS5] 36: DistN: EXTENDED; VAR Vis: BOOLEAN);
[8Y2] 37: VAR NOS: INTEGER;
[6E2] 38: T1,T2: EXTENDED;
[KU1] 39: Q1,Q2: PtUV;
[FL1] 40: P3D: Pt3D;
[U25] 41: InFrnt,InIUUV1,InIUUV2: BOOLEAN;
[MU0] 42: BEGIN
[A02] 43: Vis:=FALSE;
[YY2] 44: IF Cont THEN BEGIN
[GA2] 45: InFrnt:=TRUE;
[8N1] 46: Q1:=QCont;
[0C2] 47: END ELSE BEGIN
[W65] 48: RSPNinit^.IntersectionLineRS
[6W3] 49: (PrRay, NOS,T1,T2,Q1,Q2);
[W33] 50: IF NOS>0 THEN BEGIN
[DP3] 51: RSPNinit^.InFrontRS
[CE3] 52: (DistN,T1,T2, InFrnt);
[FK4] 53: If (DrawFrnt=1) AND NOT InFrnt
[LR1] 54: THEN EXIT;
[9Z3] 55: If (DrawFrnt=2) AND InFrnt
[LR1] 56: THEN EXIT;
[ZN0] 57: END;
[ZN0] 58: END;
[1F2] 59: CASE InFrnt OF
[LU4] 60: TRUE: IF RSPNinit^.InIUUV(Q1)
[DP2] 61: THEN Vis:=TRUE;
[TF5] 62: FALSE: IF NOT RSPNinit^.InIUUV(Q1)
[034] 63: AND RSPNinit^.InIUUV(Q2)
[DP2] 64: THEN Vis:=TRUE;
[ZN0] 65: END;
[ZN0] 66: END;
[P45] 67: PROCEDURE RSVisHiddT.HiddenRS
[XL2] 68: (PrRay: Line3D;
[FE4] 69: RSPNinit,RSPJinit: RSPtr;
[X55] 70: DistN: EXTENDED; VAR Hidd: BOOLEAN);
[SX3] 71: VAR NOS,K,J,N: Integer;
[6E2] 72: T1,T2: EXTENDED;
[KU1] 73: Q1,Q2: PtUV;
[UV3] 74: InIUUV1,InIUUV2: BOOLEAN;
[P52] 75: RSPK: RSPtr;
[MU0] 76: BEGIN
[312] 77: Hidd:=FALSE;
[FF2] 78: RSPK:=RS.RSP1;
[M44] 79: WHILE (RSPK<>NIL) DO BEGIN
[3P3] 80: IF NOT (RSPK=RSPNinit)
[0V5] 81: AND NOT (RSPK=RSPJinit) THEN BEGIN
[5F4] 82: RSPK^.IntersectionLineRS
[6W3] 83: (PrRay, NOS,T1,T2,Q1,Q2);
[W33] 84: IF NOS>0 THEN BEGIN
[MU0] 85: BEGIN
[XT2] 86: IF (T1<DistN+Eps3)
[525] 87: AND RSPK^.InIUUV(Q1) THEN BEGIN
[CM3] 88: Hidd:=TRUE; EXIT; END;
[0U2] 89: IF (T2<DistN+Eps3)
[825] 90: AND RSPK^.InIUUV(Q2) THEN BEGIN
[CM3] 91: Hidd:=TRUE; EXIT; END;
[ZN0] 92: END;
[ZN0] 93: END;
```

Listing 6. Mit der Unit »ugscheck.pas« prüfen Sie Rotationsflächen auf Sichtbarkeit.

CONTEXT 4.0

Die sympathische Textverarbeitung



Die überzeugende Kombination aus Leistung und Preis.

Sehen und Verstehen. — Laden und Loslegen: ConText 4.0 ist die durchdachte und schnelle Textverarbeitung mit breitangelegtem Funktionsspektrum und anwendergerechter Handhabung. — Und das zu einem überraschend freundlichen Preis.

Jetzt mit Schreibkomfort, den Sie nicht mehr missen möchten

■ **vielfältige Schriftgestaltung**

■ **WYSIWYG-Darstellung**

■ **Serienbrieffunktion**

■ **integrierte Adreßverwaltung**

■ **automatische Rechtschreibkorrektur**

■ **leichteste Formulargestaltung**

■ **Zeichnen mit dem Blockgrafik-Editor**

■ **arbeitsorientierte Online-Hilfe**

Technik: Kopieren zwischen den Textbereichen ■ Zeilen- und Spaltenblock ■ Kopf- und Fußzeilen ■ 2-spaltiger Umbruch ■ variables Seitenformat ■ fünf getrennte Textbereiche ■ Kopieren zwischen den Textbereichen ■ Drucken mit variablem Zeilenabstand (1, 1,5, 2) ■ zirka 300 neue Druckertreiber ■ Adreßverwaltung mit ASCII-Import/Export (dBase-Anschluß) ■ Tastatur-Makros ■ Wordstar-kompatible Befehle ■ ASCII-Sonderzeichenmenü ■ Kalender ■ DOS-Shell ■ EMS-Unterstützung.

Ergonomie: SAA-angelegte Bedienerführung ■ komfortable Dateiauswahlbox mit Wildcards ■ Kommentarzeilen für Dateien ■ bis zu 60-Zeilen-Bildschirm darstellbar ■ individuelle Farbeinstellung ■ Suchen- und Ersetzen mit Wildcards ■ Rechnen im Text (u. a. Trigonometrie, Spaltensumme) ■ Formulare: einfach gestalten und ausfüllen mit dem Grafikeditor und dem 99-Merksystem ■ Zeichnen mit der Maus.

Sicherheit: leistungsfähige Undo-Funktion ■ Zwischenspeichern ■ automatische Rechtschreibkorrektur beim Schreiben oder nachträglich; Lexikon: zuschaltbar, lernfähig und individuell erweiterbar ■ Online-Hilfesystem

Systemanforderungen: für PCs mit 512 KByte RAM, alle Grafikkarten, Festplatte für Rechtschreibkorrektur und Adreßdatenbank, DOS ab 3.2

Achtung Update-Service: Sie schicken uns als Benutzernachweis entweder eine Kopie Ihrer Kaufrechnung, Ihre Original-ConText-Disketten oder die Original-Titelseite Ihres ConText-Handbuches.

ConText 4.0

249,— * DM

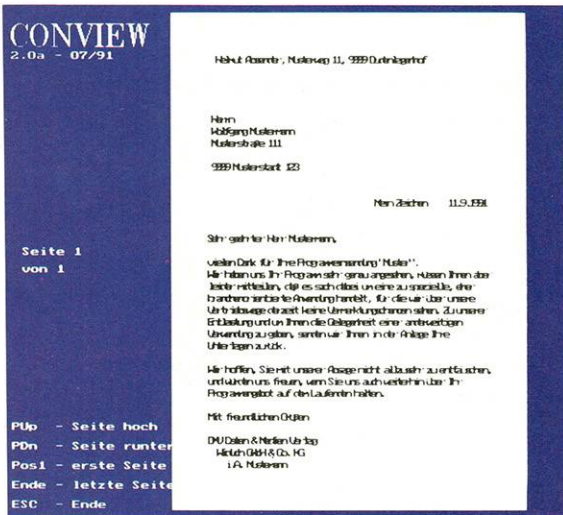
ConText Update

99,— * DM

CONVIEW

Sparen Sie Zeit, Geld und Nerven

ConView zeigt Ihnen eine Seite aus Ihrem Text verkleinert in einer Seitenübersicht. Vorteil dieser Funktion ist die sofortige Kontrolle Ihres Textlayouts vor dem Ausdrucken. ConView spart Ihnen damit jede Menge Wartezeit beim Drucken und vor allem Kosten für Papier, Toner oder Farbband.



Kontrolle ist besser

Die optische Vorprüfung der Textseiten am Bildschirm, gewährleistet, daß Ihr flüssiger Arbeitsstil nicht von Probeausdrucken, Warteschlangen und ständigen Wiederholungen unterbrochen wird. Sie können überprüfen, ob zum Beispiel Ihr Brief harmonisch gegliedert, Ihr Textumbruch gelungen, die Fußnote richtig platziert oder Ihr langer Text auf den Seiten richtig verteilt ist. ConView fügt sich nahtlos in ConText 4.0 ein und wird von dort per Tastendruck aufgerufen.

DM 49,—*

WORKSHOP

Das Zusatzpaket für den engagierten ConText-Nutzer

ConText Workshop liefert eine Fülle nützlicher Informationen und zusätzlicher Software, damit Sie Ihr ConText optimal auf den persönlichen Einsatz anpassen können. Die umfangreichen und verständlichen Zusatzinformationen lassen diese Anpassung für Sie zu einem Kinderspiel werden.

- Sie möchten die vorhandenen ConText-Schriften verändern? Der Font-Editor gibt Ihnen die Möglichkeit, vorhandene Zeichen zu verändern, eigene Schriften zu erstellen und alles auf dem Monitor auszugeben.
- Mit ConText Workshop reizen Sie die Fähigkeiten Ihres Druckers voll aus, indem Sie auf trickreiche Weise spezielle Effekte erzeugen. Sie erhalten eine Fülle von Informationen im Umgang mit dem Druckertreiber, Erklärungen aller Druckerzeichen und -funktionen und spezielle Druckertreiber.
- Das Zusatzpaket ermöglicht Ihnen die optimale Anpassung Ihrer Grafikkarte. Sie erreichen eine höhere Bildschirmauflösung, ein effektiveres Layout und sorgen für mehr Informationen auf dem Monitor (bis zu 100 Zeichen pro Zeile).
- ConText Workshop bietet ein neues variables Hilfesystem, in das Sie auf einfachste Art und Weise eigene Texte und Notizen einbauen können. Außerdem erfahren Sie alles über neue und nicht-dokumentierte ConText-Befehle, u.v.a.m.

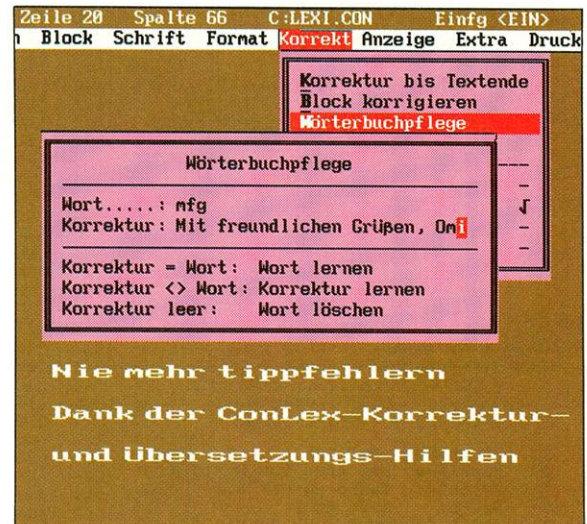
Lieferumfang: Spezialdruckertreiber (EPSONF.CDT, HPDESK.CDT, IBMTOHP.CDT)
 • Zusatzprogramme (KEYSIM.COM, FED.EXE, HCOMP.EXE, LFONT.EXE, LMODE.EXE, ZITAT.EXE)
 • Bildschirmfont (ZSATZ14.FNT, ZSATZ16.FNT, ZSATZ6.FNT, ZSATZ8.FNT)
 • Überweisungsformular (BANK.FOR) • Grafikkartentreiber (ATT.OVL, VGA800.OVL, VGA801.OVL, VGAMONO.OVL, VGATTL.OVL, VGAX.OVL) u.a. für LCD-Displays, Toshiba (T1000 SE/XE/LE, T1200 XE, T1600) • Textgrundlagen (HILFE.TXT, PHILFE.TXT, ZITATE.TXT, LIST.TXT) • Benutzerhandbuch (präzise und verständliche Anleitungen);

DM 59,—*

CONLEX - Rechtschreiblexikon Deutsch

Mit dem ConLex-Rechtschreiblexikon erweitern Sie Ihre ConText-Korrekturbibliothek auf 230.000 Wörter. Diesen umfangreichen Wortschatz können Sie beliebig mit eigenen Begriffen erweitern und damit Ihr persönliches Wörterbuch erstellen.

DM 49,—*



CONLEX - Übersetzungshilfe für Einsteiger

Deutsch-Englisch oder Englisch-Deutsch

Die Übersetzungshilfe für Einsteiger Deutsch-Englisch hilft Ihnen deutsche Texte Wort für Wort ins Englische zu übersetzen. Selbstverständlich können Sie diese Funktion wahlweise auch nachträglich für schon vorhandene Texte nutzen. Sofern Sie englische Texte Wort für Wort ins Deutsche übersetzen wollen ist Ihnen die Übersetzungshilfe Englisch-Deutsch dabei eine unschätzbare Hilfe.

je DM 99,—*

CONLEX - Übersetzungshilfe für Sprachkenner

Deutsch-Englisch oder Englisch-Deutsch

Sie verfügen über englische Sprachkenntnisse und verfassen einen englischen Text, dann korrigiert Ihnen die Übersetzungshilfe für Sprachkenner deutsch-englisch automatisch eventuelle Schreibfehler und übersetzt, sofern eine Vokabel fehlt, das eingegebene deutsche Wort ins Englische.

Wenn Sie englische Texte ins Deutsche übertragen wollen, korrigiert Ihnen die Übersetzungshilfe englisch-deutsch den eingegebenen Text auf Tippfehler hin und übersetzt die eventuell fehlende Vokabel vom Englischen ins Deutsche.

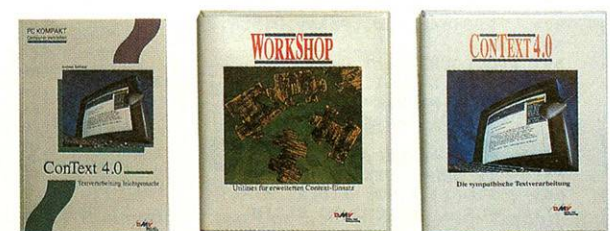
je DM 99,—*

ConText 4.0

Textverarbeitung leichtgemacht

ConText ist mit weit über 100.000 Installationen eines der verbreitetsten Textsysteme. Das Buch gibt am Beispiel dieser Software eine detaillierte Einführung in die Textverarbeitung und führt nahtlos in den effektiven Umgang mit der neuen ConText-Version 4.0 ein. Weitere Kapitel vermitteln die richtige Handhabung von Textbereichen, Textbausteinen und sogenannten Tastaturmakros.

DM 19,80*



* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.


```
[ZN0] 94: END;
[CU3] 95: RSPK:=RSPK^.NextRSP;
[ZN0] 96: END;
[ZN0] 97: END;
[234] 98: PROCEDURE GSCheckT.Check
[ZA5] 99: (PrRay: Line3D; DistN: EXTENDED;
[822] 100: VAR Chk: BOOLEAN);
[G63] 101: VAR VisRSN,VisRSJ,
[XS2] 102: HiddRS: BOOLEAN;
[MU0] 103: BEGIN
[1Y1] 104: Chk:=FALSE;
[2A2] 105: VisibilityRS
[5M4] 106: (PrRay,RSPN,DistN, VisRSN);
[FM4] 107: IF NOT VisRSN THEN EXIT ELSE
[EE1] 108: HiddenRS
[2E5] 109: (PrRay,RSPN,RSPJ,DistN, HiddRS);
[TR3] 110: IF NOT HiddRS THEN BEGIN
[EF3] 111: Chk:=TRUE; EXIT; END;
[ZN0] 112: END;
[KN0] 113: END.
```

Listing 6. Mit der Unit »ugscheck.pas« prüfen Sie Rotationsflächen auf Sichtbarkeit.

```
[L34] 69: FOR NU:=0 TO NNU DO BEGIN
[MU3] 70: IF J=1 THEN IF (NU MOD 7 < 3)
[BV4] 71: THEN Col:=Col1 ELSE Col:=Col2;
[RU3] 72: IF J=2 THEN IF (NU MOD 7 < 3)
[NV4] 73: THEN Col:=Col3 ELSE Col:=Col4;
[OM5] 74: PartitionU(RSPN^.IUV,NU,NNU, U);
[8W3] 75: DrawCurve3D(IID,LL,Col);
[4X0] 76: Stop;
[ZN0] 77: END;
[ZN0] 78: END;
[ZN0] 79: END;
[KN0] 80: END.
```

Listing 7. Das Objekt zum Zeichnen von Meridianen

```
[YF3] 1: {Programm: ucirc.pas
[8B6] 2: Funktion: Zeichnen von Breitenkreisen
[G64] 3: Sprache: Turbo Pascal 6.0
[G13] 4: Autor: Kurt Endl
[585] 5: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 6:
[TZ1] 7: UNIT UCIRC;
[SK1] 8: INTERFACE
[KX5] 9: USES Crt, Graph, U1, UDRAW, UCURVE2D,
[4W4] 10: UDRAW2D, U2DGEOM, U3DGEOM, U3D,
[BW3] 11: UCURVE3D, UGS, UGSHECK;
[ST0] 12: TYPE
[PZ3] 13: CircT = OBJECT(GSCheckT)
[T84] 14: V,OscFac,OscFr: EXTENDED;
[Y82] 15: Damp: BOOLEAN;
[AP2] 16: IIDV: IntervalID;
[413] 17: CONSTRUCTOR Init
[GX2] 18: (RSPNInit: RSPtr;
[NB5] 19: OscFacInit,OscFrInit: EXTENDED;
[863] 20: DampInit: BOOLEAN);
[DH2] 21: PROCEDURE TTop
[9A5] 22: (T: EXTENDED; VAR PrRay: Line3D;
[TC3] 23: VAR DistN: EXTENDED;
[1L3] 24: VAR P: Pt2D); VIRTUAL;
[4F2] 25: PROCEDURE Draw
[Z16] 26: (NNV,LL,Col1,Col2,Col3,Col4: INTEGER);
[ZN0] 27: END;
[4R2] 28: VAR Circ: CircT;
[000] 29:
[4L2] 30: IMPLEMENTATION
[000] 31:
[E24] 32: CONSTRUCTOR CircT.Init
[GX2] 33: (RSPNInit: RSPtr;
[NB5] 34: OscFacInit,OscFrInit: EXTENDED;
[863] 35: DampInit: BOOLEAN);
[MU0] 36: BEGIN
[7V2] 37: RSPN:=RSPNInit;
[TB3] 38: OscFac:=OscFacInit;
[1Y2] 39: OscFr:=OscFrInit;
[FL4] 40: IIDV[1].X:=RSPN^.IUV[1].V;
[NL4] 41: IIDV[2].X:=RSPN^.IUV[2].V;
[ZN0] 42: END;
[BL3] 43: PROCEDURE CircT.TTop
[9A5] 44: (T: EXTENDED; VAR PrRay: Line3D;
[MG5] 45: VAR DistN: EXTENDED; VAR P: Pt2D);
[H12] 46: VAR Q: PtUV;
[FL1] 47: P3D: Pt3D;
[MU0] 48: BEGIN
[YK3] 49: Q.U:=RSPN^.IUV[1].U
[S26] 50: +T*(RSPN^.IUV[2].U-RSPN^.IUV[1].U);
[WL4] 51: Q.V:=V+OscFac*COS(OscFr*Q.U)
[5P2] 52: *Damping(IIDV,V);
[6K3] 53: RSPN^.UVToRS(Q, P3D);
[RR4] 54: Project(P3D, PrRay,DistN,P);
[ZN0] 55: END;
[MG3] 56: PROCEDURE CircT.Draw
[Z16] 57: (NNV,LL,Col1,Col2,Col3,Col4: INTEGER);
[RK3] 58: VAR NV,Col,J: INTEGER;
[5N2] 59: IID: IntervalID;
[MU0] 60: BEGIN
[332] 61: Cont:=FALSE;
[9U3] 62: IID[1].X:=0; IID[2].X:=1;
[0F3] 63: FOR J:=1 TO 2 DO BEGIN
[N32] 64: DrawFrnt:=J;
[R34] 65: FOR NV:=0 TO NNV DO BEGIN
[RU3] 66: IF J=1 THEN IF (NV MOD 7 < 3)
[BV4] 67: THEN Col:=Col1 ELSE Col:=Col2;
[UV3] 68: IF J=2 THEN IF (NV MOD 7 < 3)
[NV4] 69: THEN Col:=Col3 ELSE Col:=Col4;
[EM5] 70: PartitionV(RSPN^.IUV,NV,NNV, V);
[8W3] 71: DrawCurve3D(IID,LL,Col);
[4X0] 72: Stop;
[ZN0] 73: END;
[ZN0] 74: END;
[ZN0] 75: END;
[KN0] 76: END.
```

Listing 8. Breitenkreise zeichnen Sie mit »ucirc.pas«.

```
[XF3] 1: {Programm: umer.pas
[UK5] 2: Funktion: Zeichnen von Meridianen
[G64] 3: Sprache: Turbo Pascal 6.0
[G13] 4: Autor: Kurt Endl
[585] 5: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 6:
[TS1] 7: UNIT UMER;
[SK1] 8: INTERFACE
[KX5] 9: USES Crt, Graph, U1, UDRAW, UCURVE2D,
[4W4] 10: UDRAW2D, U2DGEOM, U3DGEOM, U3D,
[BW3] 11: UCURVE3D, UGS, UGSHECK;
[ST0] 12: TYPE
[YR3] 13: MerT = OBJECT(GSCheckT)
[R84] 14: U,OscFac,OscFr: EXTENDED;
[Y82] 15: Damp: BOOLEAN;
[9P2] 16: IIDU: IntervalID;
[413] 17: CONSTRUCTOR Init
[GX2] 18: (RSPNInit: RSPtr;
[NB5] 19: OscFacInit,OscFrInit: EXTENDED
[863] 20: DampInit: BOOLEAN);
[DH2] 21: PROCEDURE TTop
[9A5] 22: (T: EXTENDED; VAR PrRay: Line3 ;
[TC3] 23: VAR DistN: EXTENDED;
[1L3] 24: VAR P: Pt2D); VIRTUAL;
[4F2] 25: PROCEDURE Draw
[Y16] 26: (NNU,LL,Col1,Col2,Col3,Col4: INTEGER);
[ZN0] 27: END;
[XH2] 28: VAR Mer: MerT;
[000] 29:
[4L2] 30: IMPLEMENTATION
[000] 31:
[NW3] 32: CONSTRUCTOR MerT.Init
[GX2] 33: (RSPNInit: RSPtr;
[NB5] 34: OscFacInit,OscFrInit: EXTENDED;
[863] 35: DampInit: BOOLEAN);
[MU0] 36: BEGIN
[7V2] 37: RSPN:=RSPNInit;
[TB3] 38: OscFac:=OscFacInit;
[1Y2] 39: OscFr:=OscFrInit;
[1P2] 40: Damp:=DampInit;
[CL4] 41: IIDU[1].X:=RSPN^.IUV[1].U;
[KL4] 42: IIDU[2].X:=RSPN^.IUV[2].U;
[ZN0] 43: END;
[NF3] 44: PROCEDURE MerT.TTop
[9A5] 45: (T: EXTENDED; VAR PrRay: Line3D;
[TC3] 46: VAR DistN: EXTENDED;
[Z21] 47: VAR P: Pt2D);
[H12] 48: VAR Q: PtUV;
[FL1] 49: P3D: Pt3D;
[MU0] 50: BEGIN
[6L3] 51: Q.V:=RSPN^.IUV[1].V
[X26] 52: +T*(RSPN^.IUV[2].V-RSPN^.IUV[1].V);
[ZF6] 53: IF Damp THEN Q.U:=U+OscFac*COS(OscFr*Q.V)
[9K2] 54: *Damping(IIDU,U)
[ZD5] 55: ELSE Q.U:=U+OscFac*COS(OscFr*Q.V);
[6K3] 56: RSPN^.UVToRS(Q, P3D);
[RR4] 57: Project(P3D, PrRay,DistN,P);
[ZN0] 58: END;
[LE3] 59: PROCEDURE MerT.Draw
[Y16] 60: (NNU,LL,Col1,Col2,Col3,Col4: INTEGER);
[4P4] 61: VAR NU,NNU,Col,J: INTEGER;
[5N2] 62: IID: IntervalID;
[MU0] 63: BEGIN
[332] 64: Cont:=FALSE;
[M33] 65: IF NNU=0 THEN EXIT;
[9U3] 66: IID[1].X:=0; IID[2].X:=1;
[0F3] 67: FOR J:=1 TO 2 DO BEGIN
[N32] 68: DrawFrnt:=J;
```



```

[G94] 1: {Programm: ucontour.pas
[UE5] 2: Funktion: Zeichnen von Konturen
[G64] 3: Sprache: Turbo Pascal 6.0
[G13] 4: Autor: Kurt Endl
[585] 5: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 6:
[4H2] 7: UNIT UCONTOUR;
[SK1] 8: INTERFACE
[KK5] 9: USES Crt, Graph, U1, UDRAW, UCURVE2D,
[4W4] 10: UDRAW2D, U2DGEOM, U3DGEOM, U3D,
[BW3] 11: UCURVE3D, UGS, UGSHECK;
[ST0] 12: TYPE
[ZS4] 13: Contour1T = OBJECT(GSCheckT)
[PB2] 14: Rad: EXTENDED;
[EZ1] 15: Pln: Plane;
[136] 16: CONSTRUCTOR Init(RSPNInit: RSPtr);
[DH2] 17: PROCEDURE TToP
[9A5] 18: (T: EXTENDED; VAR PrRay: Line3D;
[TC3] 19: VAR DistN: EXTENDED;
[1L3] 20: VAR P: Pt2D); VIRTUAL;
[855] 21: PROCEDURE Draw(LL,Col: INTEGER);
[ZN0] 22: END;
[0T4] 23: Contour2T = OBJECT(GSCheckT)
[DY1] 24: U: EXTENDED;
[TB3] 25: NoContour: BOOLEAN;
[136] 26: CONSTRUCTOR Init(RSPNInit: RSPtr);
[DH2] 27: PROCEDURE TToP
[9A5] 28: (T: EXTENDED; VAR PrRay: Line3D;
[TC3] 29: VAR DistN: EXTENDED;
[1L3] 30: VAR P: Pt2D); VIRTUAL;
[855] 31: PROCEDURE Draw(LL,Col: INTEGER);
[ZN0] 32: END;
[1T4] 33: Contour3T = OBJECT(GSCheckT)
[DY1] 34: U: EXTENDED;
[136] 35: CONSTRUCTOR Init(RSPNInit: RSPtr);
[DH2] 36: PROCEDURE TToP
[9A5] 37: (T: EXTENDED; VAR PrRay: Line3D;
[TC3] 38: VAR DistN: EXTENDED;
[1L3] 39: VAR P: Pt2D); VIRTUAL;
[855] 40: PROCEDURE Draw(LL,Col: INTEGER);
[ZN0] 41: END;
[HH4] 42: ContourT = OBJECT(GSCheckT)
[MK5] 43: PROCEDURE Init(RSPNInit: RSPtr);
[855] 44: PROCEDURE Draw(LL,Col: INTEGER);
[ZN0] 45: END;
[W04] 46: VAR Contour: ContourT;
[000] 47:
[4L2] 48: IMPLEMENTATION
[000] 49:
[XK7] 50: CONSTRUCTOR Contour1T.Init(RSPNInit:RSPtr);
[G83] 51: VAR CM,SS: EXTENDED;
[VN2] 52: DV,SCV,MCO: Vt3D;
[MU0] 53: BEGIN
[7V2] 54: RSPN:=RSPNInit;
[P26] 55: DifferenceVt3D(Pr.C,RSPN^.CS3D.O, DV);
[NA3] 56: ScaleVt3D(DV,1, SCV);
[W03] 57: LengthVt3D(DV, CM);
[5P2] 58: SS:=RSPN^.B/CM;
[FT3] 59: LinearCombinationVt3D
[CK4] 60: (1,SS,RSPN^.CS3D.O,SCV, MCO);
[VE3] 61: DefinePlaneWithVt3D
[0U4] 62: (MCO,SCV,RSPN^.CS3D.UZ, Pln);
[1P4] 63: Rad:=SQRT(RSPN^.B-SQR(SS));
[ZN0] 64: END;
[4B4] 65: PROCEDURE Contour1T.TToP
[9A5] 66: (T: EXTENDED; VAR PrRay: Line3D;
[MG5] 67: VAR DISTN: EXTENDED; VAR P: Pt2D);
[WD2] 68: VAR PPl: Pt2D;
[FL1] 69: P3D: Pt3D;
[MU0] 70: BEGIN
[TU2] 71: PPl.X:=Rad*COS(T);
[GU2] 72: PPl.Y:=Rad*SIN(T);
[4S3] 73: PlaneTo3D(Pln,PPl, P3D);
[RR4] 74: Project(P3D, PrRay,DistN,P);
[ZN0] 75: END;
[977] 76: PROCEDURE Contour1T.Draw(LL,Col: INTEGER);
[WB3] 77: VAR Ild: IntervallD;
[MU0] 78: BEGIN
[332] 79: Cont:=FALSE;
[H02] 80: DrawFrnt:=1;
[RB4] 81: Ild[1].X:=0; Ild[2].X:=2*PI;
[8W3] 82: DrawCurve3D(Ild,LL,Col);
[4X0] 83: Stop;
[ZN0] 84: END;
[000] 85:
[YK7] 86: CONSTRUCTOR Contour2T.Init(RSPNInit:RSPtr);
[U56] 87: VAR Den,Num,COSU,SINU,LOC: EXTENDED;
[XC1] 88: OC: Vt3D;
[MU0] 89: BEGIN
[7V2] 90: RSPN:=RSPNInit;
[F36] 91: DifferenceVt3D(Pr.C,RSPN^.CS3D.O, OC);
[Y17] 92: ScalarProductVt3D(OC,RSPN^.CS3D.UX, Den);
[447] 93: ScalarProductVt3D(OC,RSPN^.CS3D.UZ, Num);
[F23] 94: LengthVt3D(OC, LOC);
[KB2] 95: IF ABS(NUM/LOC)>
[485] 96: COS(ARCTAN(SQRT(RSPN^.A))) THEN
[XL3] 97: NoContour:=TRUE ELSE
[603] 98: NoContour:=FALSE;
[FU4] 99: COSU:=SQRT(RSPN^.A)*Num/Den;

```

```

[KC4] 100: IF 1-COSU*COSU<0 THEN EXIT;
[004] 101: SINU:=SQRT(1-COSU*COSU);
[S06] 102: IF Num<0 THEN U:=ARCTAN(SINU/COSU)+PI
[GD4] 103: ELSE U:=ARCTAN(SINU/COSU);
[M43] 104: IF U<0 THEN U:=U+2*PI;
[ZN0] 105: END;
[7B4] 106: PROCEDURE Contour2T.TToP
[9A5] 107: (T: EXTENDED; VAR PrRay: Line3D;
[MG5] 108: VAR DISTN: EXTENDED; VAR P: Pt2D);
[6A2] 109: VAR P3D: Pt3D;
[MU0] 110: BEGIN
[CC4] 111: QCont.V:=RSPN^.IUUV[1].V+
[ZV5] 112: T*(RSPN^.IUUV[2].V-RSPN^.IUUV[1].V);
[U02] 113: QCont.U:=U;
[T94] 114: RSPN^.UVToRS(QCont, P3D);
[RR4] 115: Project(P3D, PrRay,DistN,P);
[ZN0] 116: END;
[C77] 117: PROCEDURE Contour2T.Draw(LL,Col: INTEGER);
[FY2] 118: VAR J,II: INTEGER;
[5N2] 119: Ild: IntervallD;
[MU0] 120: BEGIN
[123] 121: IF NoContour THEN EXIT;
[H12] 122: Cont:=TRUE;
[9U3] 123: Ild[1].X:=0; Ild[2].X:=1;
[0F3] 124: FOR J:=1 TO 2 DO BEGIN
[UK3] 125: FOR II:=1 TO 2 DO BEGIN
[FX4] 126: IF II=1 THEN U:=U ELSE U:=-U+2*PI;
[A54] 127: IF (RSPN^.IUUV[1].U<U) AND
[924] 128: (U<RSPN^.IUUV[2].U) THEN
[8W3] 129: DrawCurve3D(Ild,LL,Col);
[ZN0] 130: END;
[ZN0] 131: END;
[332] 132: Cont:=FALSE;
[ZN0] 133: END;
[000] 134:
[ZX7] 135: CONSTRUCTOR Contour3T.Init(RSPNInit:RSPtr);
[HH5] 136: VAR Den,Num,COSU,SINU: EXTENDED;
[XC1] 137: OC: Vt3D;
[MU0] 138: BEGIN
[7V2] 139: RSPN:=RSPNInit;
[F36] 140: DifferenceVt3D(Pr.C,RSPN^.CS3D.O, OC);
[Y17] 141: ScalarProductVt3D(OC,RSPN^.CS3D.UX, Den);
[AD6] 142: COSU:=RSPN^.RSF(RSPN^.IUUV[1].V)/Den;
[KC4] 143: IF 1-COSU*COSU<0 THEN EXIT;
[004] 144: SINU:=SQRT(1-COSU*COSU);
[SK3] 145: U:=ARCTAN(SINU/COSU);
[M43] 146: IF U<0 THEN U:=U+2*PI;
[ZN0] 147: END;
[AB4] 148: PROCEDURE Contour3T.TToP
[9A5] 149: (T: EXTENDED; VAR PrRay: Line3D;
[MG5] 150: VAR DISTN: EXTENDED; VAR P: Pt2D);
[6A2] 151: VAR P3D: Pt3D;
[MU0] 152: BEGIN
[CC4] 153: QCont.V:=RSPN^.IUUV[1].V+
[ZV5] 154: T*(RSPN^.IUUV[2].V-RSPN^.IUUV[1].V);
[U02] 155: QCont.U:=U;
[T94] 156: RSPN^.UVToRS(QCont, P3D);
[RR4] 157: Project(P3D, PrRay,DistN,P);
[ZN0] 158: END;
[F77] 159: PROCEDURE Contour3T.Draw(LL,Col: INTEGER);
[FY2] 160: VAR J,II: INTEGER;
[5N2] 161: Ild: IntervallD;
[MU0] 162: BEGIN
[H12] 163: Cont:=TRUE;
[9U3] 164: Ild[1].X:=0; Ild[2].X:=1;
[0F3] 165: FOR J:=1 TO 2 DO BEGIN
[UK3] 166: FOR II:=1 TO 2 DO BEGIN
[FX4] 167: IF II=1 THEN U:=U ELSE U:=-U+2*PI;
[A54] 168: IF (RSPN^.IUUV[1].U<U) AND
[924] 169: (U<RSPN^.IUUV[2].U) THEN
[8W3] 170: DrawCurve3D(Ild,LL,Col);
[ZN0] 171: END;
[ZN0] 172: END;
[4X0] 173: Stop;
[332] 174: Cont:=FALSE;
[ZN0] 175: END;
[097] 176: PROCEDURE ContourT.Init(RSPNInit: RSPtr);
[MU0] 177: BEGIN
[7V2] 178: RSPN:=RSPNInit;
[ZN0] 179: END;
[T84] 180: PROCEDURE ContourT.Draw;
[684] 181: VAR Contour1: Contour1T;
[LH3] 182: Contour2: Contour2T;
[RH3] 183: Contour3: Contour3T;
[MU0] 184: BEGIN
[ND3] 185: CASE RSPN^.KofRS OF
[351] 186: 1: BEGIN
[HF3] 187: Contour1.Init(RSPN);
[MS3] 188: Contour1.Draw(LL,Col);
[ZN0] 189: END;
[551] 190: 2: BEGIN
[KF3] 191: Contour2.Init(RSPN);
[NS3] 192: Contour2.Draw(LL,Col);
[ZN0] 193: END;
[751] 194: 3: BEGIN

```

Listing 9. »ucontour.pas« enthält Objekte zum Zeichnen von Konturen.


```
[LF3] 195: Contour3.Init(RSPN);
[PS3] 196: Contour3.Draw(LL,Col);
[ZN0] 197: END;
[ZN0] 198: END;
[ZN0] 199: END;
[KN0] 200: END.
```

Listing 9. »ucontour.pas« enthält Objekte zum Zeichnen von Konturen.

```
[813] 1: {Programm: rs1.pas
[K36] 2: Funktion: Demonstriert das Zeichnen
[H23] 3: von Meridianen und
[FH2] 4: Breitenkreisen
[G64] 5: Sprache: Turbo Pascal 6.0
[G13] 6: Autor: Kurt Endl
[585] 7: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 8:
[G32] 9: PROGRAM RS1;
[2A5] 10: USES Crt, Graph, U1, U3DGEOM, U3D,
[UM4] 11: UGS, UMER, UCIRC, UCONTOUR;
[GP2] 12: PROCEDURE Draw;
[HY2] 13: VAR NRS: INTEGER;
[9E2] 14: Prm: EXTENDED;
[7W2] 15: O: Pt3D; UZ: Vt3D;
[D03] 16: IUV: IntervalUV;
[MU0] 17: BEGIN
[GP1] 18: WP:=FALSE;
[UX1] 19: RS.NNRS:=3;
[000] 20:
[AT2] 21: RS.Create(1, RSP);
[SX2] 22: DefPt3D(-40,-60,0,0);
[T35] 23: DirectionVt3D(TRUE,1,220,65,UZ);
[VF4] 24: DefIUV(TRUE,40,320,-40,40, IUV);
[PX4] 25: RSP^.Init(1,0,UZ,TRUE,50,IUV);
[000] 26:
[ET2] 27: RS.Create(2, RSP);
[AX2] 28: DefPt3D(0,70,-80,0);
[945] 29: DirectionVt3D(TRUE,1,260,65,UZ);
[1G4] 30: DefIUV(TRUE,40,320,10,160, IUV);
[KX4] 31: RSP^.Init(2,0,UZ,TRUE,18,IUV);
[000] 32:
[KT2] 33: RS.Create(3, RSP);
[763] 34: DefPt3D(80,-110,60,0);
[Z95] 35: DirectionVt3D(TRUE,1,190,-35,UZ);
[1H4] 36: DefIUV(TRUE,40,320,0,160, IUV);
[KX4] 37: RSP^.Init(3,0,UZ,TRUE,20,IUV);
[000] 38:
[5W4] 39: FOR NRS:=1 TO RS.NNRS DO BEGIN
[N13] 40: RS.Call(NRS, RSP);
[4Z1] 41: CASE NRS OF
[351] 42: 1: BEGIN
[3A4] 43: Mer.Init(RSP, 0.2,0.2, TRUE);
[8D3] 44: Mer.Draw(51,66, 1,2,3,4);
[864] 45: Circ.Init(RSP, 2,8, FALSE);
[6A3] 46: Circ.Draw(1,66,1,2,3,4);
[ZN0] 47: END;
[551] 48: 2: BEGIN
[3A4] 49: Mer.Init(RSP, 0.2,0.2, TRUE);
[463] 50: Mer.Draw(1,66, 1,4,1,2);
[864] 51: Circ.Init(RSP, 2,8, FALSE);
[TF3] 52: Circ.Draw(51,66,1,4,3,2);
[ZN0] 53: END;
[751] 54: 3: BEGIN
[3A4] 55: Mer.Init(RSP, 0.2,0.2, TRUE);
[WC3] 56: Mer.Draw(30,66, 1,1,4,4);
[864] 57: Circ.Init(RSP, 2,8, FALSE);
[RF3] 58: Circ.Draw(51,66,1,1,4,4);
[ZN0] 59: END;
[ZN0] 60: END;
[B73] 61: Contour.Init(RSP);
[A53] 62: Contour.Draw(66,1);
[ZN0] 63: END;
[ZN0] 64: END;
[MU0] 65: BEGIN
[A86] 66: NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 67: InitializeGraphic;
[5W4] 68: IntPol:=TRUE; NumbIntPol:=4;
[Z35] 69: DefI3D(-100,-100,-100,100,100,100, WI3D);
[YD4] 70: DefPtSph(6000,270,0, PrSph.C);
[7E4] 71: DefPtSph(-600,270,0, PrSph.O);
[M64] 72: DefPtSph(1,270,0, PrSph.Z);
[7B4] 73: DefPtSph(1,0,90, PrSph.YUP);
[0Z3] 74: DefI2D(0,10,10,0, IScrScl);
[K32] 75: Parameter3D;
[SC2] 76: DrawI2D(WI2D,1);
[VU2] 77: {ScrFile:=TRUE;
[HT3] 78: Assign(ScrF,'RS1.DAT');
[BF2] 79: Rewrite(ScrF);
[FU0] 80: Draw;
[T62] 81: Close(ScrF);
[FU0] 82: Draw;
```

```
[P92] 83: CloseGraphic;
[KN0] 84: END.
```

Listing 10. »rs1.pas« erlaubt das Zeichnen von Meridianen und Breitenkreisen.

```
[NC3] 1: {Programm: rs2d.pas
[FG6] 2: Funktion: Beispiel eine komplizierten
[EF2] 3: Flächenliste
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[M32] 8: PROGRAM RS2;
[2A5] 9: USES Crt, Graph, U1, U3DGEOM, U3D,
[UM4] 10: UGS, UMER, UCIRC, UCONTOUR;
[GP2] 11: PROCEDURE Draw;
[C53] 12: CONST NNPhi=5; NN=4;
[2L5] 13: VAR NPhi,N,NNMer,NNCirc: INTEGER;
[5L2] 14: R,Phi: EXTENDED;
[7W2] 15: O: Pt3D; UZ: Vt3D;
[D03] 16: IUV: IntervalUV;
[MU0] 17: BEGIN
[GP1] 18: WP:=FALSE;
[N23] 19: RS.NNRS:=NNPhi*NN;
[PY4] 20: FOR NPhi:=0 TO NNPhi-1 DO BEGIN
[V44] 21: Phi:=PI/8+NPhi/NNPhi*2*PI;
[A85] 22: DirectionVt3D(FALSE,60,Phi,0,0);
[KV3] 23: FOR N:=1 TO NN DO BEGIN
[7X4] 24: DirectionVt3D(TRUE,1,0,90, UZ);
[CN3] 25: RS.Create(NPhi+N, RSP);
[ZA1] 26: CASE N OF
[351] 27: 1: BEGIN
[DA1] 28: O.Z:=-70;
[CD4] 29: DefIUV(TRUE,0,360,-30,30,IUV);
[EX4] 30: RSP^.Init(1,0,UZ,TRUE,30,IUV);
[ZN0] 31: END;
[551] 32: 2: BEGIN
[DA1] 33: O.Z:=-70;
[7A4] 34: DefIUV(TRUE,0,360,0,70, IUV);
[AV4] 35: RSP^.Init(3,0,UZ,TRUE,7,IUV);
[ZN0] 36: END;
[751] 37: 3: BEGIN
[N61] 38: O.Z:=20;
[7D4] 39: DefIUV(TRUE,0,360,-20,20,IUV);
[AX4] 40: RSP^.Init(1,0,UZ,TRUE,20,IUV);
[ZN0] 41: END;
[951] 42: 4: BEGIN
[471] 43: O.Z:=70;
[ZYL] 44: UZ.Z:=-UZ.Z;
[Z94] 45: DefIUV(TRUE,0,360,0,30, IUV);
[EX4] 46: RSP^.Init(2,0,UZ,TRUE,14,IUV);
[ZN0] 47: END;
[ZN0] 48: END;
[ZN0] 49: END;
[ZN0] 50: END;
[5W4] 51: FOR NRS:=1 TO RS.NNRS DO BEGIN
[N13] 52: RS.Call(NRS, RSP);
[M03] 53: CASE (NRS-1) MOD 4 OF
[GC5] 54: 0: BEGIN NNMer:=30; NNCirc:=1; END;
[0A5] 55: 1: BEGIN NNMer:=0; NNCirc:=16; END;
[V55] 56: 2: BEGIN NNMer:=23; NNCirc:=1; END;
[4D5] 57: 3: BEGIN NNMer:=16; NNCirc:=1; END;
[ZN0] 58: END;
[794] 59: Mer.Init(RSP,0.2,0.2,FALSE);
[A44] 60: Mer.Draw(NNMer,48,4,6,4,6);
[5V3] 61: Circ.Init(RSP,2,8,TRUE);
[8C4] 62: Circ.Draw(NNCirc,48,4,6,4,6);
[B73] 63: Contour.Init(RSP);
[K53] 64: Contour.Draw(48,4);
[ZN0] 65: END;
[ZN0] 66: END;
[MU0] 67: BEGIN
[A86] 68: NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 69: InitializeGraphic;
[5W4] 70: IntPol:=TRUE; NumbIntPol:=4;
[Z35] 71: DefI3D(-100,-100,-100,100,100,100, WI3D);
[8N4] 72: DefPtSph(6000,270,30, PrSph.C);
[XN4] 73: DefPtSph(-600,270,30, PrSph.O);
[G54] 74: DefPtSph(1,270,30, PrSph.Z);
[7B4] 75: DefPtSph(1,0,90, PrSph.YUP);
[0Z3] 76: DefI2D(0,10,10,0, IScrScl);
[K32] 77: Parameter3D;
[SC2] 78: DrawI2D(WI2D,1);
[VU2] 79: {ScrFile:=TRUE;
[KT3] 80: Assign(ScrF,'RS2.DAT');
[BF2] 81: Rewrite(ScrF);
[FU0] 82: Draw;
[T62] 83: Close(ScrF);
[FU0] 84: Draw;
[P92] 85: CloseGraphic;
[KN0] 86: END.
```

Listing 11. »rs2.pas« zeigt die Definition einer umfangreichen Flächenliste.


```
[E13] 1: {Programm: rs3.pas
[044] 2: Funktion: Zeichnen einer
[KY3] 3: aufgeschnittenen Kugel
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[S32] 8: PROGRAM RS3;
[2A5] 9: USES Crt, Graph, U1, U3DGEOM, U3D,
[UM4] 10: UGS, UMER, UCIRC, UCONTOUR;
[GP2] 11: PROCEDURE Draw;
[YZ1] 12: CONST NNV=5;
[EV2] 13: VAR NV: INTEGER;
[YD3] 14: R, VM, V1, V2: EXTENDED;
[EF1] 15: UZ: Vt3D;
[D03] 16: IUUV: IntervalUV;
[MU0] 17: BEGIN
[GP1] 18: WP:=FALSE;
[392] 19: RS.NNRS:=NNV;
[FW0] 20: R:=80;
[SA4] 21: FOR NV:=0 TO NNV-1 DO BEGIN
[YP2] 22: V1:=-R+NV/NNV*2*R;
[3H4] 23: IF NV<NNV-1 THEN V2:=V1+R/NNV
[ZW1] 24: ELSE V2:=R;
[U85] 25: DirectionVt3D(TRUE,1,220,80, UZ);
[4N4] 26: DefIUUV(TRUE,0,360,V1,V2, IUUV);
[893] 27: RS.Create(NV+1, RSP);
[KD5] 28: RSP^.Init(1,0,3D,UZ,TRUE,R+1,IUUV);
[ZN0] 29: END;
[5W4] 30: FOR NRS:=1 TO RS.NNRS DO BEGIN
[N13] 31: RS.Call(NRS, RSP);
[V94] 32: Mer.Init(RSP,0.2,0.2,TRUE);
[GD3] 33: Mer.Draw(0,88,4,6,12,14);
[9Y3] 34: Circ.Init(RSP,2,12,TRUE);
[BU3] 35: Circ.Draw(16,64,4,6,12,14);
[B73] 36: Contour.Init(RSP);
[S43] 37: Contour.Draw(60,4);
[ZN0] 38: END;
[ZN0] 39: END;
[MU0] 40: BEGIN
[A86] 41: NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 42: InitializeGraphic;
[5W4] 43: IntPol:=TRUE; NumbIntPol:=4;
[Z35] 44: DefI3D(-100,-100,-100,100,100,100, WI3D);
[8N4] 45: DefPtSph(6000,270,30, PrSph.C);
[XN4] 46: DefPtSph(-600,270,30, PrSph.O);
[G54] 47: DefPtSph(1,270,30, PrSph.Z);
[7B4] 48: DefPtSph(1,0,90, PrSph.YUP);
[Z03] 49: DefI2D(0,10,10,0, IScrScal);
[K32] 50: Parameter3D;
[SC2] 51: DrawI2D(WI2D,1);
[VU2] 52: {ScrFile:=TRUE;
[LT3] 53: Assign(ScrF,'RS3.DAT');
[BF2] 54: Rewrite(ScrF);
[FU0] 55: Draw;
[T62] 56: Close(ScrF);
[FU0] 57: Draw;
[P92] 58: CloseGraphic;
[KN0] 59: END.
```

Listing 12. Mit »rs3.pas« zeichnen Sie eine aufgeschnittene Kugel.

```
[H13] 1: {Programm: rs4.pas
[A46] 2: Funktion: Zeichnet Figur aus Kegeln
[G64] 3: Sprache: Turbo Pascal 6.0
[G13] 4: Autor: Kurt Endl
[585] 5: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 6:
[W32] 7: PROGRAM RS4;
[2A5] 8: USES Crt, Graph, U1, U3DGEOM, U3D,
[UM4] 9: UGS, UMER, UCIRC, UCONTOUR;
[GP2] 10: PROCEDURE Draw;
[C53] 11: CONST NNPhi=5; NN=4;
[997] 12: VAR NPhi, N, NMer, NNCirc, Coll, Col2: INTEGER;
[5L2] 13: R, Phi: EXTENDED;
[7W2] 14: O: Pt3D; UZ: Vt3D;
[D03] 15: IUUV: IntervalUV;
[MU0] 16: BEGIN
[GP1] 17: WP:=FALSE;
[N23] 18: RS.NNRS:=NNPhi*NN;
[PY4] 19: FOR NPhi:=0 TO NNPhi-1 DO BEGIN
[V44] 20: Phi:=PI/8+NPhi/NNPhi*2*PI;
[X85] 21: DirectionVt3D(FALSE,85,Phi,0, O);
[KV3] 22: FOR N:=1 TO NN DO BEGIN
[G15] 23: DirectionVt3D(TRUE,1,0,-90, UZ);
[CN3] 24: RS.Create(NPhi+N, RSP);
[ZA1] 25: CASE N OF
[351] 26: 1: BEGIN
[X91] 27: O.Z:=100;
[F34] 28: DefIUUV(TRUE,0,360,0,50,IUUV);
[FX4] 29: RSP^.Init(2,0,UZ,TRUE,15,IUUV);
[ZN0] 30: END;
```

```
[551] 31: 2: BEGIN
[N71] 32: O.Z:=65;
[5A4] 33: DefIUUV(TRUE,0,360,0,60, IUUV);
[LX4] 34: RSP^.Init(2,0,UZ,TRUE,25,IUUV);
[ZN0] 35: END;
[751] 36: 3: BEGIN
[S61] 37: O.Z:=30;
[L34] 38: DefIUUV(TRUE,0,360,0,70,IUUV);
[RX4] 39: RSP^.Init(2,0,UZ,TRUE,35,IUUV);
[ZN0] 40: END;
[951] 41: 4: BEGIN
[5A1] 42: O.Z:=-50;
[5A4] 43: DefIUUV(TRUE,0,360,0,60, IUUV);
[2V4] 44: RSP^.Init(3,0,UZ,TRUE,5,IUUV);
[ZN0] 45: END;
[ZN0] 46: END;
[ZN0] 47: END;
[ZN0] 48: END;
[5W4] 49: FOR NRS:=1 TO RS.NNRS DO BEGIN
[N13] 50: RS.Call(NRS, RSP);
[M03] 51: CASE (NRS-1) MOD 4 OF
[RC5] 52: 0: BEGIN NMer:=23; NNCirc:=1; END;
[1D5] 53: 1: BEGIN NMer:=35; NNCirc:=1; END;
[WC5] 54: 2: BEGIN NMer:=42; NNCirc:=1; END;
[4D5] 55: 3: BEGIN NMer:=16; NNCirc:=1; END;
[ZN0] 56: END;
[HG4] 57: IF (NRS-1) MOD 4 < 3 THEN BEGIN
[K34] 58: Coll:=2; Col2:=10; END ELSE
[A54] 59: BEGIN Coll:=6; Col2:=6; END;
[794] 60: Mer.Init(RSP,0.2,0.2,FALSE);
[686] 61: Mer.Draw(NMer,48,Coll,Col2,Coll,Col2);
[5V3] 62: Circ.Init(RSP,2,8,TRUE);
[GP5] 63: Circ.Draw(NNCirc,48,Coll,Col2,Coll,
[SY0] 64: Col2);
[B73] 65: Contour.Init(RSP);
[UN3] 66: Contour.Draw(48,Coll);
[ZN0] 67: END;
[ZN0] 68: END;
[MU0] 69: BEGIN
[A86] 70: NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 71: InitializeGraphic;
[5W4] 72: IntPol:=TRUE; NumbIntPol:=4;
[ZA4] 73: DefI3D(-140,-120,-120, 140,120,120,
[WW0] 74: WI3D);
[9N4] 75: DefPtSph(6000,270,25, PrSph.C);
[YN4] 76: DefPtSph(-600,270,25, PrSph.O);
[TS4] 77: DefPtSph(1,270,25, PrSph.Z);
[7B4] 78: DefPtSph(1,0,90, PrSph.YUP);
[Z03] 79: DefI2D(0,10,10,0, IScrScal);
[K32] 80: Parameter3D;
[SC2] 81: DrawI2D(WI2D,1);
[VU2] 82: {ScrFile:=TRUE;
[MT3] 83: Assign(ScrF,'RS4.DAT');
[BF2] 84: Rewrite(ScrF);
[FU0] 85: Draw;
[T62] 86: Close(ScrF);
[FU0] 87: Draw;
[P92] 88: CloseGraphic;
[KN0] 89: END.
```

Listing 13. »rs4.pas« malt ein kleines Wäldchen aus Kegeln.

```
[HN3] 1: {Programm: draws.f.pas
[1S3] 2: Funktion: Anzeigen von
[UL3] 3: gespeicherten Bildern
[G64] 4: Sprache: Turbo Pascal 6.0
[G13] 5: Autor: Kurt Endl
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[HS2] 8: PROGRAM DRAWSF;
[PE4] 9: USES Crt, Graph, U1, UDRAW;
[GP2] 10: PROCEDURE Draw;
[MU0] 11: BEGIN
[F33] 12: DrawScrF('RS1.DAT');
[PU3] 13: Delay(3000); ClearDevice;
[H33] 14: DrawScrF('RS2.DAT');
[PU3] 15: Delay(3000); ClearDevice;
[L33] 16: DrawScrF('RS3.DAT');
[PU3] 17: Delay(3000); ClearDevice;
[N33] 18: DrawScrF('RS4.DAT');
[XN1] 19: Delay(3000);
[ZN0] 20: END;
[MU0] 21: BEGIN
[A86] 22: NRes:=0; {0:Detect; 1:CGA; 2:EGA; 3:VGA}
[563] 23: InitializeGraphic;
[VM3] 24: DefI2D(-10,-10,10,10, WI2D);
[Z03] 25: DefI2D(0,10,10,0, IScrScal);
[F32] 26: Parameter2D;
[FU0] 27: Draw;
[P92] 28: CloseGraphic;
[KN0] 29: END.
```

Listing 14. Gespeicherte Bilder zeigen Sie mit »drawsf.pas« an.

Ihr direkter Weg zu optimalen Programmen

Fritz Jobst
Musterprogramme, Utilities und Modulbibliotheken für raffinierte Lösungen in Turbo-Pascal und Weiterentwicklungen

Heinz Wolek
DOS-Systemprogrammierung für IBM-PCs und Kompatible

Inklusive drei Disketten!

Inklusive zwei Disketten!

Lösungen in Turbo-Pascal 4.0, 5.0, 5.5 und 6.0

Dieses neue Nachschlagewerk bietet Ihnen u. a.

- ▶ geprüfte Musterlösungen und raffinierte Utilities
- ▶ detailliertes Know-how zum Turbo-Pascal-System
- ▶ hardwarenahe Programmierung
- ▶ Betriebssystemroutinen
- ▶ Programmierkurse für optimales Software-Engineering

Aus dem Inhalt:

- ▶ **DOS-Systemroutinen**, z. B. UNIX-Move unter DOS, Druckeransteuerung, ROM-BIOS und IBM-Hardware, Datenschutz...
- ▶ **Module für eine grafische Benutzeroberfläche** zur optischen Aufwertung eigener Programme oder zur Programmierung von Systemsteuerungen à la Windows;
- ▶ **Raffinierte Utilities**, z. B. Schnittstellenutilities (u. a. Turbo-Inliner für TASM und MASM, Lesen von dBase III,

III+ und IV-Dateien), Grafikutilities, Dateiverwaltungs-Utilities;

- ▶ **Turbo-Pascal Modulbibliotheken**, u. a. String- und DOS-Routinen (z. B. Dialogverarbeitung im Grafik-Modus, Desktop-Steuerung), Mausfunktionen und Interrupts;

- ▶ **Finanzbuchhaltung**, inkl. Lagerverwaltung, Musterlösungen für Mathematik, Statistik, Betriebswirtschaft, Grafik...;

- ▶ **Erweiterte Mausroutinen** zur Auswertung der Maus-Events, zum Setzen und Speichern der wichtigsten Maus-Parameter;

- ▶ **Hardwarenahe Programme**, u. a. Expanded Memory, Disk-Optimizer, Floppy-Tuning.

Fordern Sie noch heute an:

„Musterprogramme, Utilities und Modulbibliotheken für raffinierte Lösungen in Turbo-Pascal“

Praktischer Ringbuchordner, Format DIN A4, Grundwerk ca. 600 Seiten, inkl. drei 5 1/4"-Disketten, Bestell-Nr.: 2500, Preis: DM 98,-, wahlweise zwei 3 1/2"-Disketten (gegen DM 4,- Aufpreis).

Mehr Erfolg mit DOS durch effizienten Einsatz der Systemroutinen

Dieses neue Nachschlagewerk

- ▶ erläutert Ihnen sämtliche DOS-Funktionen, auch die sonst nicht dokumentierten, für sicheres und gezieltes Programmieren;
- ▶ reduziert Ihren Entwicklungsaufwand durch effizienten Einsatz der Systemroutinen;
- ▶ zeigt Ihnen, wie Sie durch Interruptprogrammierung die MS-/PC-DOS-Funktionen verändern oder erweitern;
- ▶ liefert Ihnen praxiserprobte Programme und raffinierte Utilities auf Diskette.

Aus dem Inhalt:

- ▶ **Interruptprogrammierung**: detaillierte Dokumentationen zu BIOS-Interrupts und DOS-Funktionen und deren gezielte Anwendungen;
- ▶ **Das DOS-System im Detail**: genaue Beschreibungen der verschiedenen DOS-Versionen einschließlich DOS 5.0; ausführliche Erläuterungen

zur Funktion der Systemkomponenten (z. B. E/A-System oder Boot Record); systematischer DOS-Intensivkurs;

- ▶ **DOS-Kommandos**: alphabetisch gegliederte Befehlsbeschreibungen mit praktischen Anwendungsbeispielen;
- ▶ **Tools**: detaillierte Erläuterungen der DOS-Werkzeuge; mit kompletten Tools (z. B. Texteditor „TE“);
- ▶ **Musterlösungen und Utilities**: u. a. zur Ein-/Ausgabe, Dateiverwaltung, Bausteinprogrammierung, Daten- und Parameterübergabe.

Fordern Sie noch heute an:

„DOS-Systemprogrammierung für IBM-PCs und Kompatible“

Praktischer Ringbuchordner, Format DIN A4, Grundwerk ca. 750 Seiten, inkl. zwei 5 1/4"-Disketten, Bestell-Nr.: 5000, Preis: DM 98,-, wahlweise 3 1/2"-Diskette (gegen DM 2,- Aufpreis).

**Für Ihre
Anforderung
verwenden Sie
bitte neben-
stehende
Bestellkarte**



INTEREST-VERLAG
Fachverlag für Special
Interest Publikationen und
Anwendersoftware
Römerstraße 16
W-8901 Kissing
Tel.: 0 82 33 / 2 11-0
Fax: 0 82 33 / 2 11-2 99

Alle 2 - 3 Monate erhalten Sie
Erweiterungsausgaben zu diesen
Grundwerken mit je ca. 130 Seiten,
inkl. Diskette, zum Preis von
DM 59,- (Abbestellung jederzeit
möglich).

Prüfen Sie Ihr Nachschlagewerk in Ruhe
zu Hause. Wenn Sie nicht überzeugt sind,
schicken Sie dieses mit den Disketten
innerhalb von 10 Tagen an uns zurück.
Damit ist die Sache für Sie erledigt.

um
ammeln
nentbehrlich

Die
DMV-Sammelordner

PC-Software

vom
DMV
Versandservice

- leistungsstark
- praxisbewährt
- preisbewußt

Sonderhefte • Bücher • Sammelordner

Ausgabe		DATABOX	
DOS EXTRA DM 18,-	2 Stk. 5 1/4"-Disk DM 35,-	1 Stk. 3 1/2"-Disk DM 35,-	
☐ 356 Nr. 3	☐ 357	☐ 358	
☐ 381 Nr. 4	☐ 3811	☐ 3812	
☐ 382 Nr. 5	☐ 3813	☐ 3814	
☐ 383 Nr. 6	☐ 3815	☐ 3816	
☐ 384 Nr. 7	☐ 3817	☐ 3818	
☐ 385 Nr. 8	☐ 3819	☐ 3820	
☐ 387 Nr. 10	☐ 3823	☐ 3824	
☐ 389 Nr. 12	☐ 3827	☐ 3828	
☐ 391 Nr. 13			
☐ 392 Nr. 14	☐ 3829	☐ 3830	
☐ 393 Nr. 15	☐ 3831	☐ 3832	
☐ 394 Nr. 16	☐ 3833	☐ 3834	
☐ 395 Nr. 17	☐ 3835	☐ 3836	
☐ 396 Nr. 18	☐ 3837	☐ 3838	

WinDOS-Sonderhefte
DM 18,- mit Diskette im Heft

☐ 30200 Nr. 1 ☐ 30201 Nr. 2

DMV-ComputerWissen
DM 18,-

5 1/4"-Diskette	3 1/2"-Diskette
☐ 3511 Band 1, Basic Toolbox	☐ 3512 ☐ 3513
☐ 3514 Band 2, Textverarbeitung	☐ 3515 ☐ 3516
☐ 3517 Band 3, Tips und Tricks	☐ 3518 ☐ 3519
☐ 3520 Band 4, Datenbanken	☐ 3521 ☐ 3522

DOS toolbox
DM 28,- mit Diskette im Heft

☐ 30300 Heft Nr. 1 ☐ 30301 Heft Nr. 2

Bücher		DM
Best.-Nr.		
☐ 60500	DOS-Taschenbuch Band 1	29,80
☐ 60510	DOS-Taschenbuch Band 2	29,80
☐ 60520	DOS-Taschenbuch Band 3	29,80
☐ 60530	DOS-Taschenbuch Band 4	29,80
☐ 40101	Das große DOS-Profi-Arbeitsbuch	78,-
☐ 40102	Datenfernverarbeitung - professionell	48,-
☐ 40103	Clipper 5.0	58,-
☐ 40200	PC Kompakt - ConText 4.0	19,80
☐ 40210	PC Kompakt - Windows 3.0	19,80

DOS-Taschenbücher	5 1/4"-Diskette DM 24,-	3 1/2"-Diskette DM 24,-
BAND 1	☐ 60501	☐ 60502
BAND 3	☐ 60521	☐ 60522
BAND 4	☐ 60531	☐ 60532

Sammelordner 2 Stk. 15,80

☐ 530 DOS ☐ 531 CPC International
☐ 535 PCpur ☐ 529 WinDOS

Zwischensumme + Porto und Verpackung
(Inland 4,- DM, Ausland 6,- DM)

Gesamtbetrag: _____ DM

PC-Software

5 1/4"	3 1/2"	DM	5 1/4"	3 1/2"	DM
2726 ☐ 2727 ☐	ConText 4.0,	249,-	2730 ☐ 2731 ☐	DMV-WindowsTools SystemInfo	99,-
2728 ☐ 2729 ☐	ConText 4.0-Update	99,-	2732 ☐ 2733 ☐	DMV-WindowsTools MultiCopy	99,-
2532 ☐ 2533 ☐	ConBase	149,-	2734 ☐ 2735 ☐	DMV-WindowsTools HexEdit	99,-
2538 ☐ 2539 ☐	ConView	49,-	2736 ☐ 2737 ☐	DMV-WindowsTools TextSearch	99,-
2750 ☐ 2751 ☐	ConLex - Rechtschreiblexikon Deutsch	49,-	6031 ☐ 6032 ☐	ProfEd	99,-
2752 ☐ 2753 ☐	ConLex - Übersetzungshilfe für Einsteiger Deutsch/Englisch	99,-	227 ☐ 233 ☐	BCI DisAsm/i86, (MS-DOS)	199,-
2754 ☐ 2755 ☐	ConLex - Übersetzungshilfe für Einsteiger Englisch/Deutsch	99,-	2738 ☐ 2739 ☐	NeoGraph	99,-
2756 ☐ 2757 ☐	ConLex - Übersetzungshilfe für Sprachkenner Deutsch/Englisch	99,-	6017 ☐ 6018 ☐	3D-Draw Professional	199,-
2758 ☐ 2759 ☐	ConLex - Übersetzungshilfe für Sprachkenner Englisch/Deutsch	99,-	6019 ☐ 6020 ☐	Navigator 4.5	99,-
2748 ☐ 2749 ☐	ConText Workshop	99,-	6025 ☐ 6026 ☐	Lightning 3D, 1.5	199,-
2760 ☐ 2761 ☐	GRIPS-Desktop und -DeskDraw	99,-	6027 ☐ 6028 ☐	DOS-CAD 3.0	398,-
2534 ☐ 2535 ☐	DMV-ProfiTools f. Clipper	199,-	2700 ☐ 2701 ☐	DMV-Faktura	249,-
2536 ☐ 2537 ☐	DMV-Integrator f. Clipper	199,-	2710 ☐ 2711 ☐	DMV-Atelier	99,-
256 ☐ 257 ☐	PC-VirusDoktor	99,-	2712 ☐ 2713 ☐	DMV-Atelier Zusatz-Fonts I	99,-
2903 ☐ 2904 ☐	PC-VirusFinder	149,-	2714 ☐ 2715 ☐	DMV-Atelier Symbolbibliothek I	99,-
2905 ☐ 2906 ☐	GraphBas PC, Turbo Basic	99,-	2704 ☐ 2705 ☐	DMV-Atelier Zusatz-Fonts II	99,-
2907 ☐ 2908 ☐	GraphBas PC, Microsoft Quick Basic	99,-	2706 ☐ 2707 ☐	DMV-Atelier Symbolbibliothek II	99,-
2525 ☐ 2526 ☐	ERGO 4.0, Die Benutzeroberfläche	199,-	20600 ☐ 20601 ☐	MultiCopy	29,80
2527 ☐ 2528 ☐	ERGO-4.0-Update	99,-	20602 ☐ 20603 ☐	DataBase	29,80
238 ☐ 239 ☐	MaskEdit Plus mit einem Treiber	149,-	20604 ☐ 20605 ☐	RAM Patrouille	29,80
2613 ☐ 2614 ☐	MaskEdit Plus, Paket mit zwei Treibern	149,-	20606 ☐ 20607 ☐	Tipps	29,80
2621 ☐ 2622 ☐	MaskEdit Plus, Paket mit drei Treibern	199,-	20608 ☐ 20609 ☐	Menue	29,80
			20610 ☐ 20611 ☐	Terminplaner	29,80
			20612 ☐ 20613 ☐	Astro	29,80
			20614 ☐ 20615 ☐	TextToCom	89,-
			248 ☐ 249 ☐	Fraktal-Generator 3D, MS-DOS	149,-
			20200 ☐ 20201 ☐	DMV-PowerTools I für Power Basic	149,-
			20202 ☐ 20203 ☐	DMV-PowerTools II für Quick Basic	149,-
			20204 ☐ 20205 ☐	DMV-PowerTools III für MS Basic PDS	149,-
			00003 ☐ - ☐	Demo DMV-PowerTools für Basic	10,-
			23008 ☐ 23009 ☐	Bundesliga Total	69,-
			2766 ☐ 2767 ☐	Das Super-DTP-Paket	399,-
			2764 ☐ 2765 ☐	Timeworks	379,-

+ Porto/Verpackung

(Inland 4,- DM, Ausland 6,- DM) _____ DM = Gesamtbetrag: _____ DM

Erfolgreich programmieren

Bitte abtrennen oder ausschneiden!

Ja, senden Sie mir sofort

☐ Expl. „Musterprogramme, Utilities und Modulbibliotheken für raffinierte Lösungen in Turbo-Pascal“

Grundwerk ca. 600 Seiten, DIN A4, inkl. drei 5 1/4"-Disketten, Bestell-Nr.: 2500, Preis: DM 98,-, wahlweise zwei 3 1/2"-Disketten (gegen DM 4,- Aufpreis)

☐ Expl. „DOS-Systemprogrammierung für IBM-PCs und Kompatible“

Grundwerk ca. 750 Seiten, DIN A4, inkl. zwei 5 1/4"-Disketten, Bestell-Nr.: 5000, Preis: DM 98,-, wahlweise 3 1/2"-Diskette (gegen DM 2,- Aufpreis)

Disketten-Service:

☐ Ich wünsche die Disketten im 3 1/2"-Format (DM 2,- Aufpreis pro Diskette)

Zu jedem dieser Grundwerke erhalten Sie alle 2-3 Monate Erweiterungsausgaben mit je ca. 100 Seiten, inkl. Diskette, zum Preis von DM 59,- (Abbestellung jederzeit möglich).

Bitte abtrennen oder ausschneiden!

Ja, senden Sie mir sofort

☐ Expl. „Musterprogramme, Utilities und Modulbibliotheken für raffinierte Lösungen in Turbo-Pascal“

Grundwerk ca. 600 Seiten, DIN A4, inkl. drei 5 1/4"-Disketten, Bestell-Nr.: 2500, Preis: DM 98,-, wahlweise zwei 3 1/2"-Disketten (gegen DM 4,- Aufpreis)

☐ Expl. „DOS-Systemprogrammierung für IBM-PCs und Kompatible“

Grundwerk ca. 750 Seiten, DIN A4, inkl. zwei 5 1/4"-Disketten, Bestell-Nr.: 5000, Preis: DM 98,-, wahlweise 3 1/2"-Diskette (gegen DM 2,- Aufpreis)

Disketten-Service:

☐ Ich wünsche die Disketten im 3 1/2"-Format (DM 2,- Aufpreis pro Diskette)

Zu jedem dieser Grundwerke erhalten Sie alle 2-3 Monate Erweiterungsausgaben mit je ca. 100 Seiten, inkl. Diskette, zum Preis von DM 59,- (Abbestellung jederzeit möglich).

Meine Anschrift:

Name

Vorname

Straße, Haus-Nr.

PLZ, Ort

Unterschreiben Sie hier bitte Ihre Bestellung!

Bei Minderjährigen ist die Unterschrift eines gesetzlichen Vertreters erforderlich. Ohne Ihre Unterschrift kann die Bestellung nicht bearbeitet werden.

X

Datum

Unterschrift

Bitte unterschreiben Sie auch Ihre Sicherheitsgarantie,

mit der Sie folgendes zur Kenntnis nehmen: Sie haben das Recht, Ihr angefordertes Werk innerhalb von 10 Tagen ab Lieferung an den INTEREST-VERLAG GmbH, Römerstraße 16, W-8901 Kissing, zurückzusenden, wobei für die Fristwahrung das Datum der Absendung genügt. Sie kommen dadurch von allen Verpflichtungen aus der Bestellung frei.

X

Datum

Unterschrift

PC-Sonderhefte

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/Ort

- ☐ Den Betrag bezahle ich mit dem beigefügten Verrechnungsscheck.
☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Antwortkarte

**DMV-Verlag
DOS International
Postfach 250**

W-3440 Eschwege

Bitte
ausreichend
frankieren

PC-Software

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/Ort

- ☐ Den Betrag bezahle ich mit dem beigefügten Verrechnungsscheck.
☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Antwortkarte

**DMV-Verlag
DOS International
Postfach 250**

W-3440 Eschwege

Bitte
ausreichend
frankieren

1. Informieren

DOS-Taschenbücher und DOS-EXTRA-Sonderhefte sind eine zuverlässige Informationsquelle für jeden PC-Besitzer. Nutzen auch Sie dieses Wissen, und vertiefen Sie es mit den dazu angebotenen Programmen aus dem Disketten-Service. Den PC beherrschen mit

- DOS-EXTRA-Sonderhefte
- DOS-Taschenbüchern
- DOS-Disketten-Service

Zur Erinnerung:

Meine DOS-Sonderhefte, DOS-Taschenbücher, DOS-Disketten wurde bestellt am

2. Aufrüsten

PC-Software vom DMV-Verlag ist für jeden Kenner bereits ein Begriff und auch ein gutes Angebot für Sie, Ihr PC mit starken und anwendergerechten Applikationen auszustatten. Von der Textverarbeitung bis zum Programmiersystem, beim DMV-Verlag Service kaufen Sie günstig ein. Den PC konsequent nutzen! PC-Software aus dem DMV-Verlag Service

- PC-Anwendungen
- PC-Applikationen
- PC-Entwicklungssysteme

Zur Erinnerung:

Meine PC-Software wurde bestellt am

Nutzen Sie unser zusätzliches Angebot:

☐ Expl. „INTEL 16/32 Bit Assembler-Handbuch“

Dieses Nachschlagewerk bietet Ihnen u. a.

- systematische Programmierkurse inkl. Übungsbeispielen, z. B. zur Makroprogrammierung, Interruptprogrammierung;
- detaillierte Befehls- und Funktionsbeschreibungen der Prozessoren 80 (x) 86 und 80 (x) 87
- Bibliotheken, u. a. Stringbibliothek und Routinen zum Abbrechen der Druckerausgabe
- bewährte Musterlösungen (z. B. Programme zur Mausprogrammierung, Systemprogramme . . .); selbstverständlich auch auf Diskette

Stabiler Ringbuchordner, Format DIN A4, Grundwerk über 900 Seiten, inkl. 5 1/4"-Diskette, Bestell-Nr.: 2200, Preis: DM 98,-, wahlweise

☐ 3 1/2"-Diskette (gegen DM 2,- Aufpreis)

Alle 2 - 3 Monate erhalten Sie Erweiterungsausgaben zum Grundwerk mit je ca. 130 Seiten, inkl. Diskette, zum Preis von DM 59,- (Abbestellung jederzeit möglich).



IN 110 397

60 Pfennig,
die sich
lohlen!

Postkarte / Antwort

INTEREST-VERLAG

Fachverlag für Special Interest
Publikationen und Anwendersoftware
z. Hd. Herrn Boos

Römerstraße 16

W-8901 Kissing



Mit
detaillierten
Befehlsbeschreibungen

Kompetenz

für

PC-Personalcomputer

im

Abonnement

deutlich

günstiger

Information

über

PC-Personalcomputer

sammeln Sie mit

• DOS-Einzelheften

• DATABOX PLUS
mit mehr als 700 KByte
Software und Bonus-
programm

Vorsprung

für

PC-Personalcomputer

mit

DOS EXTRA

im

Abonnement

Abo-Order DOS International

Hiermit bestelle ich »DOS International« für mindestens

☐ 12 Ausgaben

☐ 6 Ausgaben

Preise: BRD

12 Ausgaben 82,50 DM

6 Ausgaben 42,- DM

Auslandspreise: Europa

12 Ausgaben 110,- DM

6 Ausgaben 55,- DM

Außereuropäisches Ausland

12 Ausgaben 150,- DM

6 Ausgaben 75,- DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Lieferanschrift

☐ Geschäft ☐ Privat

Vorname, Name

Firma (nur wenn Lieferadresse)

Straße, Nr./Postfach

PLZ/Ort

(Bitte genaue Anschrift angeben!)

Dieses Abonnement verlängert sich automatisch um 6 bzw. 12 Ausgaben, wenn es nicht sechs Wochen vor Ablauf schriftlich gekündigt wird.

Datum

1. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Abo-Order DOS DATABOX PLUS

Hiermit bestelle ich »DOS DATABOX PLUS« für mindestens

☐ 12 Ausgaben

☐ 6 Ausgaben

Diskettenformat:

☐ IBM 5 1/4"

☐ IBM 3 1/2"

Inlandspreise: 12 Ausgaben 300,- DM, 6 Ausgaben 150,- DM

Europa: 12 Ausgaben 320,- DM, 6 Ausgaben 160,- DM

Außereurop. Ausland: 12 Ausg. 360,- DM, 6 Ausg. 180,- DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

☐ Bequem und bargeldlos durch Bankabbuchung

Bankleitzahl

Konto-Nr./Inhaber

Geldinstitut

☐ Gegen Rechnung – zahlbar innerhalb zwei Wochen nach Erhalt. (Bitte keine Vorauszahlung leisten – Rechnung abwarten.)

Bitte unbedingt auf der Vorderseite eine zweite Unterschrift leisten!

DOS-Einzelhefte und DATABOX

Einzelheft Ausgabe 7,50 DM/Stck.	DATABOX 5 1/4"-Diskette 24,- DM/Stck.	DATABOX 3 1/2"-Diskette 24,- DM/Stck.
☐ 1/89	☐	☐
☐ 2/89	☐	☐
☐ 3/89	☐	☐
☐ 4/89	☐	☐
☐ 5/89	☐	☐
☐ 6/89	☐	☐
☐ 7/89	☐	☐
☐ 8/89	☐	☐
☐ 9/89	☐	☐
☐ 10/89	☐	☐
☐ 11/89	☐	☐
☐ 12/89	☐	☐
☐ 1/90	☐	☐
☐ 2/90	☐	☐
☐ 3/90	☐	☐
☐ 4/90	☐	☐
☐ 5/90	☐	☐
☐ 6/90	☐	☐
☐ 7/90	☐	☐
Einzelheft Ausgabe 7,50 DM/Stck.	DATABOX PLUS DATABOX + Bonus 5 1/4"-Diskette 24,- DM/Stck.	DATABOX PLUS DATABOX + Bonus 3 1/2"-Diskette 24,- DM/Stck.
☐ 8/90	☐	☐
☐ 10/90	☐	☐

Einzelheft Ausgabe 7,50 DM/Stck.	DATABOX PLUS DATABOX + Bonus 5 1/4" 2 Diskette 24,- DM/Stck.	DATABOX PLUS DATABOX + Bonus 3 1/2"-Diskette 24,- DM/Stck.
☐ 11/90	☐	☐
☐ 12/90	☐	☐
☐ 1/91	☐	☐
☐ 2/91	☐	☐
☐ 3/91	☐	☐
☐ 4/91	☐	☐
☐ 5/91	☐	☐
☐ 6/91	☐	☐
☐ 7/91	☐	☐
☐ 8/91	☐	☐
☐ 9/91	☐	☐
☐ 10/91	☐	☐
☐ 11/91	☐	☐
☐ 12/91	☐	☐
☐ 1/92	☐	☐
☐ 2/92	☐	☐
☐ 3/92	☐	☐
☐ 4/92	☐	☐
☐ 5/92	☐	☐
☐ 6/92	☐	☐

☐ Gratis-Sammelordner bei Bestellung von 6 Einzelheften

+ Porto/Verpackung

(Inland 4,- DM, Ausland 6,- DM)

DM

Gesamtbetrag

DM

Abo-Order DOS EXTRA

Hiermit bestelle ich »DOS EXTRA« für mindestens

☐ 4 Ausgaben (vierteljährliche Erscheinungsweise)

Preise: BRD

4 Ausgaben 60,- DM

Auslandspreise: Europa

4 Ausgaben 70,- DM

Außereuropäisches Ausland

4 Ausgaben 80,- DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Lieferanschrift

☐ Geschäft ☐ Privat

Vorname, Name

Firma (nur wenn Lieferadresse)

Straße, Nr./Postfach

PLZ/Ort

(Bitte genaue Anschrift angeben!)

Dieses Abonnement verlängert sich automatisch um 4 Ausgaben, wenn es nicht sechs Wochen vor Ablauf schriftlich gekündigt wird.

Datum

1. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Dauerbezug DOS EXTRA DATABOX

☐ Hiermit bestelle ich »DOS EXTRA DATABOX« im Dauerbezug mit Kündigungsmöglichkeit zum vierteljährlichen Erscheinungstermin.

Achtung: Nur im Bankabbuchungsverfahren möglich!

Preise: BRD:

je 39,- DM

Auslandspreise: Europa:

je 41,- DM

Außereurop. Ausland:

je 42,- DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Diskettenformat:

☐ IBM 5 1/4"

☐ IBM 3 1/2"

☐ Bequem und bargeldlos durch Bankabbuchung

Bankleitzahl

Konto-Nr./Inhaber

Geldinstitut

☐ Gegen Rechnung – zahlbar innerhalb zwei Wochen nach Erhalt. (Bitte keine Vorauszahlung leisten – Rechnung abwarten.)

Bitte unbedingt auf der Vorderseite eine zweite Unterschrift leisten!



Abonnement

Rechnungsanschrift

Vorname, Name

Straße, Nr./Postfach

PLZ/Ort

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, W-3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung meines Widerrufs Schreibens zur Fristwahrung ausreicht. Ich bestätige das durch meine zweite Unterschrift.

Datum 2. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte
ausreichend
frankieren

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

W-3440 Eschwege

1. Abonnieren

Jeden Monat neu, vermitteln Ihnen die DOS International und DOS EXTRA wichtige Informationen zu Ihrem Personal-Computer. Nutzen Sie dieses Wissen mit einem Abonnement zu Ihrem ganz persönlichen

- Preisvorteil
- Informationsvorsprung
- Zustellkomfort

Zur Erinnerung:

Mein persönliches Abonnement wurde bestellt am

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, W-3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung des Widerrufs Schreibens zur Fristwahrung ausreicht.



Einzelhefte

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/Ort

- ☐ Den Betrag bezahle ich mit dem beigegeführten Verrechnungsscheck.
☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte
ausreichend
frankieren

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

W-3440 Eschwege

2. Sammeln

DOS-International-Leser sind immer top-informiert. Daher sollte Ihnen kein Heft oder Sonderheft in Ihrer Sammlung fehlen. Das Eintippen von Listings und Programmen ersparen Sie sich durch unseren günstigen Disketten-Service. Immer gut beraten sind Sie mit DOS International

- Einzelheften
- Sonderheften
- Disketten-Service

Zur Erinnerung:

Meine Hefte und Disketten wurden bestellt am



Abonnement

Rechnungsanschrift

Vorname, Name

Straße, Nr./Postfach

PLZ/Ort

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, W-3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung meines Widerrufs Schreibens zur Fristwahrung ausreicht. Ich bestätige das durch meine zweite Unterschrift.

Datum 2. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte
ausreichend
frankieren

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

W-3440 Eschwege

3. Schenken

Bei jedem Anlaß ist ein Abonnement ein Geschenk, das ankommt und Ihren Wunsch nach »gutem Erfolg« Ausdruck verleiht. Sie brauchen nur im Feld »Lieferanschrift« die Adresse des Beschenkten einzutragen.

- DOS International
- DOS EXTRA

Zur Erinnerung:

Mein Geschenk-Abonnement wurde bestellt am

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, W-3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung des Widerrufs Schreibens zur Fristwahrung ausreicht.

Ein **starkes** Heft

Im **ABO**



Es **lohnt** sich!

Bitte benutzen Sie die Bestellkarte.

Widerrufsrecht:
Jeder Abonnent hat das Recht, seine Bestellung innerhalb einer Woche beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich zu widerrufen. Die rechtzeitige Absendung des Widerrufs Schreibens genügt zur Fristwahrung.

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV
Daten- und
Medienverlag

Bildmanipulation mit Quick Basic

Grafik ohne Grenzen

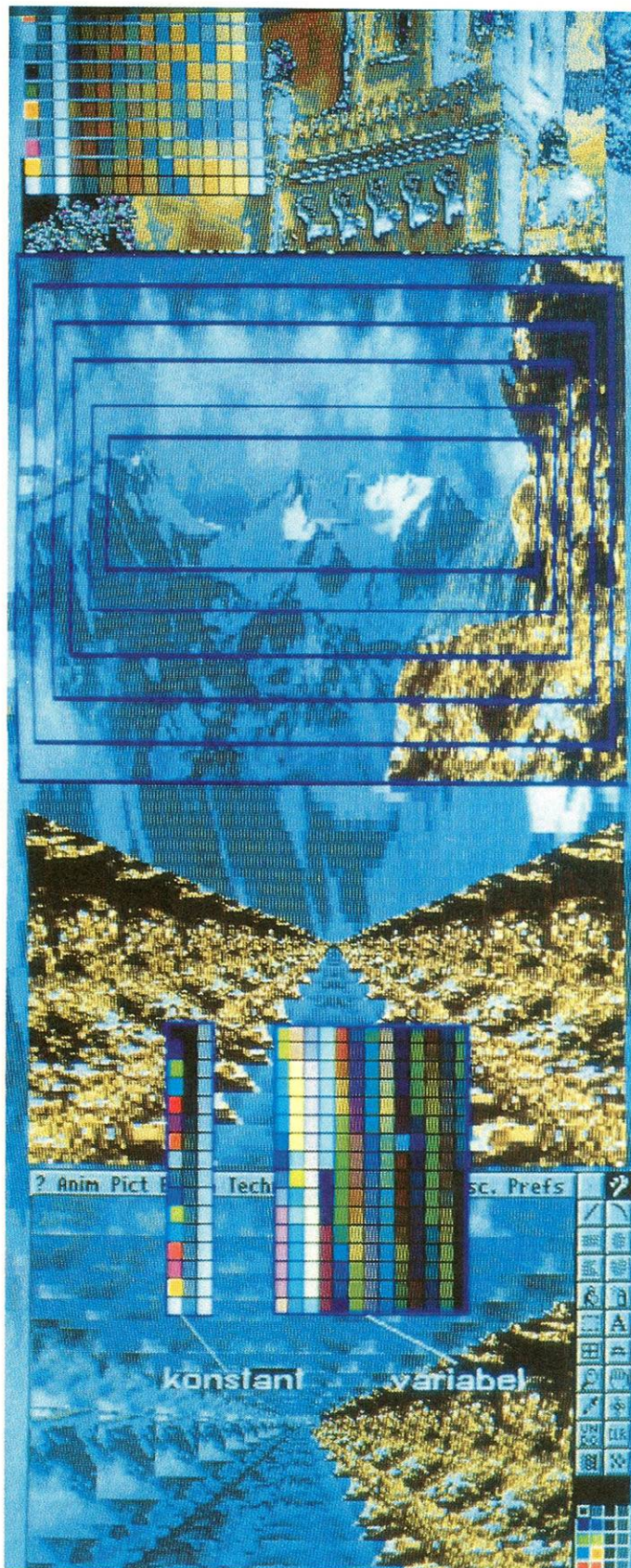
Diese Werkstatt vermittelt Tips und Tricks zur Grafikprogrammierung und Bildmanipulation, mit denen Sie Effekte erreichen, die Sie mit Basic für kaum realisierbar gehalten haben. Doch zuerst geht es um die wichtigsten Grafik-Befehle in Quick Basic, damit Sie später ohne große Probleme eigene Animationsgrafiken entwickeln können.

Die Werkstatt ist für alle Basic-Programmierer gedacht, die anspruchsvolle Grafikprogramme im VGA-Modus 320×200 mit 256 Farben entwickeln möchten. Prinzipiell bietet Basic zwar nur wenige Grafikbefehle, setzt man diese jedoch richtig ein und kennt ein paar Tricks, läßt sich damit einiges anstellen: Interaktive Animationen, blitzschnelles Scrollen von Bildbereichen, Vergrößern und Verkleinern von Bildausschnitten.

Der erste Teil der Werkstatt beschäftigt sich mit den gebotenen Grundbefehlen und vermittelt dazu detaillierte Tips, die Sie in den Basic-Manualen nicht finden. In diesem ersten Teil heißt es von Anfang an mitmachen und experimentieren – dann beherrschen Sie sämtliche Grundlagen zum Verständnis der zahlreichen Routinen, die in der Werkstatt benötigt werden. Am Ende der Werkstatt verfügen Sie über eine mächtige Grafikroutinen-Toolbox, die Sie in eigenen Programmen einsetzen können. Basic-Grundlagen werden in dieser Werkstatt nicht vermittelt. Allerdings müssen Sie kein Basic-Profi sein, um mitmachen zu können. Es reicht aus, die wesentlichen Basic-Verfahren – Entscheidungen, Schleifen und so weiter – zu kennen.

Programmiert wird ausschließlich im Screen-Modus 13 – also 320×200 Bildpunkte mit 256 Farben. Natürlich lassen sich von Basic heraus auch beliebige andere Super-VGA-Auflösungen ansprechen, dann können die Basic-Standard-Grafikbefehle (nicht einmal die Print-Anweisung!) allerdings nicht mehr eingesetzt werden, jegliche Operationen müssen über die Register der VGA-Karte erfolgen – eine knifflige Sache, da hierbei unter anderem die unterschiedlichen Chip-Sätze der VGA-Karten einzeln zu berücksichtigen sind. Der Screen Modus 13 funktioniert mit allen VGA-Karten, egal von welchem Hersteller und egal wieviel Videospeicher auf der Karte verfügbar ist. Darüber hinaus ist es eine Frage der Datenmenge: ein Bild mit $320 \times 200/256$ benötigt 64 KByte Datenträgerkapazität beziehungsweise Speicherplatz, ein Bild mit $640 \times 480/256$ mehr als 300 KByte. Damit ist auch schon klar, warum die meisten PC-Spiele im 320×200 -Modus entwickelt werden – hier wird das beste Verhältnis von Farbdarstellungsqualität, Speicherbedarf und Kompatibilität erreicht. Schließlich: je größer die Bildauflösung, umso mehr Rechenzeit wird benötigt. Beim 320×200 -Modus reicht ein 80386SX oder ein schneller 80286 zum Programmieren schneller Animationen völlig aus. Obwohl die Werkstatt auf den folgenden Seiten in Einzelbeiträge und Themen unterteilt ist, empfiehlt es sich, alle

Beiträge der Reihe nach durchzuarbeiten – die Aufteilung in einzelne Artikel soll Ihnen lediglich helfen, später schnell und gezielt nachschlagen zu können. Benötigt wird einer der Microsoft Basic-Compiler beziehungsweise -Interpreter: QBasic, Quick Basic oder Basic PDS 7.1. QBasic ist lediglich ein Basic-Interpreter, der zum Lieferumfang von MS-DOS 5.0 gehört. Obwohl sich mit QBasic einiges anstel-



len läßt, sind die Grenzen hier schnell erreicht. Trickreiche Grafikroutinen benötigen in der Regel viel Speicherplatz – und der wird bei QBasic nur eingeschränkt geboten. Immerhin: QBasic, Quick Basic und Basic PDS sind weitgehend aufwärtskompatibel, der Aufstieg zur stärkeren Version ist problemlos. Wer eine stärkere Basic-Version als QBasic benötigt, wird zwangsläufig vor der Entscheidung Quick Basic 4.5 oder Basic PDS 7.1 stehen. Quick Basic 4.5 ist für knapp 300 Mark zu haben, Basic PDS kostet etwa 800 Mark. Erster gravierender Unterschied zwischen beiden Versionen: Basic PDS 7.1 ist lediglich mit englischer Bedienungsfläche und Manualen zu haben. Die deutschen Quick-Basic-Bedienungshandbücher können es mit den Basic-PDS-Manualen allerdings nicht einmal annähernd aufnehmen. Während die deutschen Quick-Basic-Manuals äußerst oberflächliche und unzureichende Auskunft über die Funktionsvielfalt bieten, ist in den englischen Manualen alles detailliert beschrieben, gezieltes Nachschlagen ist viel besser machbar. Wer vor englischen Dokumentationen nicht zurückschreckt, sollte sich bereits aus diesem Grund für Basic PDS entscheiden. Der Basic-Compiler des PDS Basic ist zwar eine gute Portion schneller, von einem gravierenden Unterschied zu Quick Basic kann allerdings kaum die Rede sein. Weitaus interessanter sind die Toolboxen, die mit Basic PDS geliefert werden. Hier eine kurze Auflistung und Bewertung der wichtigsten Zusatztools:

- SAA-Oberfläche für Pull-down-Menüs im Textmodus (langsam und unkomfortabel, nicht empfehlenswert)
- Font-Toolbox gestattet es, Windows-Fonts unter Basic zu verwenden (sehr gut)
- ISAM-Datenbank (exzellent)

Die Entscheidung für Basic PDS sollte auf jeden Fall getroffen werden, wenn man größere Datenmengen verwalten möchte. Die ISAM-Toolbox mit ihren in Basic PDS implementierten Datenbank-Befehlen macht die Verwaltung großer Datenmengen äußerst einfach.

Achten Sie beim Einbinden beziehungsweise Abtippen von Sub-Routinen darauf, daß die Anweisung »DEFINT A-Z« nicht vergessen wird. Sie wird benötigt um alle Variablen innerhalb der Sub-Routine als Integerzahlen zu deklarieren. Es empfiehlt sich, bei Basic – insbesondere bei Grafikprogrammierung – generell zu Beginn des Hauptmoduls und sämtlicher Sub-Routinen und Funktionen alle Variablen mittels »DEFINT A-Z« als Integer festzulegen. Dann können Sie es sich unter anderem sparen, am Ende eines Variablennamens stets das Zeichen »%« als Integer-Bezeichnung zu setzen. Wird auf jegliche Globaldefinierung und Variablenkennzeichnung verzichtet, so betrachtet Basic die Variable automatisch als Single Precision. Dieser Variablentyp benötigt zum einen mehr Speicherplatz als Integer, zum anderen reduziert er die Ausführungsgeschwindigkeit drastisch. Integerzahlen können Werte im Bereich von -32768 bis 32768 annehmen – ausreichend für die meisten Vorhaben. Jegliche anderen Variablentypen sollten nur dann eingesetzt werden, wenn sie unbedingt erforderlich sind.

Im Anschluß noch eine kurze Zusammenfassung dessen, was Sie auf den folgenden Seiten im Rahmen dieser Werkstatt erwartet:

● Grafik-Befehle in Basic

Seite 42

Hier wird kurz auf die Standard-Basic-Befehle im Grafik-Bereich eingegangen. Zudem gibt es viele Tips und Tricks dazu, wie sich die Standard-Befehle effektiver einsetzen lassen. Besonders wichtig: die Aufschlüsselung von Bilddaten-Arrays und ihre Speicherung. Schnelle Animationen in Basic und umfassende Manipulationen an Bildern oder sogar ganzen Bildsequenzen sind kein Problem, wenn Sie über die richtigen Grafik-Routinen verfügen. Die VGA-Toolbox bietet Ihnen zahlreiche solcher Routinen, komplett in Basic. Die VGA-Toolbox können Sie als Library in beliebige eigene Programme einsetzen.

● Basic und die Grafikwelt

Seite 48

In diesem Beitrag werden Programme und Utilities vorgestellt, die bei der Grafik-Programmierung unter Basic helfen. Sie erfahren unter anderen, welche Grafikviewer geeignet sind und wie sie von Basic heraus angesteuert werden.

● Bunt gemischt

Seite 52

In diesem Beitrag erfahren Sie alles Wissenswerte über die Farbpalette der VGA-Karte. Themen sind hier unter anderen: Struktur der Farbpalette, Laden, Speichern, komplettes oder teilweises Manipulieren. Außerdem finden Sie hier wieder viele Tips dazu, wie Sie Paletten günstig auswählen und sich eine vielfältige individuelle Standard-Farbpalette schaffen. Als Ergebnis der Grundlagen in diesem Kapitel erhalten Sie einen Paletten-Editor in Form einer Sub-Routine, der sich bei der Grafikprogrammierung schnell als unentbehrlicher Helfer erweisen wird.

● Der Film am laufenden Band

Seite 52

In der DOS-Grafikwelt gibt es herrliche Grafiken in hoher Auflösung und Farbenanzahl – der Basic-Grafikprogrammierer steht vor dem Problem, wie er an diese per Programm herankommt. Der Auto-Grabber macht mit diesem Problem effektiv und endgültig Schluß. Sie lernen ein leistungsstarkes Basic-Utility kennen, mit dem Sie Bilder in Basic-verträgliche Formate konvertieren. Außerdem dient dieses Tool als Aufrufparameter des steuerbaren Viewer für Basic-Bildformate.

Ganz nebenbei gestattet der Auto-Grabber, sogar Deluxe-Paint-Animations-Bildsequenzen zu konvertieren und von Basic heraus abzuspielen, wobei eine interaktive Animations-Steuerung per Programm erlaubt ist.

● Zoom mit Perspektive

Seite 76

Hier geht es in Theorie und Praxis um eine leistungsstarke Zoom-Routine, mit der Sie Bildausschnitte perspektivisch zoomen sowie vergrößern und verkleinern können.

● Voraussetzungen für diese Werkstatt:

Quick Basic 4.5 oder Basic PDS 7.1, QBasic (Basic-Version von MS-DOS 5.0) ist aufgrund seiner Speicher-Einschränkungen und dem fehlenden Compiler nur sehr bedingt geeignet.

Äußerst empfehlenswert ist der Einsatz eines Grafik-Viewers/Converters, um 256-Farben-Bilder schnell und einfach in ein Basic-verträgliches Dateiformat konvertieren zu können. Eine Auswahl geeigneter Utilities finden Sie im folgenden Beitrag (Basic und der Rest der Grafikwelt).

Grafikspielereien und Operationen erfordern viel Rechenleistung. Alle Beispielpprogramme wurden auf einem 80386SX mit 20 MHz getestet – eine ausreichende Basis für den effektiven Einsatz der vorgestellten Routinen. Prinzipiell reicht natürlich auch ein schneller 80286-PC. Als Grafikkarte wird eine beliebige VGA-Karte vorausgesetzt. Hersteller und Chipsatz der Karte spielen keine Rolle. Der Programmroutinen-Geschwindigkeitstest basierte auf Einsatz einer Speedstar HiColor VGA-Karte mit 1 MByte Videospeicher.

Wer umfangreiche Animationen wünscht, sollte über eine schnelle Festplatte verfügen. Auf jeden Fall ist der Einsatz eines Cache-Programms (eventuell »smartdrv.sys«) oder einer RAM-Disk ratsam. Obwohl sich die Grafik-Werkstatt mit 640 KByte Arbeitsspeicher begnügt, sollte der ambitionierte Grafik-Programmierer über viel Zusatzspeicher verfügen. Der Zusatzspeicher macht sich insbesondere bei der Entwicklung in Basic PDS (QBX-Shell) schnell bezahlt. Schließlich ist der Interpreter-Charakter von Basic einer der größten Vorteile dieser Sprache, und es ist ärgerlich, aus Speicherplatzgründen von der MS-DOS-Ebene aus kompilieren zu müssen.

(Michael Nickles/uk)

»PUT« und »GET«

Grafikbefehle in Basic

Mit den beiden Quick-Basic-Grafikbefehlen »PUT« und »GET« können Sie weit mehr und umfangreichere Grafiken entwickeln, als die Basic-Handbücher verraten. Deshalb erfahren Sie hier alles Wissenswerte über diese beiden Befehle.

Erfreulicherweise arbeiten die Grafikbefehle »PUT« und »GET« problemlos im Screen-Modus 13 (320×200/256) und sind damit eine erste Basis für das Hantieren mit Bildern oder Bildausschnitten: »GET« legt Bildteile in einem Array ab, »PUT« bringt Array-Bildinhalte auf den Bildschirm. Bevor ein Bild in einem Array abgelegt werden kann, muß zunächst ein ausreichend großes Array vorhanden sein. Hierbei kommen für »PUT« und »GET« günstigerweise Integer-Arrays zum Einsatz. Beim Screen-Modus 13 mit 256 Farben werden für jeden Bildpunkt 8 Bit – also ein Byte – benötigt. Bei einem Integer-Array (beispielsweise »Puffer%(1 TO 16000)«) beinhaltet jedes Array-Element (Puffer%(1), Puffer%(2) und so weiter) zwei Bytes – also stets zwei Bildpunkte beim Einsatz mit »PUT« und »GET«. Die Größe eines Bild-Arrays wird so berechnet:

Groesse%=4+INT(((Bildbreite)*8+7)/8)*Bildhoehe

Möchten Sie einen kompletten 320×200-Bildschirm in einem Array unterbringen, wird nach dieser Formel ein Array benötigt, das 64004 Byte fassen kann.

DIM Puffer%(1 TO 32002)

32002 deshalb, weil beim Integer-Array ein Element zwei Bytes enthält.

Ein so großes Bild-Array zwackt natürlich 64 KByte vom Speicher ab und erfordert, daß Programme vor ihrem Start als Exe-Datei kompiliert werden. Bei der Entwicklung in der QB- oder QBX-Shell sollten Sie zunächst mit kleinen

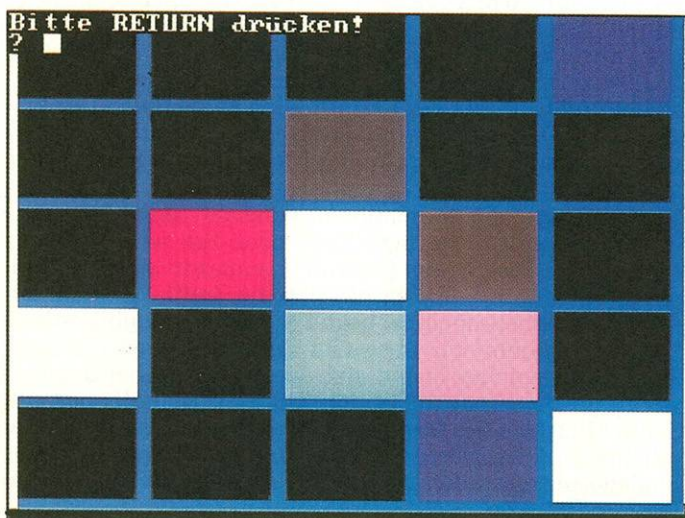


Bild 1. Das Beispielprogramm »vga.bas« in Aktion

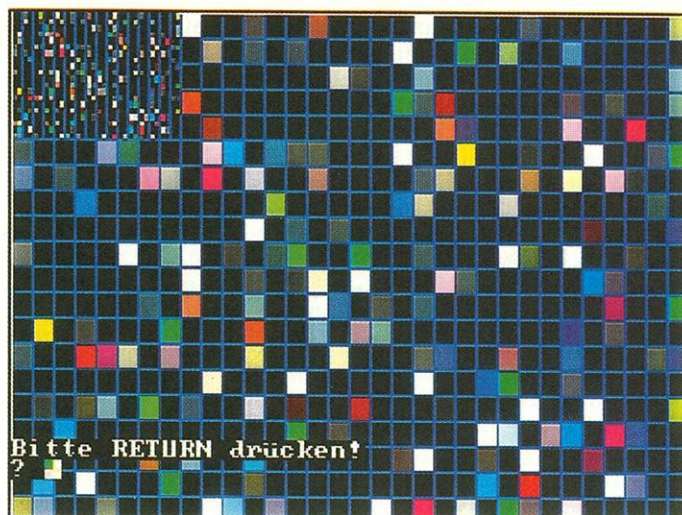


Bild 2. Sie speichern zwei Bildpunkte in einem Array-Element.

Bild-Arrays (beispielsweise für 160×100 Bildpunkte) experimentieren.

Mit der Anweisung

GET (x%,y%),x%+Breite%.1,x%+Hoehe%-1, Puffer

wird ein rechteckiger Bildausschnitt ab Bildschirmposition »x%« und »y%« mit der angegebenen Breite und Höhe (jeweils in Bildpunkten) in das Array Puffer übertragen. Nachdem ein Bildausschnitt mit »GET« eingefangen ist, kann er mit der »PUT«-Anweisung an eine beliebige Bildschirmposition kopiert werden:

PUT (x%,y%), Puffer, PSET

»x%« und »y%« stehen hier für die obere linke Ecke des rechteckigen Ausschnitts beziehungsweise die Bildschirmposition, an die er zu kopieren ist. Der Parameter »PSET« bewirkt, daß bestehende Bildpunkte mit denen des Ausschnitts überschrieben werden. An die Stelle von »PSET« lassen sich weitere Schalter setzen:

PRESET – zeichnet den Bildbereich in invertierten Farben. Der darunter liegende Bereich wird hierbei gelöscht.

AND – mischt gespeicherten Bildbereich mit dem darunter existierenden Grafikbereich.

OR – überlagert gespeicherten Bildbereich mit dem aktuellen Bild.

XOR – invertiert Punkte im Bildbereich mit Punkten im darunter liegenden Bild oder dem freien Hintergrund – interessant für Animationen, bewegte Objekte.

Was sich in einem Array befindet, kann bei Basic auch schnell und einfach als Datei gespeichert werden:

Die Anweisung

DEF SEG=VARSEG(Puffer(1))

setzt den Beginn des Speichersegments auf das erste Array-Element. Anschließend kann das Array mit der Anweisung

BSAVE Datei\$, VARPTR(Puffer(1)), Groesse%

in eine Datei gespeichert werden, die in Datei\$ angegeben wurde. »Groesse%« steht für die Anzahl der Bildpunkte, beziehungsweise die Größe des Arrays, die tatsächlich für das Bild benötigt wird.

Mit »BLOAD« lassen sich so gespeicherte Arrays problemlos wieder einlesen – natürlich in ein beliebiges Array, das entsprechend groß dimensioniert ist:

DEF SEG=VARSEG(Puffer(1))

BLOAD Datei\$, VARPTR(Puffer(1))

Nach dem Lade- oder Speichervorgang müssen Sie auf jeden Fall mit der Anweisung

DEF SEG

das Basic-Segment wiederherstellen. Wird dies vergessen, kommt Basic bei seiner Speicherzuordnung durcheinander, was zwangsläufig früher oder später zum Programmabsturz

führt. Leider verfügt keiner der Microsoft-Basic-Dialekte über eine automatische Speicher-Funktion vor Ausführen eines Programms. Einsteiger sollten stets vor Programmausführung speichern, damit keine Arbeit verlorengehen kann.

Speicherbelegung der VGA-Karte

Vor der Betrachtung des Datenformats, in dem Bilddaten in Arrays gespeichert werden, empfiehlt sich ein kurzer Blick auf die Speicherorganisation der VGA-Karte im Screen-Modus 13. Basic verfügt über zwei Grundbefehle, die den Zugriff auf die Karte unterstützen:

PSET (x,y),Farbe

setzt einen Bildpunkt in der angegebenen Farbe.

Farbe=POINT (x,y)

ermittelt die Farbe eines Bildpunkts. Damit läßt sich gut Bildpunkt für Bildpunkt im Videospeicher operieren – allerdings nicht besonders schnell. Eine Alternative bietet sich durch direkten Speicherzugriff mit »PEEK« und »POKE«.

Da mit »PEEK« und »POKE« stets nur auf ein 64 KByte großes Segment zurückgegriffen werden kann, muß der Segment-Beginn zunächst auf den Anfang des Videospeichers gesetzt werden. Dies erfolgt durch die Anweisung:

```
DEF SEG=&HA000
```

Der Videospeicher im Modus 13 beginnt stets bei der Hex-Adresse »%HA000«. Anschließend muß man nur noch wissen, wie die Bildpunktdaten in diesem Speicherbereich stehen, um gezielt darauf zugreifen zu können. Während – bedingt durch die 64-KByte-Segment-Schranke – höhere Auflösungen in 256 Farben kompliziert verwaltet werden müssen, ist die Welt für den Grafikprogrammierer bei 320 x 200 Bildpunkten noch in Ordnung: Bei dieser Auflösung werden lediglich 64 KByte – exakt 65 536 Byte – benötigt. Beginnend bei »&HA000« werden die Bildinformationen zeilenweise gespeichert. Also die ersten 320 Byte für die erste Bildschirmzeile, die folgenden 320 für die zweite und so weiter. Probieren Sie das einfach einmal aus:

```
SCREEN 13
PAINT(0,0),1 'Bildschirm blau färben
DEF SEG=&HA000
FOR i:=0 TO 63999
POKE i!,0
NEXT
DEF SEG
```

Bei dem Mini-Programm können Sie genau verfolgen, wie der Bildschirm zeilenweise gelöscht wird.

Da bekannt ist, daß jede Zeile im Modus 13 genau 320 Bildpunkte enthält, kann die Speicheradresse eines Bildpunkts problemlos aus seinen x- und y-Koordinaten berechnet werden. Beachten Sie, daß der erste Bildpunkt oben links stets den Koordinaten 0,0 entspricht und nicht 1,1.

Die Formel zur Berechnung der Speicheradresse sieht folgendermaßen aus:

```
PixelAdresse=320*Y+X
```

Das Setzen einzelner Bildpunkte erfolgt mit »POKE« unwesentlich schneller als mit »PSET«. Der Zugriff auf die VGA-Karte nach Setzen des Segments auf den Videospeicherbeginn ist allerdings nützlich, wenn komplette Bildschirme gespeichert werden sollen:

```
DEF SEG=&HA000
BSAVE Datei$,0,64000
DEF SEG
```

Diese Mini-Routine speichert den kompletten Bildschirminhalt in die angegebene Datei. Das Laden erfolgt anschließend so:

```
DEF SEG=&HA000
BLOAD Datei$,0
DEF SEG
```

Die BLOAD/BSAVE-Methode lohnt sich nicht nur für komplette Bildschirme, sondern auch dann, wenn ein Teil des

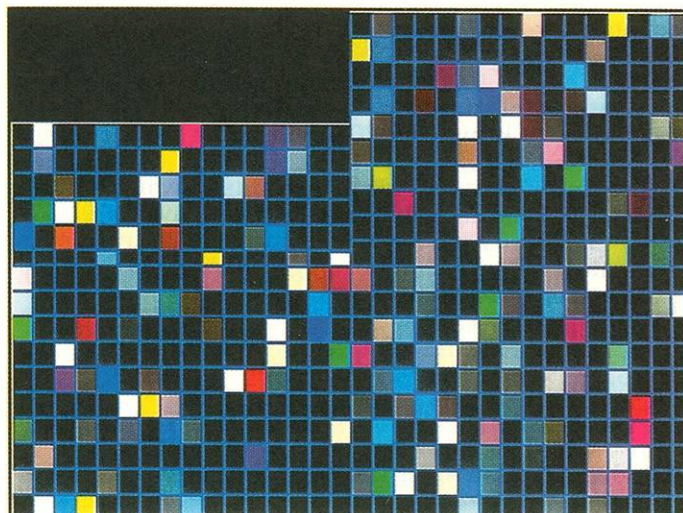


Bild 3. Eine weitere VGA-Grafikroutine aus dem Programm »vga.bas«

Bildschirms gespeichert werden soll, der folgende Bedingungen erfüllt:

- Der erste Bildpunkt des Ausschnitts (obere linke Ecke) muß sich im linken Bildrand befinden.
- Der letzte Bildpunkt des Ausschnitts (untere rechte Ecke) muß sich im rechten Bildrand befinden.

Es ist also schwierig, einen Ausschnitt aus dem Inneren des Bildschirms ohne rechts-/linksbündigen Randabschluß zu laden oder zu speichern. Wird dies gewünscht, funktioniert die Methode mit GET/PUT und anschließendem Speichern/Laden des Bild-Arrays effektiv. Wer ohne PUT/GET Ausschnitte speichern und laden möchte, kommt nicht um den Einsatz einer Binär-Datei herum, bei der die Ausschnittsdaten mit PEEK gelesen und Zeile für Zeile gespeichert werden.

Datenformat von Bild-Arrays

Die Speicherung der Daten in einem Bild-Array erfolgt nach dem gleichen Prinzip wie im VGA-Videospeicher. Auch hier werden Bilder zeilenweise gespeichert.

Intern sieht's in einem Bild-Array so aus:

```
Array(1)=Bildbreite*8
Array(2)=Bildhoehe
Array(4)-Array(n)=Bilddaten.
```

Die ersten Array-Elemente werden also benötigt, damit die »PUT«-Anweisung die Bildgröße ermitteln kann. Die Speicheradresse eines Bildpunkts in einem Bild-Array läßt sich ähnlich ermitteln wie im VGA-Videospeicher:

```
DEF SEG = VARSEG (Array(1))
PixelAdresse= 4 + Array(2) * Y + X
DEF SEG
```

Individuelle Bild-Array-Datenstrukturen

Es ist kein Problem, in einem einzigen Array mehrere Bildausschnitte abzulegen. Die Anweisung

```
GET (0,0)-(50,50),Array(1000)
```

etwa speichert einen Bildausschnitt im Array ab dem 1000sten Array-Element. Da sich leicht ausrechnen läßt, wieviel Platz ein Bild in einem Array benötigt, kann man auch leicht feststellen, ab welcher Position im Array das nächste Bild zu speichern ist. Damit ergeben sich interessante Varianten – beispielsweise die verschiedenen Bewegungsphasen einer Animation in einem Zug per BLOAD in einen Array einzulesen und dann bei der Darstellung die Einzelbilder durchzuschalten. Im Rahmen dieser Werkstatt

kommen folgende Bild-Array-Strukturen und Dateiformate zum Einsatz:

PLT-Dateien – für Paletten-Informationen (VGA-Toolbox-Routinen PaletteLaden/PaletteSpeichern)

BLD-Dateien – kompletter VGA-Bildschirm (VGA-Toolbox-Routinen LadeBild/SaveBild)

PUT ist ein einfaches gespeichertes Bild-Array im Original-Format. (VGA-Toolbox-Routinen LoadPut/SavePut)

DAT – hier wird beim Speichern eines Bild-Arrays die Position des Bildausschnitts mitgespeichert (VGA-Toolbox-Routinen LoadDat/SaveDat):

```
Array(1) = X-Position
Array(2) = Y-Position
Array(3) = unbelegt
Array(4) = Bildbreite / 8
Array(5) = Bildhöhe
Array(7)-Array(n) = Bilddaten
```

Nach dem Einlesen eines DAT-Bild-Arrays wird dieses so plaziert:

```
PUT ( Array(1),Array(2) ) , Array(4)
```

CLP – dieses Format dient dazu, mehrere Bilder in einem einzigen Array zu speichern (VGA-Toolbox-Routinen LoadClip/SaveClip). Die ersten 99 Array-Elemente sind für Indizierung reserviert. Im Array-Element Array(1) steht stets der Wert 100 – dort beginnt immer das erste Bild im CLP-Array. In Array(2) steht, ab welchem Array-Element das zweite Bild gespeichert ist, und so weiter. Möchten Sie beispielsweise das fünfte Bild einer CLP-Datei auf den Schirm bringen, dann ist folgende Anweisung erforderlich:

```
PUT (0,0), Array (Array (5))
```

Das CLP-Format verführt zu weiteren Ideen: Sie könnten in einem Bild-Array beispielsweise auch gleich die Bewegungsbahn eines Animations-Objekts ablegen.

»vga.bas« enthält viele Routinen

Das Listing beinhaltet sämtliche Routinen, um mit den beschriebenen Bild-Array-Formaten arbeiten zu können. Außerdem gibt es Routinen für die Organisation der Farbpalette und eine Routine namens Merge, die es gestattet, beim Setzen eines Bild-Arrays eine Farbe als durchsichtig zu definieren. Sie können mit diesen Routinen sämtliche Bild-Datenbestände in Ihren eigenen Programmen verwenden, die von den Utilities und Tools dieser Werkstatt angelegt werden. Nahezu alle Tools auf den folgenden Seiten (Autograber, Paletten-Editor) basieren auf geringfügig modifizierten Routinen der VGA-Toolbox. Sämtliche SUB-Routinen im Listing sind ausführlich dokumentiert. Das Hauptmodul enthält sieben Beispiele für den Einsatz der Routinen. Fortgeschrittene Basic-Programmierer können die Toolbox so wie sie ist abtippen und für eigene Zwecke verwenden – es empfiehlt sich, diese zu einer Quick-Library (QLB-Datei) zu kompilieren.

Grundlagenwissen zum Verfahren, nach dem die einzelnen Routinen arbeiten, wird im Rahmen dieser Werkstatt in den betreffenden Kapiteln vermittelt. Einsteiger sollten sich zunächst mit den Utilities der folgenden Seiten beschäftigen und dann auf die VGA-Toolbox-Routinen zurückgreifen.

(Michael Nickles/uk)

```
[EW2] 1: 'Programm: vga.bas
[XT6] 2: 'Funktion: Befehle für die VGA-Farbpalette
[MT3] 3: 'Sprache: QuickBasic 4.5
[TY3] 4: ' Autor: Michael Nickles
[1W4] 5: '(c) 1992 DMV Widuch GmbH & Co. KG
[N4E] 6: DECLARE SUB PlayPut (x%, y%, StartFrame%, EndFrame%, Array%(), Datei$, Art%, Dauer%, Speed%)

[LD8] 7: DECLARE SUB Addframe (x%, y%, breite%, hoehe%, Array%())
[735] 8: DECLARE SUB InitClip (Array%())
[ET6] 9: DECLARE SUB LoadClip (Array%(), Datei$)
[TR6] 10: DECLARE SUB LoadDat (Art%, Array%(), Datei$)

[ØV7] 11: DECLARE SUB LoadPut (x%, y%, Art%, Array%(), Datei$)
[W87] 12: DECLARE SUB Merge (x%, y%, TransFarbe%, Array%())

[VAD] 13: DECLARE SUB PlayClip (x%, y%, StartFrame%, EndFrame%, Array%(), Art%, Dauer%, Speed%)
[796] 14: DECLARE SUB SaveClip (Array%(), Datei$)
[3U9] 15: DECLARE SUB SaveDat (x%, y%, breite%, hoehe%, Array%(), Datei$)
[RW9] 16: DECLARE SUB SavePut (x%, y%, breite%, hoehe%, Array%(), Datei$)

[T2C] 17: DECLARE SUB SoftScrollDown (x%, y%, breite%, hoehe%, AnzahlZeilen%, Array%())
[M9C] 18: DECLARE SUB SoftScrollLeft (x%, y%, breite%, hoehe%, AnzahlSpalten%, Array%())
[3VC] 19: DECLARE SUB SoftScrollRight (x%, y%, breite%, hoehe%, AnzahlSpalten%, Array%())
[SZB] 20: DECLARE SUB SoftScrollUp (x%, y%, breite%, hoehe%, AnzahlZeilen%, Array%())
[UNV] 21: DECLARE SUB VgaZoom (Array() AS INTEGER, XZOOMFAKTOR AS SINGLE, YZOOMFAKTOR AS SINGLE, X AS INTEGER, Y AS INTEGER, BildNeuBreedite AS INTEGER, BildNeuHoehe AS INTEGER)

[TL1] 22: DEFINT A-Z
[W94] 23: COMMON SHARED FrameAnzahl%
[SV4] 24: COMMON SHARED ClippingStart%
[N83] 25: Start = 1: Ende = 32400
[174] 26: DIM Puffer (Start TO Ende)
[153] 27: DIM SHARED Pal(255, 2)
[7C1] 28: SCREEN 13
[VA4] 29: '-----
[GS6] 30: 'Beispiel 1: Zoomen des Bildbereichs von
[S76] 31: 'Ø,Ø - 5Ø,5Ø auf volle Bildschirmgröße.
[PØØ] 32: CLS
[CF2] 33: GOSUB TestBild
[NT3] 34: GET (Ø, Ø)-(5Ø, 5Ø), Puffer
[ZØ5] 35: VgaZoom Puffer(), 1, 1, Ø, Ø, 32Ø, 2ØØ
[4M6] 36: PRINT "Bitte RETURN drücken!": INPUT i
[VA4] 37: '-----
[RN6] 38: 'Beispiel 2: Verkleinern des Bildschirms
```

```
[W63] 39: 'auf Größe 8Ø x 5Ø.
[PØØ] 40: CLS
[CF2] 41: GOSUB TestBild
[GU3] 42: GET (Ø, Ø)-(319, 199), Puffer
[7V4] 43: VgaZoom Puffer(), 1, 1, Ø, Ø, 8Ø, 5Ø
[MM1] 44: LOCATE 22, 1
[4M6] 45: PRINT "Bitte RETURN drücken!": INPUT i
[VA4] 46: '-----
[Y36] 47: 'Beispiel 3: Zeichnet ein Testbild und
[466] 48: 'speichert einen Bildausschnitt als
[YF6] 49: 'PUT-Bilddatei "test.put", dann CLS und
[UW1] 50: 'Bild laden.
[PØØ] 51: CLS
[CF2] 52: GOSUB TestBild
[KØ6] 53: SavePut Ø,Ø,1ØØ,1ØØ,Puffer(),"test.put"
[PØØ] 54: CLS
[PR5] 55: LoadPut 3Ø, 3Ø, 1, Puffer(), "test.put"
[4M6] 56: PRINT "Bitte RETURN drücken!": INPUT i
[VA4] 57: '-----
[2U6] 58: 'Beispiel 4: Zeichnen eines Testbilds und
[Y56] 59: 'Speichern eines Bildausschnitts als
[P86] 60: 'DAT-Bilddatei "test.dat", dann CLS und
[UW1] 61: 'Bild laden.
[PØØ] 62: CLS
[CF2] 63: GOSUB TestBild
[XZ5] 64: SaveDat 3Ø,3Ø,5Ø,5Ø,Puffer(), "test.dat"
[PØØ] 65: CLS
[2R4] 66: LoadDat 1, Puffer(), "test.dat"
[4M6] 67: PRINT "Bitte RETURN drücken!": INPUT i
[VA4] 68: '-----
[ØA6] 69: 'Beispiel 5: Zeichnen eines Testbilds,
[LL5] 70: 'dann Speichern einer Filmsequenz
[BØ6] 71: 'als CLP-Array, dann Abspielen dieses
[HY1] 72: 'Clippings.
[TK2] 73: InitClip Puffer()
[PØØ] 74: CLS
[CF2] 75: GOSUB TestBild
[7N4] 76: '2Ø Einzelbilder aufzeichnen
[W62] 77: FOR i = Ø TO 2Ø
[ML4] 78: Addframe i * 2, Ø, 4Ø, 25, Puffer()
[ZSØ] 79: NEXT
[XM2] 80: 'Clip speichern
[HR4] 81: SaveClip Puffer(), "test.clp"
[GW1] 82: 'Clip laden
[SN4] 83: LoadClip Puffer(), "test.clp"
[ML2] 84: 'Clip abspielen
[155] 85: PlayClip Ø, Ø, 1, 1Ø, Puffer(), Ø, 5, 5Ø
[4M6] 86: PRINT "Bitte RETURN drücken!": INPUT i
[VA4] 87: '-----
[P36] 88: 'Beispiel 6: Soft-Scroll-Funktionen
[PØØ] 89: CLS
[CF2] 90: GOSUB TestBild
[8T5] 91: SoftScrollDown Ø,Ø,16Ø,1ØØ,1ØØ,Puffer()
[YP5] 92: SoftScrollUp Ø,1ØØ,16Ø,1ØØ,1ØØ,Puffer()
```



```
[B66] 93: SoftScrollLeft 160,0,160,100,160,Puffer()
[UN6] 94: SoftScrollRight 160,100,160,160,160,Puffer()

[4M6] 95: PRINT "Bitte RETURN drücken!": INPUT i
[VA4] 96: '-----
[2N6] 97: 'Beispiel 7: Zeichnen eines Kreises und
[2S5] 98: 'eines Vierecks. Dann wird der Kreis
[XW5] 99: 'in Array abgelegt und in das Viereck
[CT6] 100: 'gezeichnet, wobei die Hintergrundfarbe 0
[2E5] 101: 'als transparent behandelt wird.
[PN0] 102: CLS
[LZ2] 103: CIRCLE (30, 30), 30, 15
[1X3] 104: LINE (0, 100)-(100, 199), 1, BF
[TT3] 105: GET (0, 0)-(60, 60), Puffer
[PL1] 106: LOCATE 1, 1
[3R3] 107: PRINT "Bitte warten..."
[1H3] 108: Merge 0, 105, 0, Puffer()
[4M6] 109: PRINT "Bitte RETURN drücken!": INPUT i
[4M0] 110: END
[YK1] 111: TestBild:
[R44] 112: FOR Spalte = 0 TO 319 STEP 10
[XS4] 113: LINE (Spalte, 0)-(Spalte, 199), 1
[ZS0] 114: NEXT
[374] 115: FOR Zeile = 0 TO 199 STEP 10
[HF4] 116: LINE (0, Zeile)-(319, Zeile), 1
[ZS0] 117: NEXT
[H82] 118: FOR i = 1 TO 300
[C93] 119:   x = INT(319 * RND(1)) + 1
[2A3] 120:   y = INT(199 * RND(1)) + 1
[TY3] 121:   Farbe = INT(255 * RND(1)) + 1
[853] 122:   PAINT (x, y), Farbe, 1
[ZS0] 123: NEXT
[2F3] 124: LINE (0, 0)-(319, 199), 15, B
[651] 125: RETURN
[000] 126:
[1W5] 127: 'Hängt einem CLP-Array das nächste
[AG5] 128: 'Bildelement an und berechnet die
[CE7] 129: 'Array-Position für das nächste Element.
[EG6] 130: SUB Addrframe(x%,y%,breite%,hoehe%,Array())
[7G6] 131:   Groesse! = 4*INT(((breite%*(8)+7)/8)*1*hoe
    he%
[4Z8] 132:   GET(x%,y%)-(x%+breite%-1,y%+hoehe%-1), Arr
    ay(ClippingStart%)
[NX4] 133:   FrameAnzahl% = FrameAnzahl% + 1
[F26] 134:   Array(FrameAnzahl%) = ClippingStart%
[857] 135:   ClippingStart% = ClippingStart%+(Groesse!/2)
    +8
[HA1] 136: END SUB
[FL6] 137: 'Bereitet den gewählten Puffer für ein
[HD3] 138: 'neues Clipping vor.
[DL3] 139: SUB InitClip (Array())
[Z21] 140:   ERASE Array
[D83] 141:   Start = 1: Ende = 32000
[Y94] 142:   REDIM Array(Start TO Ende)
[8G2] 143:   FrameAnzahl% = 0
[733] 144:   ClippingStart% = 1000
[HA1] 145: END SUB
[9N6] 146: 'SUB-Routine zum Einlesen eines im BLD-
[W86] 147: 'Format gespeicherten Bilds im SCREEN
[E25] 148: 'Modus 13 (VGA 320x200 in 256 Farben.
[4E5] 149: 'Aufruf mit Ladebild Dateiname$
[AB3] 150: SUB LadeBild (Datei$)
[W72] 151: DEF SEG = &H0000
[D85] 152: 'Beginn des VGA-Videospeichers
[TX2] 153: Dateiname$ = Datei$
[4V3] 154: 'Dateiname der Bilddatei
[013] 155: BLOAD Dateiname$, 0
[U25] 156: 'Direktes Laden der Bilddatei in den
[ZB6] 157: 'Videospeicher der VGA-Karte. Bild baut
[Z67] 158: 'sich somit bereits während dem Einlesen
[G43] 159: 'am Bildschirm auf.
[091] 160: DEF SEG
[HA1] 161: END SUB
[K26] 162: 'SUB-Routine lädt ein Array, das mehrere
[6T5] 163: 'Bilder enthält in den angegebenen
[3A2] 164: 'Array-Puffer.
[HN4] 165: SUB LoadClip (Array(), Datei$)
[344] 166: DEF SEG = VARSEG(Array(1))
[WT4] 167: BLOAD Datei$, VARPTR(Array(1))
[0V6] 168: DEF SEG 'Stelle BASIC-Segment wieder her.
[HA1] 169: END SUB
[000] 170:
[N37] 171: 'Lädt ein DAT-Bildarray und positioniert
[CW5] 172: 'es abhängig von "Art" am Bildschirm.
[856] 173: 'Ist Art=0, so wird das Bild lediglich
[XY4] 174: 'ins angegebene Array geladen.
[YC5] 175: SUB LoadDat (Art%, Array%(), Datei$)
[344] 176: DEF SEG = VARSEG(Array(1))
[WT4] 177: BLOAD Datei$, VARPTR(Array(1))
[4P6] 178: DEF SEG 'Stelle BASIC-Segment wieder her.
[WT1] 179: x = Array(1)
[ZU1] 180: y = Array(2)
[972] 181: 'Art = Array(3)
[3F6] 182: 'hier ungenutzt - reserviert um in den
[DS6] 183: 'Clip-Bilddaten gleich mitzuspeichern, in
[1U5] 184: 'welcher Art das Bild gePUTet wird.
[703] 185: breite = Array(4) / 8
[ZF2] 186: hoehe = Array(5)
[BP2] 187: SELECT CASE Art%
```

```
[ZR1] 188: CASE IS = 1
[474] 189:   PUT (x%, y%), Array(4), PSET
[3S1] 190: CASE IS = 2
[6L4] 191:   PUT (x%, y%), Array(4), PRESET
[7S1] 192: CASE IS = 3
[Z24] 193:   PUT (x%, y%), Array(4), AND
[BS1] 194: CASE IS = 4
[RV3] 195:   PUT (x%, y%), Array(4), OR
[FS1] 196: CASE IS = 5
[864] 197:   PUT (x%, y%), Array(4), XOR
[GS1] 198: END SELECT
[HA1] 199: END SUB
[ME5] 200: 'SUB-Routine zum Laden eines mit
[B87] 201: 'SavePut-SUB-Routine gespeicherten Bildes.
[T57] 202: 'Lädt ein PUT-Bildarray und positioniert
[CW5] 203: 'es abhängig von "Art" am Bildschirm.
[856] 204: 'Ist Art=0, so wird das Bild lediglich
[XY4] 205: 'ins angegebene Array geladen.
[8K3] 206: 'Aufruf der Routine:
[UY4] 207: 'LOAD x%, y%,Art%,Array%() Datei$
[547] 208: SUB LoadPut(x%,y%,Art%,Array%(),Datei$) STA
    TIC
[3U5] 209: 'x% / y% = Position des Bildschirms
[B34] 210: 'Art% 0=nur in Array laden
[781] 211: '1 = PSET
[3T1] 212: '2 = PRESET
[311] 213: '3 = AND
[W01] 214: '4 = OR
[041] 215: '5 = XOR
[8F5] 216: 'Datei$ = Dateiname mit Extension
[344] 217: DEF SEG = VARSEG(Array(1))
[WT4] 218: BLOAD Datei$, VARPTR(Array(1))
[4P6] 219: DEF SEG 'Stelle BASIC-Segment wieder her.
[403] 220: breite = Array(1) / 8
[LF2] 221: hoehe = Array(2)
[BP2] 222: SELECT CASE Art%
[ZR1] 223: CASE IS = 1
[XW3] 224:   PUT (x%, y%), Array, PSET
[3S1] 225: CASE IS = 2
[PD4] 226:   PUT (x%, y%), Array, PRESET
[7S1] 227: CASE IS = 3
[LN3] 228:   PUT (x%, y%), Array, AND
[BS1] 229: CASE IS = 4
[9N3] 230:   PUT (x%, y%), Array, OR
[FS1] 231: CASE IS = 5
[AS3] 232:   PUT (x%, y%), Array, XOR
[GS1] 233: END SELECT
[HA1] 234: END SUB
[MV6] 235: 'Positioniert ein Bildarray am Bildschirm
[7W6] 236: 'wobei eine Farbe der Palette (TransFarbe%)
[C95] 237: 'als duchsichtig definiert ist.
[1Y5] 238: SUB Merge (x%, y%, TransFarbe%, Array%())
[403] 239: breite = Array(1) / 8
[LF2] 240: hoehe = Array(2)
[VC1] 241: Poker! = 3
[344] 242: DEF SEG = VARSEG(Array(1))
[0L3] 243: ZeileEnd = y% + hoehe% - 1
[R24] 244: SpalteEnd = x% + breite% - 1
[954] 245: FOR Zeile = y% TO ZeileEnd
[VG4] 246:   FOR Spalte = x% TO SpalteEnd
[RS2] 247:     Poker! = Poker! + 1
[ZC5] 248:   IF PEEK(Poker!) = TransFarbe% THEN
[095] 249:     POKE Poker!, POINT(Spalte, Zeile)
[801] 250:   END IF
[ZS0] 251: NEXT
[ZS0] 252: NEXT
[091] 253: DEF SEG
[XW3] 254: PUT (x%, y%), Array, PSET
[HA1] 255: END SUB
[EH6] 256: 'Weist einer Farbe oder einem Bereich der
[986] 257: 'Palette die angegebenen RGB-Werte zu.
[5E6] 258: SUB PalDef (VonFarbe, BisFarbe, r%, g%, b%)
[HS4] 259: FOR i = VonFarbe TO BisFarbe
[8K1] 260:   OUT 968, i
[7L1] 261:   OUT 969, r
[6K1] 262:   OUT 969, g
[RH1] 263:   OUT 969, b
[ZS0] 264: NEXT
[HA1] 265: END SUB
[000] 266:
[DS6] 267: 'Dunkelt eine Farbe oder einen Bereich der
[WK7] 268: 'Palette um den angegebenen Prozentwert ab.
[ZB7] 269: SUB PalDunkler (VonFarbe,BisFarbe,Prozent!)
[HS4] 270: FOR i = VonFarbe TO BisFarbe
[4K1] 271:   OUT 967, i
[XS1] 272:   r = INP(969)
[7S1] 273:   g = INP(969)
[XR1] 274:   b = INP(969)
[8K1] 275:   OUT 968, i
[993] 276:   OUT 969, r * Prozent!
[883] 277:   OUT 969, g * Prozent!
[T73] 278:   OUT 969, b * Prozent!
[ZS0] 279: NEXT
[HA1] 280: END SUB
[517] 281: 'Überträgt die aktuelle Farbpalette in das
[B26] 282: 'angegebene Array. Array muß mit den
[AH5] 283: 'Dimensionen (255,2) angelegt sein.
```

»vga.bas« ist ein Sammelsurium interessanter Grafik-Routinen.


```
[KD4] 284: SUB PaletteBackup (Array())
[592] 285: FOR i = 0 TO 255
[VR1] 286: OUT &H3C7, i
[FC3] 287: Array(i, 0) = INP(&H3C9)
[KC3] 288: Array(i, 1) = INP(&H3C9)
[NC3] 289: Array(i, 2) = INP(&H3C9)
[ZS0] 290: NEXT
[HA1] 291: END SUB
[LR6] 292: 'Routine zum Laden einer Farbpalette im
[A75] 293: 'BASIC-PLT-Format (256 Farben, RGB).
[6T5] 294: 'Die beiden ersten Aufrufparameter
[9A5] 295: 'VonFarbe% und BisFarbe% geben den
[NK6] 296: 'Bereich der zu ladenden Palette an, der
[8U5] 297: 'in die aktuell vorhandene Palette
[Y96] 298: 'übertragen werden soll. Der dritte
[245] 299: 'Parameter - ZielBereichStart% -
[6B5] 300: 'gibt an ab welcher Position der
[TT5] 301: 'aktuellen Palette der neue Bereich
[082] 302: 'geladen wird.
[C15] 303: 'Um eine komplett neue Palette
[8N4] 304: '(alle 256 Farben) zu laden, ist
[HB5] 305: 'folgender Aufruf erforderlich:
[594] 306: 'PaletteLaden 0,255,0,Datei$
[CZB] 307: SUB PaletteLaden (VonFarbe%, BisFarbe%, Zie
lBereichStart%, Array%(), Datei$)
[W72] 308: DEF SEG = &H0000
[NA2] 309: BLOAD Datei$, 0
[Z75] 310: FOR Plt = VonFarbe% TO BisFarbe%
[LR5] 311: PltPtr = ZielBereichStart + Plt * 3
[RN2] 312: r = PEEK(PltPtr)
[V13] 313: g = PEEK(PltPtr + 1)
[N13] 314: b = PEEK(PltPtr + 2)
[XF2] 315: Array(Plt, 0) = r
[8F2] 316: Array(Plt, 1) = g
[ZE2] 317: Array(Plt, 2) = b
[L21] 318: OUT 968, Plt
[7L1] 319: OUT 969, r
[6K1] 320: OUT 969, g
[RH1] 321: OUT 969, b
[ZS0] 322: NEXT
[091] 323: DEF SEG
[HA1] 324: END SUB
[000] 325:
[WK1] 326: DEFSEG A-Z
[NF6] 327: 'Überträgt die im Array gespeicherten
[Y14] 328: 'VGA-Palettentdaten in die
[ND5] 329: 'Paletten-Register der VGA-Karte.
[4R4] 330: SUB PaletteRestore (Array())
[592] 331: FOR i = 0 TO 255
[8K1] 332: OUT 968, i
[5Z2] 333: OUT 969, Array(i, 0)
[9Z2] 334: OUT 969, Array(i, 1)
[DZ2] 335: OUT 969, Array(i, 2)
[ZS0] 336: NEXT
[HA1] 337: END SUB
[000] 338:
[TL1] 339: DEFINT A-Z
[3P5] 340: 'Abspielen von Einzelbildern einer
[GF2] 341: 'CLP-Bilddatei.
[744] 342: 'x/y = Bildschirmposition
[L46] 343: 'StartFrame = Nummer des Bilds bei dem
[E94] 344: 'Abspielen beginnt.
[SN5] 345: 'EndFrame = Nummer des Bilds bei dem
[ZH5] 346: 'Abspielen von vorn beginnt.
[K66] 347: 'Array%() = Array das für die Bilddaten
[6M3] 348: 'verwendet wird.
[VV2] 349: 'Art%=0 Vorwärts
[7N2] 350: ' =1 Rückwärts
[L92] 351: ' =2 PingPong
[U66] 352: 'Dauer = Anzahl der Durchläufe in der
[SD4] 353: 'angegebenen Abspiel-Art.
[KG5] 354: 'Speed = Verzögerung, Dauer der
[4R4] 355: 'Warteschleife zwischen
[N83] 356: 'Einzelbildern.
[5PB] 357: SUB PlayClip (x%, y%, StartFrame%, EndFrame
%, Array(), Art%, Dauer%, Speed%)
[VV2] 358: 'Art%=0 Vorwärts
[G03] 359: 'Art%=1 Rückwärts
[WM2] 360: 'Art%=2 PingPong
[Z03] 361: FOR i1% = 1 TO Dauer%
[BP2] 362: SELECT CASE Art%
[VR1] 363: CASE IS = 0
[5E5] 364: FOR i2% = StartFrame% TO EndFrame%
[FK5] 365: PUT (x%, y%), Array(Array(i2)), PSET
[VE2] 366: 'Warteschleife
[8W2] 367: FOR d1 = 1 TO Speed%
[L83] 368: FOR d2 = 1 TO 100: NEXT
[ZS0] 369: NEXT
[ZS0] 370: NEXT
[ZR1] 371: CASE IS = 1
[BE6] 372: FOR i2% = EndFrame% TO StartFrame% STEP
-1
[FK5] 373: PUT (x%, y%), Array(Array(i2)), PSET
[VE2] 374: 'Warteschleife
[8W2] 375: FOR d1 = 1 TO Speed%
[L83] 376: FOR d2 = 1 TO 100: NEXT
[ZS0] 377: NEXT
[ZS0] 378: NEXT
[3S1] 379: CASE IS = 2
```

```
[5E5] 380: FOR i2% = StartFrame% TO EndFrame%
[FK5] 381: PUT (x%, y%), Array(Array(i2)), PSET
[VE2] 382: 'Warteschleife
[8W2] 383: FOR d1 = 1 TO Speed%
[L83] 384: FOR d2 = 1 TO 100: NEXT
[ZS0] 385: NEXT
[ZS0] 386: NEXT
[BE6] 387: FOR i2% = EndFrame% TO StartFrame% STEP
-1
[FK5] 388: PUT (x%, y%), Array(Array(i2)), PSET
[VE2] 389: 'Warteschleife
[8W2] 390: FOR d1 = 1 TO Speed%
[L83] 391: FOR d2 = 1 TO 100: NEXT
[ZS0] 392: NEXT
[ZS0] 393: NEXT
[GS1] 394: END SELECT
[ZS0] 395: NEXT
[HA1] 396: END SUB
[000] 397:
[TB5] 398: 'Abspielen von durchnummerierten
[5U2] 399: 'PUT-Bilddateien.
[744] 400: 'x/y = Bildschirmposition
[L46] 401: 'StartFrame = Nummer des Bilds bei dem
[E94] 402: 'Abspielen beginnt.
[SN5] 403: 'EndFrame = Nummer des Bilds bei dem
[ZH5] 404: 'Abspielen von vorn beginnt.
[K66] 405: 'Array%() = Array das für die Bilddaten
[6M3] 406: 'verwendet wird.
[GS6] 407: 'Datei$ = Dateiname der PUT-Sequenz ohne
[NH4] 408: 'Nummer und Extension!
[VV2] 409: 'Art%=0 Vorwärts
[7N2] 410: ' =1 Rückwärts
[L92] 411: ' =2 PingPong
[U66] 412: 'Dauer = Anzahl der Durchläufe in der
[SD4] 413: 'angegebenen Abspiel-Art.
[KG5] 414: 'Speed = Verzögerung, Dauer der
[4R4] 415: 'Warteschleife zwischen
[N83] 416: 'Einzelbildern.
[UTC] 417: SUB PlayPut (x%, y%, StartFrame%, EndFrame%
, Array%(), Datei$, Art%, Dauer%, Speed%)
[VV2] 418: 'Art%=0 Vorwärts
[G03] 419: 'Art%=1 Rückwärts
[WM2] 420: 'Art%=2 PingPong
[Z03] 421: FOR i1% = 1 TO Dauer%
[BP2] 422: SELECT CASE Art%
[VR1] 423: CASE IS = 0
[5E5] 424: FOR i2% = StartFrame% TO EndFrame%
[ZF8] 425: LoadPut x%, y%, 1, Array(), Datei$ + LT
RIM$(STR$(i2%)) + ".put"
[C32] 426: AktFrame = i2%
[8W2] 427: FOR d1 = 1 TO Speed%
[L83] 428: FOR d2 = 1 TO 100: NEXT
[ZS0] 429: NEXT
[791] 430: NEXT i2%
[ZR1] 431: CASE IS = 1
[BE6] 432: FOR i2% = EndFrame% TO StartFrame% STEP
-1
[ZF8] 433: LoadPut x%, y%, 1, Array(), Datei$ + LT
RIM$(STR$(i2%)) + ".put"
[C32] 434: AktFrame = i2%
[8W2] 435: FOR d1 = 1 TO Speed%
[L83] 436: FOR d2 = 1 TO 100: NEXT
[ZS0] 437: NEXT
[791] 438: NEXT i2%
[3S1] 439: CASE IS = 2
[5E5] 440: FOR i2% = StartFrame% TO EndFrame%
[ZF8] 441: LoadPut x%, y%, 1, Array(), Datei$ + LT
RIM$(STR$(i2%)) + ".put"
[C32] 442: AktFrame = i2%
[8W2] 443: FOR d1 = 1 TO Speed%
[L83] 444: FOR d2 = 1 TO 100: NEXT
[ZS0] 445: NEXT
[791] 446: NEXT i2%
[BE6] 447: FOR i2% = EndFrame% TO StartFrame% STEP
-1
[ZF8] 448: LoadPut x%, y%, 1, Array(), Datei$ + LT
RIM$(STR$(i2%)) + ".put"
[C32] 449: AktFrame = i2%
[8W2] 450: FOR d1 = 1 TO Speed%
[L83] 451: FOR d2 = 1 TO 100: NEXT
[ZS0] 452: NEXT
[791] 453: NEXT i2%
[GS1] 454: END SELECT
[ZS0] 455: NEXT
[HA1] 456: END SUB
[000] 457:
[0E3] 458: SUB SaveBild (Datei$)
[P56] 459: 'Speichert kompletten Bildschirm in
[Z56] 460: 'die als Datei$ angegebene Datei. Laden
[W72] 461: DEF SEG = &H0000
[723] 462: BSAVE Datei$, 0, 64000
[091] 463: DEF SEG
[HA1] 464: END SUB
[000] 465:
[ZV6] 466: 'Speichert das Gesamtbildarray, das durch
[G57] 467: 'zufügen von Bildteilen mit der Routine
[R44] 468: 'AddFrame entstanden ist.
[AR4] 469: SUB SaveClip (Array(), Datei$)
[SX4] 470: Groesse! = ClippingStart% - 100
```



```
[E24] 471: Groesse! = Groesse! * 2 + 100
[C14] 472: Array(FrameAnzahl% + 1) = 0
[344] 473: DEF SEG = VARSEG(Array(1))
[TE6] 474: BSAVE Datei$, VARPTR(Array(1)), Groesse!
[ØV6] 475: DEF SEG 'Stelle BASIC-Segment wieder her.
[HA1] 476: END SUB
[ØØØ] 477:
[8F8] 478: SUB SaveDat (x%, y%, breite%, hoehe%, Array
%, Datei$)
[7R1] 479: Array(1) = x
[CR1] 480: Array(2) = y
[7P1] 481: Array(3) = 0
[L56] 482: 'reserviert für Art (PSET, PUT etc)
[PV2] 483: '(hier ungenutzt!)
[NX7] 484: Groesse% = 4 + INT(((breite%) * (8) + 7
) / 8) * 1 * hoehe% + 6
[YR7] 485: GET (x%, y%)-(x% + breite% - 1, y% + ho
eh% - 1), Array(4)
[344] 486: DEF SEG = VARSEG(Array(1))
[XE6] 487: BSAVE Datei$, VARPTR(Array(1)), Groesse
%
[T47] 488: DEF SEG ' Stelle BASIC-Segment wiede
r her.
[HA1] 489: END SUB
[ØØØ] 490:
[HN9] 491: SUB SavePut (x%, y%, breite%, hoehe%, Array
%, Datei$) STATIC
[LH8] 492: 'SUB-Routine zum Speichern eines Bildaussch
nitts.
[8K3] 493: 'Aufruf der Routine:
[UB5] 494: 'SAVE x%, y%, Breite%, Hoehe%, Datei$
[NF7] 495: 'x% / y% = Obere linke Ecke des zu speicher
nden
[MX2] 496: ' Ausschnitts
[8X6] 497: 'Breite% / Hoehe% = Größe des Ausschnitts
[8F5] 498: 'Datei$ = Dateiname mit Extension
[4H7] 499: Groesse! = 4 + INT(((breite%) * (8) + 7) /
8) * 1 * hoehe%
[LF7] 500: GET (x%, y%)-(x% + breite% - 1, y% + hoehe
% - 1), Array
[344] 501: DEF SEG = VARSEG(Array(1))
[TE6] 502: BSAVE Datei$, VARPTR(Array(1)), Groesse!
[T47] 503: DEF SEG ' Stelle BASIC-Segment wieder h
er.
[HA1] 504: END SUB
[ØØØ] 505:
[A57] 506: 'Scrollt den angegebenen Bildschirmbereich
[X17] 507: '(x,y,breite,hoehe) um n-Zeilen nach unten.
[UK6] 508: 'Als Scroll-Puffer dient das angegebene
[G71] 509: 'Array.
[YRA] 510: SUB SoftScrollDown (x%, y%, breite%, hoehe%
, AnzahlZeilen%, Array%())
[654] 511: FOR i = 1 TO AnzahlZeilen%
[WR7] 512: GET (x%, y%)-(x% + breite% - 1, y% + hoehe
% - 2), Array(1)
[175] 513: LINE (x%, y%)-(x% + breite% - 1, y%), Ø
[SL4] 514: PUT (x%, y% + 1), Array(1), PSET
[ZSØ] 515: NEXT
[HA1] 516: END SUB
[ØØØ] 517:
[A57] 518: 'Scrollt den angegebenen Bildschirmbereich
[317] 519: '(x,y,breite,hoehe) um n-Zeilen nach links.
[UK6] 520: 'Als Scroll-Puffer dient das angegebene
[G71] 521: 'Array.
[TYA] 522: SUB SoftScrollLeft (x%, y%, breite%, hoehe%
, AnzahlSpalten%, Array%())
[9D4] 523: FOR i = 1 TO AnzahlSpalten%
[K48] 524: GET (x% + 1, y%)-(x% + breite% - 1, y% + h
oehe% - 1), Array(1)
[HC8] 525: LINE (x% + breite% - 1, y%)-(x% + breite%
- 1, y% + hoehe% - 1), Ø
[Y64] 526: PUT (x%, y%), Array(1), PSET
[ZSØ] 527: NEXT
[HA1] 528: END SUB
[ØØØ] 529:
[A57] 530: 'Scrollt den angegebenen Bildschirmbereich
[T67] 531: '(x,y,breite,hoehe) um n-Zeilen nach rechts
[UK6] 532: 'Als Scroll-Puffer dient das angegebene
[G71] 533: 'Array.
[8GB] 534: SUB SoftScrollRight (x%, y%, breite%, hoehe
%, AnzahlSpalten%, Array%())
[9D4] 535: FOR i = 1 TO AnzahlSpalten%
[VR7] 536: GET (x%, y%)-(x% + breite% - 2, y% + hoehe
% - 1), Array(1)
[EU4] 537: LINE (x%, y%)-(x%, y% + hoehe% - 1), Ø
[SL4] 538: PUT (x% + 1, y%), Array(1), PSET
[ZSØ] 539: NEXT
[HA1] 540: END SUB
[ØØØ] 541:
[A57] 542: 'Scrollt den angegebenen Bildschirmbereich
[ØR6] 543: '(x,y,breite,hoehe) um n-Zeilen nach oben.
[UK6] 544: 'Als Scroll-Puffer dient das angegebene
[G71] 545: 'Array.
[XMA] 546: SUB SoftScrollUp (x%, y%, breite%, hoehe%,
AnzahlZeilen%, Array%())
[654] 547: FOR i = 1 TO AnzahlZeilen%
```

```
[K48] 548: GET (x%, y% + 1)-(x% + breite% - 1, y% + h
oehe% - 1), Array(1)
[3E8] 549: LINE (x%, y% + hoehe% - 1)-(x% + breite% -
1, y% + hoehe% - 1), Ø
[Y64] 550: PUT (x%, y%), Array(1), PSET
[ZSØ] 551: NEXT
[HA1] 552: END SUB
[ØØØ] 553:
[T97] 554: 'SUB-Routine für Perspektive-Zoom. Zoomt
[F66] 555: 'ein Bildarray von seinem Mittelpunkt
[BF5] 556: 'heraus um den angegebenen Faktor.
[ZV2] 557: 'Aufrufparameter:
[R85] 558: ' Array() = Array mit den Bilddaten
[HT5] 559: ' ZoomFaktor! = Zoomfaktor in n-fach
[7R4] 560: ' (gültige Werte ab 1.0)
[EK3] 561: ' X = X-Koordinate der
[ØØ5] 562: ' Zielbildschirm-Position
[GK3] 563: ' Y = Y-Koordinate der
[ØØ5] 564: ' Zielbildschirm-Position
[BB6] 565: ' BildNeuBreite = Neue Breite des Bildes
[HN5] 566: ' BildNeuHöhe = Neue Höhe des Bildes
[S56] 567: ' (die Ausgangshöhe und Breite wird
[AB5] 568: ' automatisch vom Array-Header
[8C2] 569: ' ermittelt.)
[ZAU] 570: SUB VgaZoom (Array() AS INTEGER, XZoomFakto
r AS SINGLE, YZoomFaktor AS SINGLE, x AS IN
TEGER, y AS INTEGER, BildNeuBreite AS INTEG
ER, BildNeuHoehe AS INTEGER)
[HA4] 571: BildAltBreite = Array(1) / 8
[FV5] 572: 'BildBreite aus Array(1) ermitteln
[WP3] 573: BildAltHoehe = Array(2)
[YD5] 574: 'Bildhöhe aus Array(2) ermitteln
[NØ8] 575: AusschnittBreite = BildAltBreite / XZoomFak
tor!
[GX5] 576: 'bestimmt die Breite des Bildbereichs
[834] 577: 'der herausgezoomt wird.
[SP7] 578: AusschnittHoehe = BildAltHoehe / YZoomFakto
r!
[PG5] 579: 'bestimmt die Höhe des Bildbereichs
[U14] 580: 'der herausgezoomt wird
[AV7] 581: StartZeile = (BildAltHoehe - AusschnittHoe
he) / 2
[TP4] 582: 'bestimmt die Startzeile des
[SW4] 583: 'herauszuzoomenden Bereichs
[8M5] 584: '(gerechnet ab der Ø,Ø Koordinate des
[G32] 585: ' Bild-Arrays)
[YH8] 586: StartSpalte = (BildAltBreite - AusschnittBr
eite) / 2
[HU4] 587: 'bestimmt die Startspalte des
[SW4] 588: 'herauszuzoomenden Bereichs
[8M5] 589: '(gerechnet ab der Ø,Ø Koordinate des
[G32] 590: ' Bild-Arrays)
[344] 591: DEF SEG = VARSEG(Array(1))
[846] 592: 'Segment auf Bildarray-Beginn setzen
[8G7] 593: FaktorHoehe! = AusschnittHoehe / BildNeuHoe
he
[HK5] 594: 'Anhand dieses Faktors erfolgt die
[R56] 595: 'Abbildung der Quellbild-Zeilenzahl
[UB5] 596: 'auf die Zielbild-Zeilenzahl.
[GK5] 597: IStart = Ø: IEnde = BildNeuHoehe - 1
[GZ4] 598: 'Die folgende "Zeilen"-Schleife
[PH6] 599: 'urchläuft das "Zielbild" zeilenweise,
[AR5] 600: 'beginnend bei Zeile 1, endend bei der
[3D6] 601: 'Bildhöhe (=maximale ZeilenAnzahl des
[5X1] 602: 'Zielbilds)
[UU3] 603: FOR i = IStart TO IEnde
[EK2] 604: ZielZeile = x + i
[WZ5] 605: 'Zur ZielZeile wird die gewünschte
[774] 606: 'X-Startposition addiert
[9G6] 607: UrsprungsZeilenNR = i * FaktorHoehe! + 1
[6Ø6] 608: 'Errechnen der Ursprungszeile durch
[T76] 609: 'Multiplizieren der aktuellen Zeile
[HY2] 610: 'mit FaktorHoehe!.
[1T6] 611: Poker! = (StartZeile + UrsprungsZeilenNR)
[TC5] 612: Poker! = Poker! * BildAltBreite + 4
[546] 613: 'Position des Zeilenbeginns im Array
[CR1] 614: 'errechnen
[458] 615: FaktorBreite! = AusschnittBreite / BildNeu
Breite
[HK5] 616: 'Anhand dieses Faktors erfolgt die
[9P6] 617: 'Abbildung der Quellbild-Spaltenanzahl
[MF5] 618: 'auf die Zielbild-Spaltenanzahl.
[BR5] 619: AStart = Ø: AEnde = BildNeuBreite - 1
[H55] 620: 'Bereich für Spalten-Schleife
[ZM5] 621: PokerStart! = Poker! + StartSpalte
[US3] 622: FOR a = AStart TO AEnde
[8E6] 623: UrsprungsSpaltenNR = a * FaktorBreite!
[TKA] 624: PSET (a + y, ZielZeile), PEEK(PokerStart
! + UrsprungsSpaltenNR)
[ZSØ] 625: NEXT
[ZSØ] 626: NEXT
[S17] 627: DEF SEG 'Segment zurück auf BASIC setzen
[HA1] 628: END SUB
[ØØØ] 629:
```

»vga.bas« ist ein Sammelsurium interessanter Grafik-Rou-
tinen.

Basic und die Grafikwelt

Wozu sollten Sie eine Bildimport-Routine programmieren, wenn sich dies mit einem externen Utility wesentlich komfortabler erledigen läßt? Hier lernen Sie eine Auswahl an Programmen kennen, die dem Grafikprogrammierer die Arbeit enorm erleichtern.

Das erste Problem, auf das der Basic-Grafikprogrammierer stößt, sind die zahlreichen Grafikformate der Computerwelt: GIF, PCX, LBM, TIFF und viele mehr. Detaillierte Auskunft über den Aufbau dieser Formate geben zahlreiche Fachbücher. Zudem sind diverse Bibliotheken erhältlich, die meist in C geschrieben sind. Dies stellt jedoch kein größeres Problem dar: Quick Basic und Basic PDS gestatten das Einbinden von Routinen anderer Microsoft-Sprachen in Basic-Programme. Theoretisch ist das ziemlich einfach, der Einsteiger wird allerdings über zahlreiche Hürden stolpern und muß sich bis zum Erfolg auf einen steinigen Weg gefaßt machen. Doch es gibt Alternativen.

Das mächtigste Tor zur Welt der Profi-Grafikprogramme stellt ein zunächst scheinbar unwichtiger Basic-Befehl dar: »SHELL«.

Dieser Befehl gestattet es, von einem laufenden Basic-Programm heraus zu MS-DOS zu springen, ein anderes Programm auszuführen und dann zum Ablauf des Basic-Programms zurückzukehren. Hier ein Beispiel:

```
PRINT "Ich springe jetzt zu DOS..."
SHELL "DIR"
PRINT "Bin jetzt im Basic-Ablauf..."
END
```

Dieses Miniprogramm springt zu MS-DOS, listet über den Befehl »DIR« die Dateien des aktuellen Verzeichnisses auf und kehrt anschließend zu Basic zurück. Nach dem Shell-Befehl können Sie, im eingrenzenden Anführungszeichen, eine beliebige MS-DOS-Anweisung angeben. Ebenfalls legitim ist folgendes:

```
Kommando$="DIR /P"
SHELL Kommando$
```

Damit führen Sie, ähnlich wie bei der Batch-Programmierung, ein MS-DOS-Programm mit verschiedenen Aufrufparametern problemlos aus.

Der aktuell in Basic eingestellte Grafikmodus wird beim Sprung zu MS-DOS und bei der Rückkehr beibehalten.

```
SCREEN 13
SHELL "DIR"
```

bewirkt beispielsweise die Ausgabe des Verzeichnisses im VGA-Modus von 320 x 200 Bildpunkten bei 256 Farben. Bild 1 zeigt, was bei folgendem Aufruf passiert:

```
SCREEN 13
SHELL "DIR"
LINE (0,0)-(300,100),B 'zeichnet blauen Rahmen
```

Dieses einfache Beispiel demonstriert, daß der mit dem MS-DOS-Befehl »DIR« erzeugte Bildschirminhalt nach Rücksprung zu Basic erhalten bleibt.

Nahezu jedes Grafikprogramm enthält ein View-Utility, mit

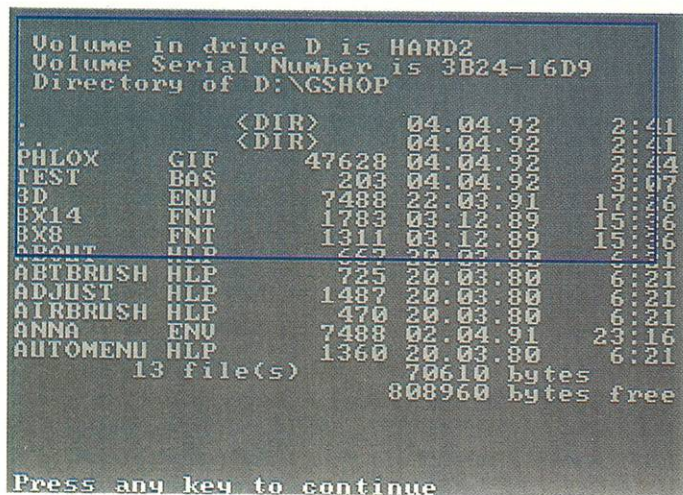


Bild 1. Mit der Shell-Anweisung können Sie zur MS-DOS-Ebene springen und ein anderes Programm zwischenstarten.

dem Sie Bilddateien ohne das eigentliche Grafikprogramm betrachten können. Damit bietet es sich an, einen solchen Viewer mit der gewünschten Bilddatei über den Shell-Befehl aufzurufen, um das dargestellte Bild anschließend mit Basic weiterzuverarbeiten oder es in einem Format zu speichern, das Sie dann direkt von Basic aus laden können. Leider ist nicht jedes View-Utility geeignet. Sie benötigen einen Viewer, der es per Aufrufparameter gestattet, nach der Bilddarstellung abzubrechen, ohne den Bildschirm zu löschen oder den Grafikmodus bei Programmende umzuschalten. Wie Sie ein auf diese Weise »importiertes« Bild in Basic speichern und laden, wird im Rahmen der Werkstatt ausführlich erläutert.

Viewer für schnellen Bildimport

Hier eine Auswahl verschiedener geeigneter Viewer und Tips, wie Sie diese einsetzen:

• »rixlate.exe« (Bild 2) gehört zum Lieferumfang des kommerziellen Grafikwerkzeugs Colorix von RIX Softworks. Rixlate kann lediglich Bilder darstellen, die im RIX-Bildfor-

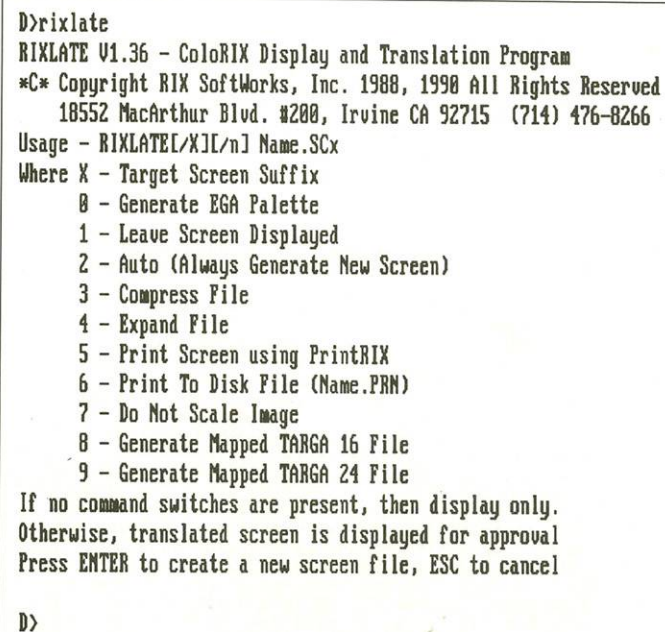


Bild 2. Rixlate ist ein Grafik-Viewer, der zum Lieferumfang des Grafikprogramms Colorix gehört.

PICEM - Picture view utility
Version 2.7 - January 24, 1992
Written by John Bridges, Copyright(C) 1987,88,89,90,91,92

To use this utility, execute the following command:

PICEM [/k /l /w:t /v:vmode[,xsize,y,size,vnum] /d:x,y /m /c] fname ...

The parameters are as follows:

/k	- do not wait for a keypress between pictures
/l	- display pictures in continous loop
/s	- scale all images to screen size (like TAB key)
/w:t	- Wait t number of 1/100ths of a second after each picture
/n	- Don't sort list of filenames
/e	- Don't clear screen on exit (leave image on screen)
/d:x,y	- set lower left hand coordinates to display picture
/d	- force centering of image (default)
/m	- multi-picture display prevents palette switch between display of multiple images
/c	- clear the screen

Press RETURN to continue, ESC to exit

Bild 3. Picem konvertiert als einer der wenigen Grafikkonverter auch ins Basic-BLD-Format.

mat SC? gespeichert sind. Das ist kaum tragisch, da im Hauptprogramm Colorix zahlreiche Im- und Exportfunktionen für die wichtigsten Grafikformate der Computerwelt integriert sind. Rixlate rufen Sie über das Shell-Kommando wie folgt auf:

```
rixlate /1 Datei.sci
```

Der Parameter »/1« steht für die Option »Leave Screen Displayed«. Sie bewirkt, daß Rixlate nach der Bilddarstellung zu MS-DOS (und von dort bei SHELL zurück zu Basic) springt und den Bildschirminhalt beibehält. RIX-Dateien mit der Endung »sci« sind Bilder mit einer Auflösung von 320 x 200 Punkten in 256 Farben. Andere Buchstaben hinter »sc« stehen für andere Auflösungen. Mit der Anweisung

```
rixlate /1 /i Datei.sc?
```

rechnet Rixlate ein beliebiges SC?-Bild bei der Darstellung in die 320 x 200-Auflösung um. Das »?« hinter »sc« beim Aufruf steht für den Buchstaben der entsprechenden Quellbilddarstellung.

● »picem.exe« (Bild 3) ist ein Viewer- und Grafikdateikonverter, der über den Sharewaremarkt vertrieben wird. Picem eignet sich insbesondere als Viewer für Bilder im GIF-Grafikformat. Der Aufruf von MS-DOS aus beziehungsweise über den Shell-Befehl in Basic erfolgt mit

```
picem /e /v:1 Datei.gif
```

»/e« bewirkt, daß der Bildschirminhalt nach der Darstellung erhalten bleibt. »/v:l« schaltet den Darstellungsmodus auf die Auflösung von 320 x 200 Bildpunkten bei 256 Farben. Prinzipiell bietet Picem einen Schalter, der dazu dienen soll, daß sich das Programm nach der Darstellung beendet. Leider funktioniert er nicht zuverlässig. Damit Picem nach der Darstellung über die MS-DOS-Shell zu Basic zurückspringt, müssen Sie eventuell die Eingabetaste drücken. Neben der Viewer-Funktion bietet »picem.exe« eine direkte Konvertierungsoption in das Basic-Format BLD. Bilddaten und Palette werden hierbei in zwei getrennten Dateien abgelegt: die Bilddatei mit der Endung »bas«, die Palettendatei mit der Endung »plt«.

● »vpic.exe« (Bild 4) wird wie Picem als Sharewareprogramm vertrieben. Dieser Viewer/Konverter eignet sich insbesondere für alle, die auf den Import vieler verschiedener Grafikformate angewiesen sind. In der aktuellen Version 4.8 kann Vpic folgende Bilddateien darstellen: CUT, PIC,

GIF (Standard 87a und 89a), LBM, IFF, MAC, PIC, PCX und SC?. Für den Einsatz in Verbindung mit SHELL rufen Sie Vpic so auf:

```
vpic /r Datei.gif
```

Der Aufrufparameter »/r« bewirkt bei Vpic die erforderliche Option, das Programm zu beenden und den Bildschirminhalt beizubehalten.

Fotografieren und Malen mit BLD-Format

Vgacad (Bild 5) ist eines der wenigen Grafikpakete, die das direkte Speichern und Laden von BLD-Dateien erlauben und auch bei Basic vorliegen. Der Grafikprogrammierer in Basic hat damit ein ideales Werkzeug für VGA-Grafiken mit einer Auflösung von 320 x 200 dpi bei 256 Farben. Mit Vgacad wählen Sie im Dateimenü BLD- oder GIF-Grafikdateien. Laden Sie ein GIF-Bild, dessen Auflösung größer ist, so rechnet es Vgacad auf Wunsch automatisch um. Anschließend können Sie das Bild im BLD-Format speichern und von Basic aus laden. Praktischerweise ist im Vgacad-Paket ein Capture-Tool namens »Vgacap« enthalten, mit dem Sie Bildschirmhalte »abfotografieren« können. Vgacap speichert Bilder, ähnlich wie Picem, beim Konvertieren: Bild- und Palettendatei bearbeiten Sie problemlos über das Bload-Kommando von Basic weiter.

Paletten auf einen Nenner gebracht

Ein Bild besteht stets aus zwei Datengruppen: den Bilddaten, die bestimmen, welche Bildpunkte in welcher Farbe gesetzt sind, und den Palettendaten, die festlegen, welche der 256 000 VGA-Farben in der 256 Farben großen Farbpalette bereitgestellt werden. Es empfiehlt sich bei der Entwicklung eines Grafikprogramms, alle Bilder, die Sie verwenden möchten, so zu arrangieren, daß sie über die gleiche Farbpalette verfügen. Dann müssen Sie die erforderliche Farbpalette zu Beginn des Programmablaufs nur ein einziges Mal laden. Später laden Sie nur noch die Bilddaten ohne die Paletteninformation. Neben einem kleinen Geschwindigkeitsgewinn durch geringfügig kürzere Ladezeit von der Festplatte ergeben sich noch weitere entscheidende Vorteile, wenn Sie eine gemeinsame Palette verwenden: Sie können ohne Schwierigkeiten verschieden große Bilder gleichzeitig in unterschiedlichen Fenstern darstellen, da

(C) Copyright 1990, 1991 Bob Montgomery, All Rights Reserved
Views GIF, MAC, ZSoft PCX, Pictor/PC Paint PIC, Dr. Halo CUT, Targa TGA, DP11 LBM and IFF, MicroSoft BMP, and ColorIX/EGA Paint SCx files in EGA/VGA.
Defaults: Current directory for path, GIF for file ext and SLD for slide file.
vpic [path][filename][.ext] [/a]/[b]/[c]/[e]/[mN]/[r]/[w]/[xN]/[yN]/[z]/[?]
where a => Autocode, b => beep, c => Show config, e => EGA, r => Return
w => Wait, x => left margin, y => top margin, z => No filenames, ? => Help
vpic /@[path][filename][.ext] = do a slide show from commands in filename.
vpic [path][filespec] gives menu of all viewable files in path meeting filespec.
Menu commands:
Up, down, right, left arrow keys move cursor, ESCape returns to DOS.
Spacebar marks/unmarks files for viewing.
Enter views marked files in order marked, or single file if none marked.
Alphanumeric key jumps to next file starting with key letter or number.
F1 = Help, F2 = Show marked files as slides, F3 = File info.
F4 = EGA/VGA toggle, F5/F6 Decrease/increase resolution
F7 Lock/Auto/Normal Video mode. F8 = Wait between images toggle
F9 = Get new path for files. F10 = Show current path.
Alt_X = Delete the file at the cursor (with prompt).
For commands while viewing, press / or ? while picture is on the screen.
See vpic.doc for details.
Configuring VPIC: 'config' or 'cvpic configfile[.ext]' where default ext is CFG
To identify VGA chip enter 'whichvga'; see config.doc for details.

Bild 4. Vpic ist ein seit langem bewährter Standard-Viewer für alle verbreiteten Grafik-Dateiformate



Bild 5. Das Grafikprogramm Vgacad verarbeitet Basic-BLD/PLT-Formate und bietet ein passendes Capture-Tool.

alle über ein und dieselbe Farbpalette verfügen. Liegen bereits mehrere Bilder mit unterschiedlichen Farbpaletten vor, sollten Sie die Bilder so umrechnen, daß eine optimale gemeinsame Palette entsteht – ein kniffliges Unterfangen, das sich nicht mit jedem beliebigen Grafikprogramm befriedigend beziehungsweise komfortabel erledigen läßt. Abhilfe schaffen sogenannte Farbpaletten-Optimierungstools, die aus mehreren verschiedenen Paletten eine optimierte gemeinsame Palette errechnen. Das Optimierungs-Utility arbeitet folgendermaßen: Es untersucht alle angegebenen Paletten und die zugehörigen Bilder darauf, welche Farben sehr häufig, selten oder gar nicht auftreten. Die am häufigsten auftretenden und damit wichtigsten Farben werden dann in die gemeinsame Palette integriert. Dabei geht bei allen Bildern meist etwas Farbe verloren, oftmals erreichen Sie jedoch mit der Optimierung ausreichend gute Ergebnisse. Sehr gute und vor allem schnelle Resultate erzielen Sie beispielsweise mit dem Shareware-Utility »GDS« (Bild 6). Es ist prinzipiell zwar nur ein GIF-Bilder-Katalogisierungstool, doch generiert es vor der Darstellung verkleinerter Miniaturbilder (Bild 7) eine gemeinsame Palette, die Sie mit einem Capture-Programm einfangen können.

Hier noch einige allgemeine Tipps zum Einsatz der Viewer: Die Tools können noch weit mehr, als hier kurz zusammengefasst wurde. So verfügen sie zum Beispiel über Funktionen, mit denen Sie bei der Darstellung automatisch skalieren oder eine Quellbildauflösung automatisch in eine andere Auflösung umrechnen können. So transportieren Sie

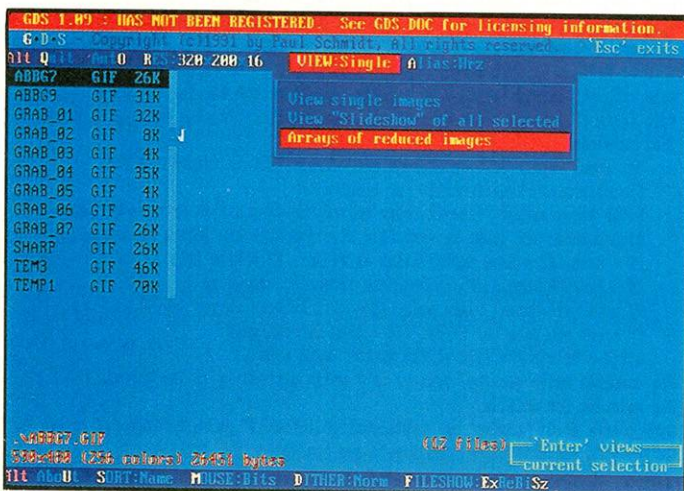


Bild 6. GDS ist zwar für die Bildkatalogisierung gedacht, lässt sich jedoch auch als Paletten-Optimierer einsetzen.

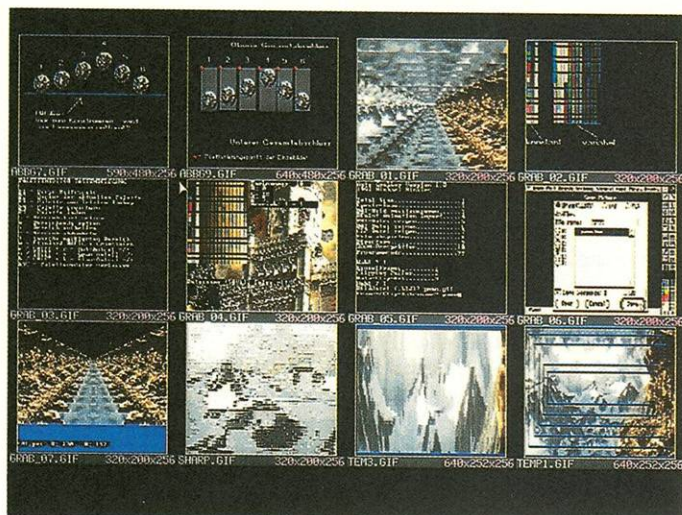


Bild 7. Verkleinerte Darstellung mehrerer Bilder gleichzeitig mit optimierter gemeinsamer Farbpalette

auch problemlos Super-VGA-Bilder mit einer Auflösung von bis zu 1024×768 bei 256 Farben in das 320×200 -Format. Die View-Utilities benötigen unterschiedlich viel Speicherplatz. Bei einem großen Basic-Programm, das Sie aus der Entwicklungsumgebung heraus starten, reicht die Speicherkapazität unter Umständen nicht, um den Viewer zu starten. In diesem Fall kompilieren Sie in der Regel das Basic-Programm und starten die Exe-Programmdatei von der MS-DOS-Oberfläche aus. Außerdem kann der Shell-Einsatz zu Schwierigkeiten führen, wenn sich im Speicher residente Programme eingenistet haben. Dazu zählen auch MS-DOS-Shells wie Norton Commander.

In diesem Beitrag wurden lediglich elementare Grafik-Utilities vorgestellt. Es gibt selbstverständlich noch viele weitere. Wenn Sie ein umfangreiches Grafikprogrammierungsprojekt planen, sollten Sie erst Ausschau nach geeigneten Tools halten und dann programmieren. So sparen Sie unter Umständen viel Zeit und Eigenaufwand.

(Michael Nickles/uk)

Info: Vpic, Picem, GDS: CDV PD und Shareware, 7500 Karlsruhe 1
Vgacad: Programm und deutsches Handbuch, Systhema Verlag
Colorix: DMV-Verlag, Eschwege

Fragen zum Inhalt richten Sie bitte an die

**Redaktion
DOS International
Gruberstraße 46c
8011 Poing**

Anfragen zum Verlags-
programm richten Sie bitte
an den

**DMV-Verlag
DOS International
Postfach 250
3440 Eschwege**

CONBASE

Die universelle Datenbank



Damit Sie Ihre Daten in den Griff bekommen

- SAA-Oberfläche
- Selektieren über jedes Feld
- bis zu 10 Indexdateien
- dBase-kompatibel

- komfortabler Maskeneditor
- frei definierbare Struktur
- 128 Felder pro Datensatz
- Listen- und Etikettendruck

Was immer Sie an Daten haben und immer wieder brauchen, mit ConBase halten Sie Ihre Datenbestände jederzeit auf Abruf bereit. Wenn Sie zum Beispiel eine der folgenden Aufgaben erledigen müssen, ist ConBase das richtige Programm für Sie:

- eine Kundenkartei anlegen
- Adressen verwalten
- die Vereinsverwaltung in Ordnung bringen
- die Mitglieder überblicken
- Briefmarken katalogisieren
- den Literaturbestand aufnehmen
- Schallplatten ordnen
- das Familienstammbuch führen
- Hochzeitsgeschenke zuordnen
- Geschäftspartner einstufen
- Schulnoten überblicken
- Fußballergebnisse festhalten
- einen Gunter-Gabriel-Fan-Club gründen
- Zeitungsartikel filtern
- den Lagerbestand aufnehmen
- ein Klassentreffen arrangieren
- den Weinkel-ler aufräumen
- Videofilme gruppieren
- Erbstücke verteilen
- Schmuck katalogisieren
- eine Bibliothek aufmachen
- oder eine

Telefonliste führen. Ihrer Phantasie sind hier keine Grenzen gesetzt! Die Datenbestände lassen sich problemlos an Ihre Textverarbeitung übergeben, zum Beispiel um mit ConText 4.0 Serienbriefe zu schreiben. Schnittstellen gibt es zu ConText 4.0, Wordstar, MS-Word und WordPerfect.

Systemanforderung: PC, 512 KByte RAM, DOS ab 3.0, Festplatte

ConBase

Die Datenbank mit dem sensationellen Preis

149,-* DM

Der ideale Partner:

ConText 4.0

Die sympathische Textverarbeitung

249,-* DM

*

Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,- bzw. für das Ausland DM 6,- Porto und Verpackung.

DMV-Verlag · Postfach 250 · W-3440 Eschwege

DMV
Daten- und
Medienverlag

VGA-Farbpalette im Detail

Bunt gemischt

Grundsätzliche Überlegungen zum Einsatz der Farbpalette werden bei der Grafikprogrammierung oft vergessen oder erfolgen zu spät. Dann ist es ärgerlich, wenn Sie alle eingebundenen Bilder nachträglich nochmals bearbeiten müssen. Abhilfe schafft von Anfang an eine Paletten-Planung und ein geeigneter Farben-Editor.

Die VGA-Karte verfügt über 262 144 Farben, von denen jeweils 256 gleichzeitig darstellbar sind. Unter »Palette« ist eine Tabelle im Programm zu verstehen, die festlegt, welche RGB-Werte den 256 möglichen Farben zugeordnet sind. Sie können sich dies wie einen Malkasten mit 256 Farbtöpfchen vorstellen, in denen sich beliebige Farben zurechtmischen lassen. Mischen heißt beim Zusammenstellen der VGA-Farbpalette Zuordnen von RGB-Werten (Rot-, Grün- und Blau-Anteile einer Farbe). Der Wertebereich von R, G und B geht jeweils von 0 bis 63. Damit erklärt sich auch schon, warum die VGA-Karte theoretisch exakt 262 144 Farben darstellen kann: jeweils 64 Werte für R, G und B erlauben insgesamt $64 \times 64 \times 64 = 262\,144$ Farbkombinationen. Grundlagen des RGB-Farbmodells und der additiven Farbmischung werden in diesem Workshop nicht vermittelt. Sie werden allerdings auch nur dann benötigt, wenn Sie detailliert wissen möchten, wie Farbtöne durch Angabe von Rot-, Grün- und Blauanteil entstehen. 256 gleichzeitig darstellbare Farben klingt zwar nach Vielfalt, in der Praxis treten jedoch schnell Probleme auf, wenn Sie mit fotorealistischen Bildern arbeiten. Bilder – egal ob in Gesamtbildschirmgröße oder im Ausschnitt – können nur dann gleichzeitig und gemeinsam dargestellt werden, wenn sie über die gleiche Farbpalette verfügen. Tips und Tricks zu diesem Problem finden Sie im späteren Teil dieses Beitrags.

Zunächst gibt es ein wenig Theorie darüber, wie man die Palette unter Basic überhaupt in den Griff bekommt. Die miserabelste Methode, per Basic an der Palette herumzumanipulieren, ist die Basic-Anweisung »PALETTE«. Sie kommt im Rahmen dieses Workshops jedoch nicht zum Einsatz, da sie es nicht gestattet, direkt Änderungen durch Angabe eines R-, G-, oder B-Wertes vorzunehmen, sondern vielmehr nach folgendem Schema arbeitet: $\text{Farbwert} = 65\,536 \times B + 256 \times G + R$. Die Umrechnung ist zwar einfach, kostet allerdings Zeit – zu viel Zeit für den, der schnelle Grafikroutinen wünscht. Zudem unterscheidet sich diese Art der Farbwert-Tabelle von der internen Tabelle der VGA-Karte, die mit getrennten R-, G- und B-Werten arbeitet.

»INP« und »OUT« statt »PALETTE«

Es empfiehlt sich, Palettenmanipulationen durch direkten Zugriff auf die Hardware der VGA-Karte durchzuführen. Über die Befehle »INP« und »OUT« gestattet Basic, einem Hardware-I/O-Port direkt einen Wert zuzuweisen – Sie müssen nur wissen, welche Ports der VGA-Karte für welche Funktionen zuständig sind. Basic-Einsteiger brauchen sich von den folgenden Anweisungen nicht verwirren zu lassen, wenn Grundlagenwissen über die Hardware der VGA-Karte fehlt – die Beispiele können so, wie sie sind, direkt in eigene

Programme übernommen werden. Der Out-Befehl hat folgende Grundsyntax:

OUT Port, Wert

Mit den Befehlen »INP« und »OUT« gestattet Basic den direkten Zugriff auf Funktionen der VGA-Karte. »Port« steht für eine Adresse (in der Regel im Bereich oberhalb der 640-KByte-Grenze), auf die sowohl von einem Programm als auch von einer Hardware-Karte zugegriffen wird. Der »Port« ist also eine Kommunikationsschnittstelle zwischen Hardware (Steckkarten im Rechner) und Programmen.

Wollen Sie eine Farbe der VGA-Palette in ihren RGB-Werten verändern, so sind die beiden Port-Adressen »&H3C8« und »&H3C9« die richtige Anlaufstelle. In die Adresse »&H3C8« wird zunächst eingetragen, welche Farbe zu verändern ist (0 bis 255), anschließend schreiben Sie in die Adresse »&H3C9« nacheinander die gewünschten RGB-Werte für die Farbe. Beispiel: die Farbe Weiß entsteht beim Setzen der höchstmöglichen R-, G- und B-Werte (bei VGA-Karte 63). Die folgenden Basic-Zeilen weisen der Farbe 40 die Farbe Weiß zu:

```
OUT &H3C8,40
OUT &H3C9,63
OUT &H3C9,63
OUT &H3C9,63
```

Möchten Sie wissen, aus welchen RGB-Werten eine Farbe der Palette besteht, so ist dafür die Port-Adresse »&H3C7« zuständig. Die folgenden BASIC-Zeilen ermitteln, welche RGB-Werte die Farbe 220 der Palette enthält:

```
OUT &H3C7,220
R=INP (&H3C9)
G=INP (&H3C9)
B=INP (&H3C9)
PRINT R
PRINT G
PRINT B
```

Das Einlesen und Auslesen der RGB-Werte erfolgt, wie aus beiden Beispielen ersichtlich, stets über die Port-Adresse »&H3C9« in der Reihenfolge Rot-Anteil (0 bis 63), Grün-Anteil (0 bis 63), Blau-Anteil (0 bis 63).

Hier noch ein Hinweis für das Schreiben oder Lesen der Farbwerte in die Palette

Schreiben:

```
OUT &H3C8, Farbe '(0-255)
OUT &H3C9,R 'R=Rotanteil(0-63)
OUT &H3C9,G 'G=Gelbanteil(0-63)
OUT &H3C9,B 'B=Blauanteil(0-63)
```

Lesen:

```
OUT &H3C7, Farbe '(0-255)
R=INP (&H3C9) 'R=Rotanteil(0-63)
```

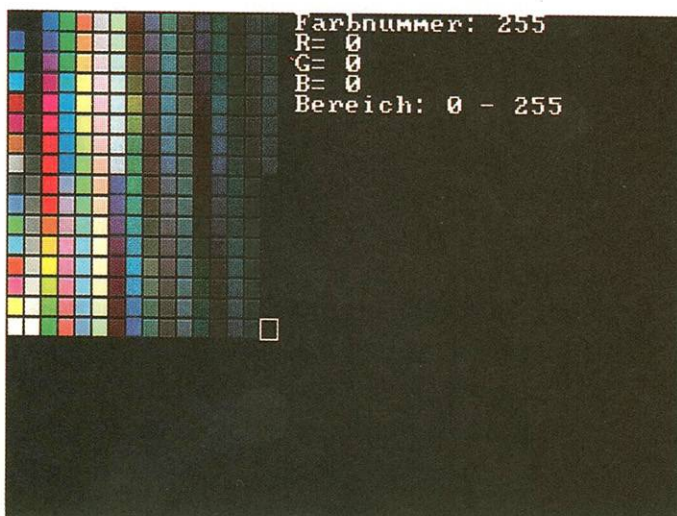


Bild 1. Die Standard-Farbpalette bei Basic: Sie kann jederzeit durch die Anweisung »PALETTE« – ohne Zusatzparameter – gesetzt werden.

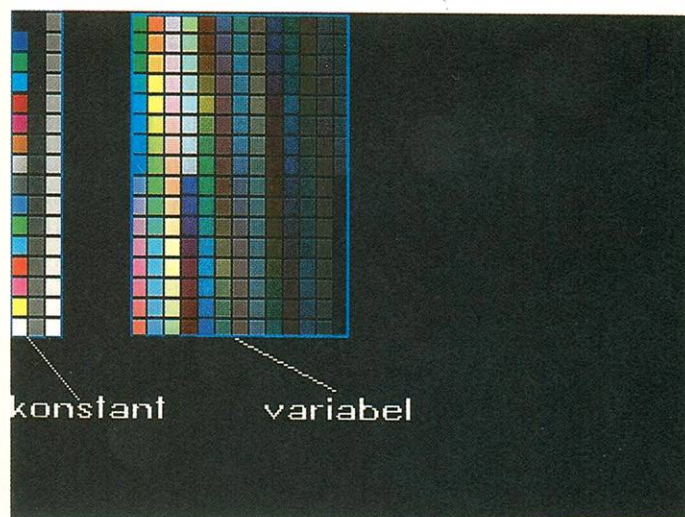


Bild 2. Eine modifizierte Standardpalette – erst 16 Basic-Grundfarben, dann 32 Grautöne für SW-Bilder. Der restliche Palettenbereich bleibt für verschiedenfarbige Bilder reserviert.

```
R=INP (&H3C9) 'G=Gelbanteil(0-63)
G=INP (&H3C9) 'B=Blauanteil(0-63)
```

Die Zugriffe mit »INP« und »OUT« lassen in ihrer Ausführungsgeschwindigkeit auf einem schnellen 286er nichts zu wünschen übrig und übertreffen andere Varianten unter Basic weit – auch dann, wenn innerhalb einer Schleife die komplette Palette durchlaufen wird.

Das kurze Programm (im Textkasten) erhöht den Rot-Anteil der kompletten Palette bei jedem Aufruf um den Wert 1 – das Bild wird mit jedem Aufruf rötlicher. Der komplette Durchlauf erfolgt so schnell, daß sich die Palette praktisch augenblicklich ändert.

Komplette Palette in Sekundenbruchteilen

Was sich der Reihe nach einlesen und schreiben läßt, kann auch immer problemlos gespeichert und geladen werden. Damit erlauben »INP« und »OUT« etwas, das mit einfachem Basic-Dialekt kaum zu bewerkstelligen ist. Die Standardmethode besteht darin, die Palette zum Speichern mit »INP« einzulesen und die Werte in eine Binärdatei zu

```
FOR i=0 TO 255      R=R+1
OUT &H3C7, i        OUT &H3C7, i
R=INP (&H3C9)       OUT &H3C9,R
R=INP (&H3C9)       OUT &H3C9,G
G=INP (&H3C9)       OUT &H3C9,B
NEXT
```

Mit diesem Programm erhöhen Sie den Rotwert in Ihrem Bild.

schreiben. Entsprechend können die so gewonnenen Paletten-dateien dann mittels »OUT« in die VGA-Karte geschrieben werden. Auch die zu Beginn dieses Workshops beschriebenen Shareware-Konverter-Utilities (Picem, VGA-CAD) erzeugen Paletten-dateien (Dateiendung »plt«) nach diesem Verfahren. Hier noch eine andere Methode zum Laden und Speichern von Paletten, die besonders schnell ist. Sie ist nicht sehr elegant, da sie für den Transport der Paletteninformationen den Beginn des Bildschirmspeichers verwendet. Folglich ist in der ersten Bildschirmzeile beim Laden und Speichern einer Palette ein »Streifen« zu sehen. Das macht aber nichts, wenn Sie erst die Palette und dann die Bildinformation laden, da der »Streifen« dann mit den Bilddaten überschrieben wird. Alternativ ist es erlaubt, als Segment-Adresse für »BLOAD/BSAVE« nicht den Bildschirmspeicher, sondern das Segment eines ausreichend gro-

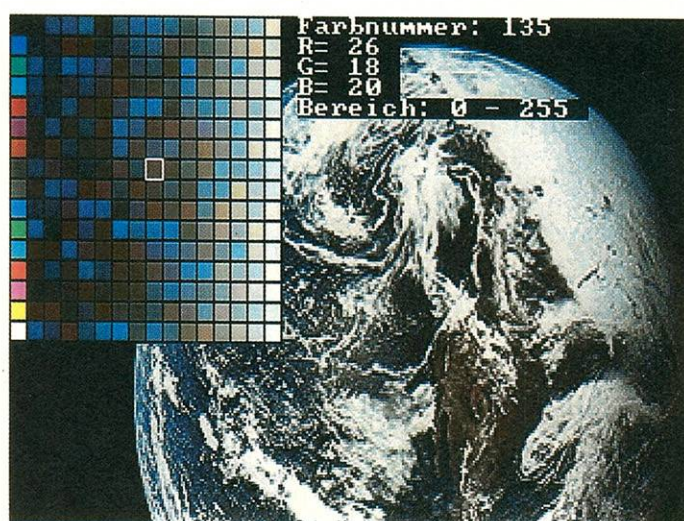


Bild 3. Paletten-Editor für Grafikentwickler zum Einbinden in eigene Basic-Programme – links die Paletten-darstellung in 16 Spalten mit je 16 Farben, rechts daneben Infos.

ßen Arrays zu setzen. Deshalb sollen Sie im folgenden die Methode beim Einsatz des VGA-Bildspeichers kennenlernen.

Sicherung der Farbpalette

Die Idee besteht darin, die Farbpaletten-Einträge der VGA-Karte zunächst auszulesen und dann per »BSAVE« als Datei zu speichern. Die folgenden Zeilen zeigen, wie die Palette gespeichert wird, wenn die 256 RGB-Werte in einem Array namens »Pal()« mit den Dimensionen »Pal(255,2)« vorliegen. Mit der Anweisung

```
DEF SEG=&HA000
```

wird der Segment-Beginn zunächst auf den Beginn des Bildschirmspeichers gelegt. Dort sollen die Daten vorläufig eingetragen werden, damit sie sich anschließend per »BSAVE« schnell speichern lassen. Immerhin läßt sich mit der Methode der Paletteninhalt der VGA-Karte auch ohne Umweg direkt nach Einlesen mit »INP« in den Bildschirmspeicher transportieren. Dann können Sie die Palette speichern und laden, ohne daß ein Extra-Array zum Einsatz kommt. Für welche Methode Sie sich entscheiden, kommt natürlich darauf an, ob es Ihnen etwas ausmacht, wenn der Bildschirminhalt verletzt wird oder nicht – geht es nur darum, den Bildschirm kurz vor Programmende oder einem neuen Bildaufbau zu speichern, beeinträchtigt diese Methode kaum. Zudem wurde bereits darauf hingewiesen, daß es sich stets lohnt, mit einer Palette zu arbeiten, die nur einmal zu Programmbeginn geladen wird – und dann stört das kurze Aufblinken der »Paletten-Farbzeile« wenig. Die folgende Schleife überträgt die 256 RGB-Werte nacheinander an den Anfang des Videospeichers:

```
FOR plt=0 TO 255
pltptr=plt*3
POKE (pltptr), Pal(plt,0)
POKE (pltptr+1), Pal(plt,1)
POKE (pltptr+2), Pal(plt,2)
NEXT
```

Mit der Zeile

```
BSAVE Palette$,0,768
```

wird die Farbpalette anschließend in einer Datei gesichert – die Größe dieser Datei ist $256 \times 3 = 768$ Byte.

Abschließend ist es sinnvoll, den Segment-Beginn mit der Anweisung

```
DEF SEG
```

ohne Parameter wieder auf den Basic-Bereich zurückzusetzen. Wird das vergessen, so stürzt ein Programm in der Regel nach kurzer Zeit ab, da es im Speicher durcheinanderkommt.

Im Rahmen einer Grafik-Programmentwicklung sollte stets berücksichtigt werden, daß ein Bild aus zwei getrennten Informationen besteht:

- Die Paletteninformation ist ausschließlich für die VGA-Karte zuständig, um ihr mitzuteilen, welche Farben sie in ihren 256 Töpfen für ein Bild bereitstellen soll.
- Die Bildinformation gibt für jeden verfügbaren Bildpunkt an, welchem Farbwert (Farbtopf) er zugeordnet ist.

Palette und Bild – zwei Paar Stiefel

Um zu verhindern, daß ein Bild für einen Sekundenbruchteil mit Falschfarben aufblinkt, sollten Sie stets zuerst die Paletten- und dann die Bildinformation laden.

Möchten Sie am Bildschirm eine Slideshow kontinuierlich ablaufen lassen (Bild für Bild nachladen und darstellen), so gibt es ein Problem. Verfügen die Bilder über unterschiedliche Farbpaletten, so läßt es sich ohne Bildschirmlöschchen zwischen den Bildern nicht verhindern, daß Falschfarben-Kurzbilder entstehen. Das Problem tritt nicht auf, wenn beide Bilder über eine identische Palette verfügen und bringt zudem einen Vorteil: die Palette muß nur ein einziges Mal für alle Bilder geladen werden. Das hat natürlich kaum Bedeutung, wenn Sie nur ein Bild laden wollen. Für schnelle und trickreiche Animations-Abläufe, bei denen Einzelbilder von der Festplatte geladen werden, ist das aber sehr wichtig – hier zählt jede Portion gesparte Rechen- und Ladezeit.

Bei der Entwicklung eines Grafikprojekts – sei es ein Spiel oder ein Low-End-Multimedia-System – empfiehlt es sich, zu Beginn des Projekts einen individuellen Standard für die Palette festzulegen, an dem sich dann alle Grafiken orientieren müssen. Es ist hierbei nicht wichtig, eine komplette Gesamtpalette auszuhecken. Vielmehr sollten Sie sich Gedanken über Bereiche in der Palette machen. Schließlich ist es erlaubt, nur Teile einer Palette zu laden, und gleichzeitig einen oder mehrere Teile stets konstant zu halten. Hieran führt unter anderem dann kein Weg vorbei, wenn Sie beispielsweise ein Programm schreiben möchten, das neben unterschiedlichen Bildern auch stets gleichfarbige (und vielfarbige) Menüs enthält – oder Sie möchten hin und wieder einen bestimmten Palettenbereich aufhellen oder abdunkeln. Es sieht beispielsweise recht schick aus, einen Bildhintergrund komplett abzudunkeln, dann ein Menü oder einen Text hell einzublenden und den Vorgang anschließend rückgängig zu machen.

Basic-Paletten exportieren

Bild 1 zeigt die Basic-Standardpalette. Sie liegt vor, wenn Basic gestartet oder der Befehl »PALETTE« ohne Zusatzparameter ausgeführt wird. Bei dieser Art der Palettendarstellung versteht sich die Farbenanordnung so: Die erste Farbe der Palette (Farbe 0) ist links oben, dann folgen darunter die nächsten 15 Farben. Die letzte Farbe (Farbe 255) finden Sie schließlich rechts unten in der Darstellung. Die erste Spalte der Palettendarstellung präsentiert die 16 Basic-Grundfarben, wie sie auch in den Basic-Manualen beschrieben sind. Es empfiehlt sich, diese 16 Farben für die eigene Palette beizubehalten. Dann steht für Programm-Hinweistexte und Menüs ein großer konstanter Palettenbereich zur Verfügung. Die zweite Spalte besteht aus 16 Grautönen. Wer in einem Programm gleichzeitig Farb- und Schwarzweiß-Bilder darstellen möchte, sollte den verfügbaren Grauwertbereich vergrößern. Ob ein Bild aus 64 oder 63 Graustufen besteht, ist für die Darstellungsqualität nicht sehr entscheidend – das Auge unterscheidet nur etwa 32 Werte im Graubereich. 16 sind allerdings deutlich zu wenig. Bild 2 zeigt eine modifizierte Palette: Sie enthält stets konstant 16 Basic-Grundfarben und 32 Grautöne, die restlichen 208 Far-

ben bleiben frei beziehungsweise werden im Laufe eines Programms unterschiedlich belegt. Wenn Sie eine Variante suchen, in Basic entstandene oder im PLT-Format vorliegende Paletten in ein anderes Grafikprogramm einzulesen, gehen Sie wie folgt vor: ein Capture-Programm starten, Bildschirminhalt »abfotografieren«, dann das »eingefangene« Bild ins Grafikprogramm laden, und die Palette steht dort zur Verfügung. Capture-Programme gehören zum Lieferumfang nahezu jedes Grafikprogramms.

Der Palette Farben zuweisen

Werden beim Ablauf eines Programms bestimmte Farbtopfe nicht gleichzeitig benötigt, so können Sie ihnen natürlich per »OUT« neue Werte zuweisen, also etwa unterschiedliche Menüs, die nicht gleichzeitig auftreten, verschiedenfarbig gestalten, ohne daß dafür verschiedene Palettenbereiche erforderlich sind. Ein Bild von 256 auf 208 Farben zu reduzieren, ist keine tragische Sache. Das läßt sich mit nahezu jedem besseren Grafik-/Malprogramm erledigen und führt selten zum Qualitätsverlust eines Bildes. In der Grafikwelt gibt es sogenannte Standard-RGB-Paletten. Bei ihnen wurden die Farben so gewählt, daß alle Farben annähernd gleichwertig vertreten sind. Diese Paletten eignen sich prinzipiell für die Darstellung oder Umrechnung jeglicher Bilder. Die Ergebnisse lassen allerdings meist sehr zu wünschen übrig. Eine Standardpalette für alles kann keineswegs eine ordentlich organisierte Palette ersetzen, die Teilbereiche nachläßt.

Hier der Vorgang, wenn Sie ein beliebiges Bild – etwa in GIF oder PCX – für das Einlesen in Basic vorbereiten möchten:

- Laden Sie das Bild mit einem Grafiktool und betrachten Sie die Palette. Das eingesetzte Zeichenprogramm sollte in der Lage sein, Bereiche der Palette freizustellen beziehungsweise das Bild an Farben zu reduzieren. Hinweise zu den entsprechenden Funktionen finden sich im Manual zum Malprogramm. Befreien Sie das Bild von jenen Farben (Farbtopfen), die Sie in Ihrer Palette konstant halten möchten.
- Nachdem das modifizierte Bild gespeichert ist, kann es mit einem Grafikkonverter (oder einer der Methoden, die zu Beginn des Workshops vorgestellt wurden) ins Basic-ladbare Format konvertiert werden.
- Zu Beginn Ihres Grafikprojekts sollten Sie zuerst die komplette Grundpalette laden. Wird dann ein Bild geladen, das auf einen Teil der Palette reduziert ist, geht das so: erst den entsprechenden Teilbereich laden (in der VGA-Tool-

PALETTENEDITOR-TASTENBELEGUNG

F1 - Diese Hilfeseite
F2 - Backup der aktuellen Palette
F3 - Palette von letztem Backup
F4 - Palette speichern
F5 - Palette laden
s - Bereichstart setzen
e - Bereichende setzen
+ - Bereich aufhellen
- - Bereich abdunkeln
t - tausche markierten Bereich
m - Standard BASIC + Graustufen
r / **SHIFT + r** - Rot-Anteil -/+
g / **SHIFT + g** - Grün-Anteil -/+
b / **SHIFT + b** - Blau-Anteil -/+
ESC - Paletteneditor verlassen

Bild 4. Tastenbelegung des Paletten-Editors – diese Hilfeseite ist jederzeit mit Funktionstaste [F1] abrufbar.

box des Workshops finden Sie hierfür Routinen), dann das Bild.

Im Listing finden Sie einen Paletten-Editor. Manipulationen an der Palette lassen sich zwar mit jedem Grafikprogramm durchführen, allerdings ist es lästig, immer zwischen Basic-Programmierungsumgebung und Grafiktool zu wechseln. Der vorgestellte Paletten-Editor ist für alle Programmierer gedacht, die im Grafikmodus 13 arbeiten und bei der Entwicklung Eingriffe an der Palette vornehmen. Sie können das Listing abtippen und dann direkt als Modul laden. Zum Start des Paletteneditors ist dann einfach die »SUB«-Routine »ZeigPal« aufzurufen. Vor dem Aufruf muß der Grafikmodus 13 per Anweisung »SCREEN 13« aktiviert sein. Der Paletten-Editor ist weniger als eigenständiges Tool, sondern vielmehr als Tool für die Programmierung gedacht. Starten Sie den Editor also einfach an der Stelle in Ihrem Programm, an der Sie einen Zugriff auf die Palette wünschen. Alternativ können Sie natürlich auch einen Funktionstastenaufruf (»ON KEY«) für den Editor anlegen. Im Paletten-Editor wurden bewußt keine Sub-Routinen verwendet. So können Sie den Editor schneller in eigene Programme einbinden und entfernen – egal unter welcher Basic-Entwicklungsumgebung Sie arbeiten. Beim Start speichert der Editor stets den aktuellen Bildschirminhalt, um ihn später restaurieren zu können.

Besonderheiten des Editors

Sie können also Paletten-Manipulationen durchführen und dann Ihr Programm weiterlaufen lassen. Beachten Sie, daß der Editor den Bildschirminhalt in eine Datei sichert – auf dem aktuellen Datenträger sollten mindestens 66 KByte frei sein! Das Listing ist ausführlich mit Rem-Anweisungen dokumentiert, so daß eingehende Erläuterungen kaum erforderlich sind. Auf umfassende Sicherheitskontrollen beim Editor-Ablauf wurde verzichtet, um das Tool möglichst kompakt zu gestalten. Mit dem Editor lassen sich nahezu alle Standard-Paletten-Funktionen ausführen. Außerdem werden einige Besonderheiten geboten: Sie können beispielsweise beim Laden einer Palette nur einen bestimmten Bereich berücksichtigen. Sämtliche Funktionen des Editors wirken sich entweder auf eine einzelne Farbe oder einen definierten zusammenhängenden Bereich aus.

Bei Computerspielen ist oftmals eine interessante Sache zu beobachten – insbesondere bei Grafikadventures der Sierra-Art: Spielfiguren (sogenannte Sprites) können sich vor oder hinter Objekten bewegen (beispielsweise Spielfigur läuft hinter Busch vorbei) der Objekte selbst zwischen meh-

rerer anderen. Für solche Animationsvorhaben gibt es zwei grundsätzliche Verfahren:

- Am einfachsten ist es, das Komplettbild, das aus mehreren Grafikobjekten besteht, bei jeder Veränderung neu aufzubauen. In der Praxis kommt dieses Verfahren bestenfalls dann zum Einsatz, wenn sich ein Bildschirm nur aus sehr wenigen kleinen Objekten zusammensetzt. Andernfalls kostet es zu viel Rechenzeit, das Verfahren ist schlichtweg zu langsam.

- Für die verschiedenen Tiefenebenen im Bild werden eigene Palettenbereiche reserviert. Diese können durchaus sogar mit identischen Farben belegt sein. Relevant ist lediglich, daß Bilder unterschiedlicher Ebenen über unterschiedliche Farbtöpfe verfügen. Gemäß einer Tabelle weiß das Programm stets, welche Farbtöpfe zu welcher Ebene gehören. Wird ein Objekt (die Spielfigur) am Bildschirm zwischen zwei Ebenen platziert, so muß das Objekt Pixel für Pixel gesetzt werden. Vor jedem Setzen ist zu prüfen, welchem Farbbereich die Position am Bildschirm, die gesetzt werden soll, entspricht. Befindet sich ein Pixel der Spielfigur vor einem Objekt, so wird es gesetzt, befindet es sich dahinter, wird es nicht gesetzt. So läßt sich beispielsweise schnell und einfach eine Spielfigur durch einen »dreidimensionalen« Wald steuern. Eines ist bei diesem Verfahren noch wichtig: eine Farbe der Palette (am günstigsten Farbe 0) muß als durchsichtiger Hintergrund definiert sein. Schmerzhaft für den Nur-Basic-Programmierer: auch das zweite Verfahren kostet viel Rechenzeit und läßt sich kaum schnell genug programmieren. Hier führt schlicht kein Weg an einer Assembler-Routine vorbei. Allerdings: per »INP« und »OUT« kann nicht nur auf die Farbpalette, sondern auch auf viele andere Funktionen der VGA-Karte zurückgegriffen werden.

Vorsicht mit den Ports

Dabei gibt es zwei Arten von Funktionen: Standardfunktionen, die für alle kompatiblen VGA-Karten gelten und spezielle Funktionen, die vom Chipsatz der VGA-Karte abhängen. An dieser Stelle nur soviel: die Standard-Funktionen der VGA-Karte sind enorm. Sie gestatten es beispielsweise, den kompletten Videospeicher der Karte zu verwenden, um Bildteile blitzschnell verschieben zu können. Paletten-tricks, die für das zuletzt beschriebene Verfahren erforderlich sind, werden ebenfalls bereitgestellt. Lassen Sie sich von den hier vorgestellten Inp-/Out-Beispielen mit der VGA-Karte allerdings nicht täuschen: Die hardwarenahe Programmierung der VGA-Karte erscheint auf den ersten



Bild 5. Jede Funktion des Paletten-Editors wirkt sich entweder auf eine einzelne Farbe oder einen markierten Farbbereich aus.

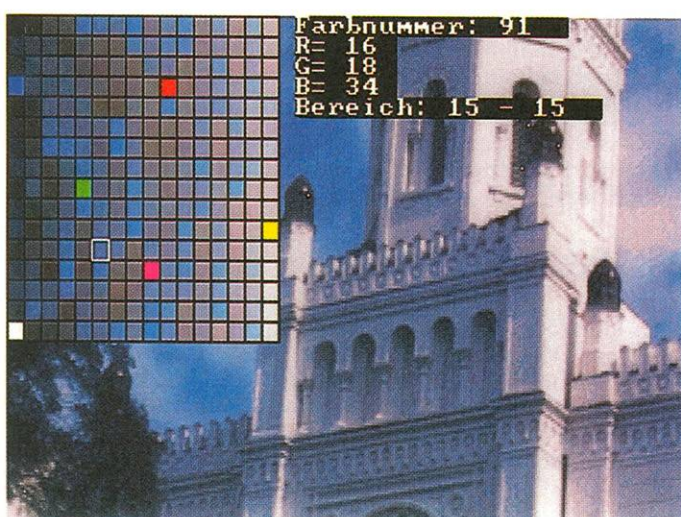


Bild 6. Ein Bild und seine Palette – letztere wird bei den folgenden Bildern manipuliert.

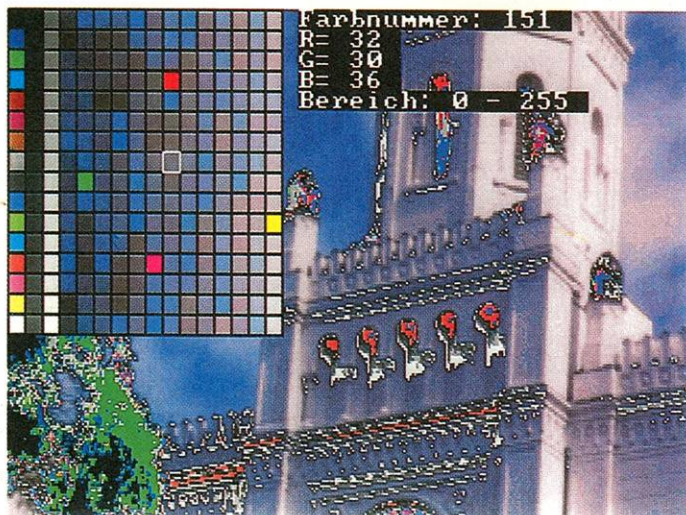


Bild 7. Hier wurde die Palette aus Bild 6 der erweiterten Standard-Farbpalette mit 32 Graustufen angepaßt. Der Paletten-Editor erledigt dies per Tastendruck.

Blick einfacher, als sie tatsächlich ist. Entscheidend ist es nicht nur zu wissen, welche Port-Adressen welchem Zweck dienen, sondern vor allem, in exakt welcher Reihenfolge die Funktionen auszuführen sind – und das ist recht kompliziert. Die Innereien der VGA-Karte sind ein Thema für fortgeschrittene Grafikprogrammierer – am besten machen Sie sich mit einem Standardbuch zur VGA-Karten-Hardware ans Studium, die mitgelieferten Dokumentationen der VGA-Karten sind oft unzureichend. Die übliche Warnung zum Schluß: wer nicht über ausreichende Kenntnisse verfügt, kann über die Ports durchaus seinen Monitor ins Jenseits befördern. Und zwar dann, wenn über Ports eine Auflösung oder Bilddarstellungs-Frequenz aktiviert wird, die der Monitor nicht verkraftet. Bild 3 zeigt den gestarteten Paletten-Editor. Im Hintergrund ist ein Bild sichtbar, das vor Aufruf des Editors geladen wurde. Die Palette wird im oberen linken Bildschirmbereich dargestellt, Farbe 0 oben links, Farbe 255 unten rechts. In der Palettendarstellung ist eine Marke (weißes Quadrat) zu sehen, die sich mit den Pfeiltasten auf beliebige Farben in der Palette bewegen läßt. Neben der Palettendarstellung finden sich Informationszeilen, die bei jedem Bewegen der Marke aktualisiert werden: Die Farbnummer entspricht der Farbe unter der Marke, darunter sind die RGB-Werte dieser Farbe eingetragen. Die Bereichszeile gibt an, auf welchen Bereich der Palette sich Operationen aktuell auswirken. Bei Start des Editors ist standardmäßig die gesamte Palette markiert. Wird ein anderer Standardbereich gewünscht, so können Sie im Programm die Zeilen 16 und 18 modifizieren. Der Editor speichert nach seinem Start außerdem die aktuelle Farbpalette im Array Pal, das so dimensioniert ist:

```
DIM Pal (255,2)
```

Pal entspricht einer Tabelle, in der die RGB-Werte der 256 Farben separat eingetragen sind.

Wichtige Editor-Funktionen

Bei allen Editor-Funktionsausführungen gilt: Sie können so lange per Tastendruck rückgängig gemacht werden, bis ein neues Backup der Palette angefertigt wird. Sie können also experimentieren. Letztlich geht eine bestehende Palette nur dann verloren, wenn sie durch Angabe desselben Dateinamens überschrieben wird. Die Funktionen des Paletten-Editors (Bild 4) sind im Kasten zusammengefaßt. Hier einige Beispielanwendungen:

Bei Bild 5 wurde durch Drücken der Taste [-] (Abdunkeln) lediglich das hintere Drittel der Palette abgedunkelt. Der

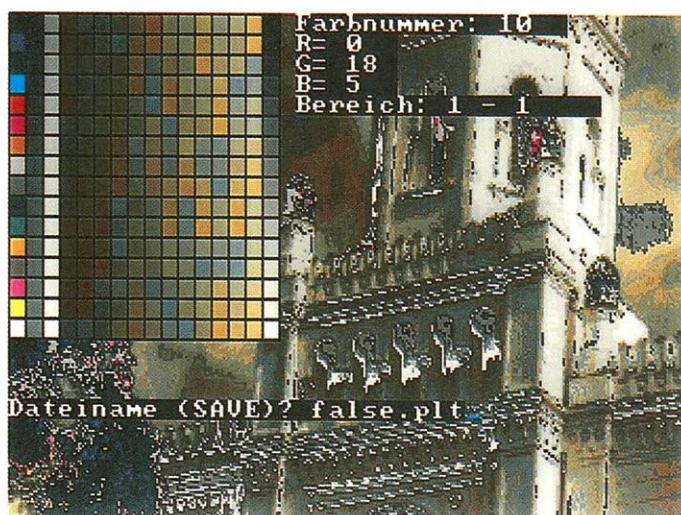


Bild 8. Falschfarben-Darstellung durch willkürliches Laden einer anderen Farbpalette – dabei entstehen oft recht interessante Darstellungen.

abgedunkelte Bereich wurde zuvor mit den Tasten [s] (Bereichsstart) und [e] (Bereichsende) markiert. Bild 6 zeigt ein geladenes 256-Farben-Bild mit seiner Standardpalette. Bei Bild 7 wurde per [m] der eingangs beschriebene Standardpaletten-Bereich generiert. Das Bild im Hintergrund zeigt, wie sich dies auf seine Darstellung auswirkt. Vor dem Generieren der erweiterten Standardpalette ist es also stets erforderlich, das Bild mit einem Malprogramm oder Grafik-Utility zu konvertieren (Befreien der ersten 48 Farben). In Bild 8 sehen Sie, wie durch Laden einer andersfarbigen Palette sogenannte Falschfarbenbilder entstehen. Diese kommen oft bei wissenschaftlichen Auswertungen von Bildern zum Einsatz – beispielsweise bei der Computer-Tomographie.

Der Funktionsumfang des Paletten-Editors

Ein Spezialtrick des Paletten-Editors ist in Bild 9 zu sehen. Hier wurde in das Bild 8 ein Teil der Originalpalette zurückgeladen (hintere Palettenhälfte), womit das Bild zum Teil seinen Falschfarb-Charakter wieder verloren hat. Der Paletten-Editor gestattet beim Laden einer Palette, nur einen bestimmten Palettenbereich zu berücksichtigen – ein Unterfangen das bei vielen DOS-Zeichenprogrammen nur umständlich machbar ist. Damit haben Sie die Chance, aus bestehenden Paletten neue zusammenzustellen. Zudem können Sie gut experimentieren, welche Farben eines Bildes am meisten reduziert werden sollten.

Der Paletten-Editor wird Ihnen im Rahmen der Grafikprogramm-Entwicklung gute Dienste leisten.

Das Listing enthält zunächst drei Hauptprogramm-Modulzeilen und dann die »SUB ZeigPal«-Routine. Beachten Sie, daß im Sub-Modul zu Beginn die Zeile »DEFINT A-Z« stehen muß.

Basic-Einsteiger sollten sich mit dem Listing ausführlich beschäftigen, wenn sie wissen möchten, wie sie im Schnellverfahren ein komfortables und bedienbares individuelles Tool entwickeln. An dieser Stelle für alle Interessierten noch einige Hinweise zum Listing:

In Zeile 6 bis 26 findet die Initialisierung des Editors statt. Hier passiert unter anderem folgendes: Dimensionieren des Paletten-Puffers und Festlegen des Standardbereichs beim Programmstart.

Mit den Zeilen 27 bis 39 stellen Sie die Farbpalette dar. In zwei Schleifen werden Zeilen und Spalten der Palette durchlaufen und die aktuelle Farbe dabei jeweils um den Wert 1 erhöht. Durch die Anweisung »COLOR Farbe« wird die Farbe gesetzt und mit »PRINT CHR\$(219)« ein ausge-



Bild 9. Mischen zweier Palettenbereiche – hier wurde in das Falschfarbenbild aus Bild 8 ein Teil der Originalpalette zurückgeladen (die hintere Palettenhälfte).

fülltes Rechteck mit diesem Farbtönen gesetzt. Die Initialisierung der Ausgangsposition der Editormarke und das Zeichnen des Markenrechtecks erreichen Sie mit den Zeilen 40 bis 42.

Die Zeilen 44 bis 127 stellen das eigentliche Hauptprogramm dar. In einer Endlosschleife »DO-LOOP« wird die Tastatur abgefragt und bei Tastendrücken das entsprechende Unterprogramm mit »GOSUB« aufgerufen. Beim Setzen von Bereichsanfang und Bereichsende wird über eine If-Abfrage sichergestellt, daß das Bereichsende nicht vor dem Bereichsbeginn liegt. Ist das der Fall, werden die entsprechenden Variablen einfach mit »SWAP« getauscht. Wenn Lautsprechertöne nerven, der kann die Sound-Anweisungen weglassen – sie sind als Bestätigung für eine Funktionsausführung allerdings gut geeignet. Am Ende jedes Durchlaufs der Schleife werden die Infozeilen neben der Palettendarstellung aktualisiert.

Den Start des Paletten-Editors erreichen Sie durch den Aufruf der Sub-Routine »ZeigPal« und verlassen ihn durch Betätigen der Escape-Taste. Auf dem aktuellen Datenträger müssen 66 KByte frei sein. Die Positionen in der Palette werden durch Bewegen der Marke mit den Pfeiltasten angewählt. Über die Funktionstaste F1 können Sie jederzeit die Belegung sämtlicher Tasten einblenden lassen.

● **Farbbereich markieren:** Beim Start sind standardmäßig alle Farben markiert (Bereich: 0 bis 255). Sämtliche Funktionen des Editors (außer »Speichern«) wirken sich auf den markierten Bereich aus. Bereichsanfang und -ende setzen Sie durch Drücken folgender Tasten, was jeweils durch einen Signalton bestätigt wird:

Taste [s]: Bereichsstart und Taste [e]: Bereichsende.

Sollen sich die Funktionen des Editors lediglich auf eine einzige Palettenposition auswirken, peilen Sie diese mit der Marke an und drücken Sie erst die Taste [s], dann [e], womit die Farbe als Bereich markiert ist.

● **Farbe ändern:** Die RGB-Werte der aktuellen Farbe werden stets in den Infozeilen dargestellt. Um die RGB-Teile eines Palettenbereichs (oder nur einer einzigen Farbe) zu ändern, dienen folgende Tasten beziehungsweise Tastenkombinationen:

- r – Rot-Anteil um 1 erniedrigen
- g – Grün-Anteil um 1 erniedrigen
- b – Blau-Anteil um 1 erniedrigen
- SHIFT + r – Rot-Anteil um 1 erhöhen
- SHIFT + g – Grün-Anteil um 1 erhöhen
- SHIFT + b – Blau-Anteil um 1 erhöhen

Achtung: Die Auswirkungen bei RGB-Veränderungen an der Gesamtpalette lassen sich unter Umständen nur durch

Programmautoren gesucht

Haben Sie Programme geschrieben, die noch nicht veröffentlicht wurden? Dann schicken Sie sie an die DOS-Redaktion. Wir suchen Utilities, Anwendungen und Tips und Tricks in Basic, Pascal, C oder einer anderen Programmiersprache, die unter MS-DOS auf dem PC eingesetzt wird.

Schicken Sie Ihr Programm auf einem Datenträger (5 1/4- oder 3 1/2-Zoll-Format) an

**Redaktionsbüro
DOS International
»DOS Software«
Gruberstraße 46c
8011 Poing**

Damit wir Ihr Programm mit Hilfe unseres Checksummers übersichtlich formatiert ausdrucken können, möchten wir Sie bitten, – soweit möglich – die Breite der Programmzeilen im Quellcode auf maximal 44 Zeichen (mit Einrückungen) zu begrenzen.

In dem Begleitschreiben muß unbedingt der Interpret beziehungsweise Compiler genannt werden. Vermerken Sie bitte auch Ihre Anschrift sowohl auf dem Anschreiben als auch auf der Diskette, so daß wir Ihnen im Falle einer Veröffentlichung Ihren Honorarscheck problemlos zuschicken können.

WIESE HANDELSVERTRETUNG Ihr Modem-Discounter

Lightfax 9624P

Pocket-Fax/Modem, 2400 1200 / 300 bps Modem, Voll-Duplex, 9600 bps Fax senden (G3), 4800 bps empfangen, inklusiv Netzteil, bis zu 8 Stunden Betrieb mit einer Batterie! Größe: 7,8 x 12,6 x 2,51 cm, Quicklink II - Fax - Software.

379.- DM

Twincom 14.400

extern, 14.400 bps Modem, Voll-Duplex V.32, V.32bis, Mnp5, V.42bis bis zu 57.600 bps, 14.400 bps Fax (G3) Senden & Empfangen, Quick Link II Fax DOS Software, FAXIT Windows Fax Software.

899.- DM

ZyXEL



ZyXEL U-1496E

G3-Fax & Highspeed Modem in einem! 1.200 - 14.400 bps, V.22, V.22bis, V.32, V.32bis, Trellis Modulation, MNP2-5, V.42, V.42bis, Security Callback & Password-Schutz, Remote Configuration, Datendurchsatz / DTE-Speed bis zu 57.600 bps, 2 Jahre Garantie.

998.- DM

ZyXEL U-1496 LCD

besitzt alle Features des U1496E, Hayes & V.25bis Befehlssatz, V.33 / 14.400 synchrone Datenübertragung, unterstützt 2/4 Draht Standleitungen, Dial Backup Funktion, Modem-Einstellung über menügeführtes LC-Display und Front-Taster, DTE-Speed bis 76.800 bps, 2 Jahre Garantie.

1.398.- DM

Alle Preise zzgl. Versand per Post-NN

Händleranfragen erwünscht!

Unsere Pluspunkte:

- Hohe Liefertfähigkeit
- 12 Monate Mindestgarantie
- ROM Update-Service
- technische Hotline
- eigener Reparaturservice

Diese Modems sind nur für den Export bestimmt. Der Anschluß an das Telefonnetz der BRD ist strafbar!

Tel.: 02 11 / 6 79 85 47
Fax.: 02 11 / 68 32 07

die Restaurierungs-Funktion (Funktionstaste F3) rückgängig machen.

● Helligkeit ändern:

- + – Helligkeit im markierten Palettenbereich erhöhen
- – Helligkeit im markierten Palettenbereich vermindern

● **Palette restaurieren** (»UNDO«): Der Editor legt beim Start ein Backup der aktuellen Palette an. Durch Druck auf die Funktionstaste F3 können Sie den Ausgangs-Palettenzustand wieder herstellen.

Sind Sie mit einer Palette zufrieden, so kann jederzeit per [F2] ein Backup der aktuellen Palette angefertigt werden. Der vorige Inhalt des Backup-Puffers wird dabei natürlich überschrieben.

● **Palette speichern:** Nach Druck auf die F4-Taste werden Sie aufgefordert, einen Dateinamen anzugeben, unter dem die aktuelle Palette dann gespeichert wird. Beim Speichern wird stets die komplette Palette (Farbe 0 bis 255) gesichert.

● **Palette laden:** Die Ladefunktion wird über [F5] gestartet. Sie können jede Palette laden, die mit dem Paletten-Editor gespeichert wurde und darüber hinaus weitere: beispielsweise Paletten von Konvertern wie Picem oder dem Zeichenprogramm VGACAD. Verträgliche Palettendateien sind solche, bei denen RGB-Werte separat gespeichert sind. Das ist bei vielen Palettendateien der Fall. Basic-Palettendateien sind $256 \times 3 = 768$ Byte (zuzüglich Dateihäuser) groß.

Beim Ladevorgang werden Farben der neuen Palette nur in den markierten Bereich der aktuellen geladen. Soll also eine komplette neue Palette geladen werden, so muß zuvor 0 als Bereichsstart und 255 als Bereichsende gesetzt werden.

● **Bereich tauschen:** Durch Betätigen der t-Taste lassen sich schnell und einfach Bereiche der Palette verschieben. Der zu verschiebende Bereich muß zunächst mit den Tasten

[s] und [e] markiert werden. Anschließend fährt man die Editormarke auf den neuen Startbereich und betätigt [t]. Der alte Bereich wird dann an die neue Startposition verschoben. Gleichzeitig werden in den Quellbereich die Farben des neuen Bereichs übertragen. [t] dient also konkret zum Vertauschen zweier Palettenbereiche.

Durch Druck auf die Taste [m] belegt der Editor die aktuelle Palette mit Standardfarben: in der ersten senkrechten Palettenreihe die 16 Basic-Standardfarben, danach 32 Grautöne.

(Michael Nickles/uk)

Die DOS Extra ist eine Zeitschrift zum Mitmachen. Vor allem die Tips-&-Tricks-Rubriken leben von Ihrer Mitarbeit. Wenn Sie sich bereits besser mit Ihrem PC auskennen, überlegen Sie: Was hat Ihnen als Einsteiger damals besondere Schwierigkeiten bereitet? An welchem Problem sind Sie wieder und wieder gescheitert? Schreiben Sie uns! Durch Ihre Einsendung erhalten auch andere Leser die Gelegenheit, von Ihrer Erfahrung zu profitieren. Ein Beitrag für »Tips & Tricks für Einsteiger« muß weder journalistisch ausgefeilt noch eine überragende Neuentdeckung auf dem PC-Sektor sein. Und ein angemessenes Honorar gibt es dafür natürlich auch (bei Veröffentlichung). Die Höhe des Honorars richtet sich nach der Qualität der Idee und eventueller Listing. Schreiben Sie an:

**Redaktion DOS International,
Stichwort Tips & Tricks,
Gruberstraße 46c, 8011 Poing bei München**

```
[WL3] 1: 'Programm: palette.bas
[UX5] 2: 'Funktion: Anzeige der Farbpaletten-
[9B2] 3: '
[2N3] 4: 'Sprache: QuickBasic 4.5
[G04] 5: 'Autor: Michael Nickles
[1W4] 6: '(c) 1992 DMV Widuch GmbH & Co. KG
[000] 7:
[MM3] 8: DECLARE SUB zeigpal ()
[7C1] 9: SCREEN 13
[1C1] 10: zeigpal
[000] 11:
[TL1] 12: DEFINT A-Z
[CK3] 13: 'Aufruf mit: ZeigPal'
[E22] 14: SUB zeigpal
[Y76] 15: REDIM Pal(255, 2) 'Puffer für Farbpalette
[E21] 16: Start:
[GZ2] 17: GOSUB PaletteKeep
[SX6] 18: 'Backup von aktueller Palette anfertigen
[852] 19: StartFarbe = 0
[LZ4] 20: 'Arbeitsbereich in der Palette
[L22] 21: EndFarbe = 255
[DX4] 22: '(zu Beginn die ganze Palette)
[FZ5] 23: 'Bildsch. in Datei "backup.$$$" sichern
[W72] 24: DEF SEG = &H0000
[KE5] 25: 'Segment auf Beginn VGA-Speicher
[NH3] 26: BSAVE "backup.$$$", 0, 64000
[XW6] 27: DEF SEG 'Segment auf BASIC zurücksetzen
[HM5] 28: 'Palette am Bildschirm darstellen
[R91] 29: Farbe = 0
[PY3] 30: 'Farbzähler auf 1. Farbe
[PW2] 31: FOR Spalte = 0 TO 15
[705] 32: 'Palette in 16 Reihen a 16 Farben
[SY2] 33: FOR Zeile = 0 TO 15
[7R5] 34: COLOR Farbe 'Aktuelle Farbe setzen
[G14] 35: LOCATE Zeile + 1, Spalte + 1
[565] 36: 'Cursor auf Paletten-Position
[K92] 37: PRINT CHR$(219);
[735] 38: 'Ausgefülltes Rechteck setzen
[N08] 39: LINE (Spalte*8, Zeile*8)-(Spalte*8+8, Zeile*8+8), 0, B
[B15] 40: Farbe = Farbe+1 'Nächste Farbe
[ZS0] 41: NEXT
[ZS0] 42: NEXT
[FS6] 43: Spalte = 0 'Spalten-Startpos. des Zeigers
[CH6] 44: Zeile = 0 'Zeilen-Startpos. des Zeigers
[FZ6] 45: LINE (Spalte, Zeile)-(Spalte+8, Zeile+8), 15, B
[M57] 46: 'Abfrage der Steuerungstasten für die vier
[3W6] 47: 'Richtungen rauf, runter Links und rechts
```

```
[PP5] 48: 'und Bewegen des Paletten-Zeigers.
[NB0] 49: DO
[072] 50: Wert$ = INKEY$
[VE6] 51: IF LEN(Wert$)>1 THEN Wert$=RIGHT$(Wert$,1)
[GU2] 52: IF Wert$ <> "" THEN
[WU6] 53: LINE (Spalte, Zeile)-(Spalte+8, Zeile+8),
0, B
[2T2] 54: SELECT CASE Wert$
[Z63] 55: CASE IS=CHR$(27)'ESC
[W94] 56: 'Bildschirm restaurieren
[552] 57: DEF SEG=&H0000
[KE5] 58: 'Segment auf Beginn VGA-Speicher
[ZS2] 59: BLOAD "backup.$$$", 0
[091] 60: DEF SEG
[XK5] 61: 'Segment auf BASIC zurücksetzen
[TE1] 62: EXIT SUB
[3U5] 63: CASE IS = "s" 'StartFarbe markieren
[UV3] 64: SOUND 800, .3: SOUND 1000, .3
[LV2] 65: StartFarbe = Farbe
[BH4] 66: IF StartFarbe>EndFarbe THEN
[UA4] 67: SWAP StartFarbe, EndFarbe
[PD5] 68: CASE IS = "e" 'EndFarbe markieren
[UV3] 69: SOUND 800, .3: SOUND 1000, .3
[HG2] 70: EndFarbe = Farbe
[5R4] 71: IF StartFarbe > EndFarbe THEN
[UA4] 72: SWAP StartFarbe, EndFarbe
[NU3] 73: CASE IS = "k" 'Links
[YK5] 74: IF Spalte-8>0 THEN Spalte=Spalte-8
[FY3] 75: CASE IS = "m" 'rechts
[6X5] 76: IF Spalte+8<=8*15 THEN Spalte=Spalte+8
[PR3] 77: CASE IS = "h" 'rauf
[705] 78: IF Zeile-8>=0 THEN Zeile=Zeile-8
[K14] 79: CASE IS = "p" 'runter
[7G5] 80: IF Zeile+8<=8*15 THEN Zeile=Zeile+8
[TU1] 81: CASE IS = "r"
[912] 82: GOSUB DecRed
[TU1] 83: CASE IS = "r"
[622] 84: GOSUB IncRed
[EUI] 85: CASE IS = "g"
[HD2] 86: GOSUB DecGreen
[EUI] 87: CASE IS = "g"
[EE2] 88: GOSUB IncGreen
[9U1] 89: CASE IS = "b"
[G52] 90: GOSUB DecBlue
[9U1] 91: CASE IS = "b"
[D62] 92: GOSUB IncBlue
[KT1] 93: CASE IS = "+"
[WN2] 94: GOSUB Aufheilen
```



```
[MT1] 95: CASE IS = "-"
[PR2] 96: GOSUB AbdunkeLn
[MU1] 97: CASE IS = "m"
[9D3] 98: GOSUB MacheStandard
[VU1] 99: CASE IS = "t"
[RF3] 100: GOSUB TauscheBereich
[VN5] 101: CASE IS = CHR$(59) 'Funktionstaste F1
[7Z1] 102: GOSUB HiLife
[CN5] 103: CASE IS = CHR$(60) 'Funktionstaste F2
[UV3] 104: SOUND 800, .3: SOUND 1000, .3
[GZ2] 105: GOSUB PaLetteKeep
[GN5] 106: CASE IS = CHR$(61) 'Funktionstaste F3
[UV3] 107: SOUND 800, .3: SOUND 1000, .3
[NM3] 108: GOSUB PaLetteRestore
[MN5] 109: CASE IS = CHR$(62) 'Funktionstaste F4
[UV3] 110: SOUND 800, .3: SOUND 1000, .3
[8S3] 111: GOTO PaLetteSpeichern
[SN5] 112: CASE IS = CHR$(63) 'Funktionstaste F5
[UV3] 113: SOUND 800, .3: SOUND 1000, .3
[103] 114: GOTO PaLetteLaden
[GS1] 115: END SELECT
[FZ6] 116: LINE (SpaLte,ZeiLe)-(SpaLte+8,ZeiLe+8),15,B
[EY4] 117: Farbe=POINT(SpaLte+1, ZeiLe+1)
[EN1] 118: LOCATE 1, 18
[F71] 119: COLOR 15
[644] 120: PRINT "Farbnummer: "; Farbe
[XD2] 121: OUT &H3C7, Farbe
[W02] 122: r = INP(&H3C9)
[602] 123: g = INP(&H3C9)
[WZ1] 124: b = INP(&H3C9)
[F34] 125: LOCATE 2, 18: PRINT "R="; r
[314] 126: LOCATE 3, 18: PRINT "G="; g
[104] 127: LOCATE 4, 18: PRINT "B="; b
[6T1] 128: LOCATE 5, 18:
[YH6] 129: PRINT "Bereich: "; StartFarbe;"-";EndFarbe
[801] 130: END IF
[HS0] 131: LOOP
[5R3] 132: '-----
[W86] 133: 'Unterprogramme - können auch in SUB'
[6C5] 134: 'umgewandelt und dann für eigene
[376] 135: 'vorhaben verwendet werden. Hier aLs'
[346] 136: 'GOSUB's zusammengefaßt, damit der
[SU5] 137: 'PaLetten-Editor schnell in eigene
[7U5] 138: 'Programme eingebunden werden kann!
[5R3] 139: '-----
[YD5] 140: 'Routine tauscht den markierten
[1X4] 141: 'PaLettenreih an die aktueLLe
[G23] 142: 'Cursor-Position.
[9R2] 143: TauscheBereich:
[TT4] 144: FOR i = StartFarbe TO EndFarbe
[VR1] 145: OUT &H3C7, i
[142] 146: r1 = INP(&H3C9)
[B32] 147: g1 = INP(&H3C9)
[132] 148: b1 = INP(&H3C9)
[EW4] 149: OUT &H3C7, i - StartFarbe + Farbe
[442] 150: r2 = INP(&H3C9)
[E32] 151: g2 = INP(&H3C9)
[432] 152: b2 = INP(&H3C9)
[XR1] 153: OUT &H3C8, i
[CV1] 154: OUT &H3C9, r2
[1V1] 155: OUT &H3C9, g2
[WU1] 156: OUT &H3C9, b2
[GW4] 157: OUT &H3C8, i - StartFarbe + Farbe
[AV1] 158: OUT &H3C9, r1
[ZU1] 159: OUT &H3C9, g1
[UU1] 160: OUT &H3C9, b1
[ZS0] 161: NEXT
[651] 162: RETURN
[VA4] 163: '-----
[CA6] 164: 'Routine zum AbdunkeLn des markierten
[UU5] 165: 'PaLettenbereichs. Bei jedem Aufruf
[2N6] 166: 'werden die Farbwerte um 1% abgedunkeLt.
[MP1] 167: AbdunkeLn:
[TT4] 168: FOR i = StartFarbe TO EndFarbe
[VR1] 169: OUT &H3C7, i
[W02] 170: r = INP(&H3C9)
[602] 171: g = INP(&H3C9)
[WZ1] 172: b = INP(&H3C9)
[XR1] 173: OUT &H3C8, i
[292] 174: OUT &H3C9, .9 * r
[C82] 175: OUT &H3C9, .9 * g
[282] 176: OUT &H3C9, .9 * b
[ZS0] 177: NEXT
[651] 178: RETURN
[VA4] 179: '-----
[2B6] 180: 'Routine zum AufheLlen des markierten
[UU5] 181: 'PaLettenbereichs. Bei jedem Aufruf
[HF6] 182: 'werden die Farbwerte um 1% aufgeheLLt.
[ES1] 183: AufheLlen:
[TT4] 184: FOR i = StartFarbe TO EndFarbe
[VR1] 185: OUT &H3C7, i
[W02] 186: r = INP(&H3C9)
[602] 187: g = INP(&H3C9)
[WZ1] 188: b = INP(&H3C9)
[XR1] 189: OUT &H3C8, i
[N72] 190: OUT &H3C9, r / .9
[A72] 191: OUT &H3C9, g / .9
[572] 192: OUT &H3C9, b / .9
[ZS0] 193: NEXT
```

```
[651] 194: RETURN
[VA4] 195: '-----
[PP6] 196: 'Routine vermindert den Rot-Anteil der
[HB5] 197: 'Farben im angegebenen Bereich bei
[FB4] 198: 'jedem Aufruf um den Wert 1.
[L71] 199: DecRed:
[TT4] 200: FOR i = StartFarbe TO EndFarbe
[8G3] 201: 'PaLette durchLaufen
[VR1] 202: OUT &H3C7, i
[RA6] 203: 'Akt. Farbwert in PaL-Read-Register 3C7
[3A5] 204: 'AnschLießend aus Register 3C9
[9U5] 205: 'nacheinander die R, G und B Werte des
[TA4] 206: 'Farbregisters einLesen.
[W02] 207: r = INP(&H3C9)
[602] 208: g = INP(&H3C9)
[WZ1] 209: b = INP(&H3C9)
[ZF2] 210: IF r > 0 THEN r=r-1
[GH6] 211: 'Akt. Farbwert in PaL-Write-Register 3C8
[XR1] 212: OUT &H3C8, i
[176] 213: 'Die R, G und B Werte zurückschreiben
[8S1] 214: OUT &H3C9, r
[XR1] 215: OUT &H3C9, g
[SR1] 216: OUT &H3C9, b
[ZS0] 217: NEXT
[651] 218: RETURN
[VA4] 219: '-----
[4A7] 220: 'Routine vermindert den Grün-Anteil der
[HB5] 221: 'Farben im angegebenen Bereich bei
[FB4] 222: 'jedem Aufruf um den Wert 1.
[LF1] 223: DecGreen:
[TT4] 224: FOR i = StartFarbe TO EndFarbe
[VR1] 225: OUT &H3C7, i
[W02] 226: r = INP(&H3C9)
[602] 227: g = INP(&H3C9)
[WZ1] 228: b = INP(&H3C9)
[223] 229: IF g > 0 THEN g = g - 1
[8K1] 230: OUT 968, i
[7L1] 231: OUT 969, r
[6K1] 232: OUT 969, g
[RH1] 233: OUT 969, b
[ZS0] 234: NEXT
[651] 235: RETURN
[VA4] 236: '-----
[BY6] 237: 'Routine vermindert den BLau-Anteil der
[HB5] 238: 'Farben im angegebenen Bereich bei
[FB4] 239: 'jedem Aufruf um den Wert 1.
[AC1] 240: DecBlue:
[TT4] 241: FOR i = StartFarbe TO EndFarbe
[VR1] 242: OUT &H3C7, i
[W02] 243: r = INP(&H3C9)
[602] 244: g = INP(&H3C9)
[WZ1] 245: b = INP(&H3C9)
[N03] 246: IF b > 0 THEN b = b - 1
[8K1] 247: OUT 968, i
[7L1] 248: OUT 969, r
[6K1] 249: OUT 969, g
[RH1] 250: OUT 969, b
[ZS0] 251: NEXT
[651] 252: RETURN
[VA4] 253: '-----
[636] 254: 'Routine erhöht den Rot-Anteil der
[HB5] 255: 'Farben im angegebenen Bereich bei
[FB4] 256: 'jedem Aufruf um den Wert 1.
[S81] 257: IncRed:
[TT4] 258: FOR i = StartFarbe TO EndFarbe
[VR1] 259: OUT &H3C7, i
[W02] 260: r = INP(&H3C9)
[602] 261: g = INP(&H3C9)
[WZ1] 262: b = INP(&H3C9)
[ZX2] 263: IF r < 63 THEN r = r + 1
[8K1] 264: OUT 968, i
[7L1] 265: OUT 969, r
[6K1] 266: OUT 969, g
[RH1] 267: OUT 969, b
[ZS0] 268: NEXT
[651] 269: RETURN
[VA4] 270: '-----
[MP6] 271: 'Routine erhöht den Grün-Anteil der
[HB5] 272: 'Farben im angegebenen Bereich bei
[FB4] 273: 'jedem Aufruf um den Wert 1.
[SG1] 274: IncGreen:
[TT4] 275: FOR i = StartFarbe TO EndFarbe
[VR1] 276: OUT &H3C7, i
[W02] 277: r = INP(&H3C9)
[602] 278: g = INP(&H3C9)
[WZ1] 279: b = INP(&H3C9)
[YW2] 280: IF g < 63 THEN g = g + 1
[8K1] 281: OUT 968, i
[7L1] 282: OUT 969, r
[6K1] 283: OUT 969, g
[RH1] 284: OUT 969, b
[ZS0] 285: NEXT
[651] 286: RETURN
[VA4] 287: '-----
[VA6] 288: 'Routine erhöht den BLau-Anteil der
```

»palette.bas« ist ein Paletten-Editor, mit dem Sie die aktuelle Belegung der Farbtabelle verändern können.


```
[HB5] 289: 'Farben im angegebenen Bereich bei
[FB4] 290: 'jedem Aufruf um den Wert 1.
[FD1] 291: IncBLUE:
[TT4] 292: FOR i = StartFarbe TO EndFarbe
[VR1] 293: OUT &H3C7, i
[W02] 294: r = INP(&H3C9)
[G02] 295: g = INP(&H3C9)
[W21] 296: b = INP(&H3C9)
[FW2] 297: IF b < 63 THEN b = b + 1
[8K1] 298: OUT 968, i
[7L1] 299: OUT 969, r
[6K1] 300: OUT 969, g
[RH1] 301: OUT 969, b
[ZS0] 302: NEXT
[651] 303: RETURN
[VA4] 304: '-----
[NF5] 305: 'Routine speichert den aktuelleLlen
[VF6] 306: 'Inhalt der Farbpalette im Array PaL().
[E52] 307: PaletteKeep:
[592] 308: FOR i = 0 TO 255
[VR1] 309: OUT &H3C7, i
[B23] 310: PaL(i, 0) = INP(&H3C9)
[C23] 311: PaL(i, 1) = INP(&H3C9)
[D23] 312: PaL(i, 2) = INP(&H3C9)
[ZS0] 313: NEXT
[651] 314: RETURN
[VA4] 315: '-----
[AB6] 316: 'Routine aktuaLLisiert die Palette mit
[FH4] 317: 'den Werten des Array PaL().
[ZU2] 318: PaletteRestore:
[592] 319: FOR i = 0 TO 255
[8K1] 320: OUT 968, i
[7M2] 321: OUT 969, PaL(i, 0)
[9M2] 322: OUT 969, PaL(i, 1)
[BM2] 323: OUT 969, PaL(i, 2)
[ZS0] 324: NEXT
[651] 325: RETURN
[VA4] 326: '-----
[T01] 327: Hilfe:
[152] 328: DO UNTIL i$ = ""
[YL1] 329: i$ = INKEY$
[HS0] 330: LOOP
[VD6] 331: 'Bildschirm in Datei "backup2.$$$" sichern
[W72] 332: DEF SEG = &H4000
[9P3] 333: BSAVE "backup2.$$$", 0, 64000
[091] 334: DEF SEG
[PN0] 335: CLS
[F71] 336: COLOR 15
[4C6] 337: PRINT "PALETTEEDITOR-TASTENBELEGUNG"
[GX0] 338: PRINT
[0D4] 339: PRINT "F1 - Diese Hilfeseite"
[VC6] 340: PRINT "F2 - Backup der aktuellen Palette"
[ND6] 341: PRINT "F3 - Palette von letztem Backup"
[3R4] 342: PRINT "F4 - Palette speichern"
[YY3] 343: PRINT "F5 - Palette Laden"
[K25] 344: PRINT "s - Bereichstart setzen"
[2L4] 345: PRINT "e - Bereichende setzen"
[BA4] 346: PRINT "+ - Bereich aufheLlen"
[3B4] 347: PRINT "- - Bereich abdunkeln"
[YY5] 348: PRINT "t - tausche markierten Bereich"
[H56] 349: PRINT "m - Standard BASIC + Graustufen"
[855] 350: PRINT "r / SHIFT + r - Rot-Anteil -/+"
[EP5] 351: PRINT "g / SHIFT + g - Grün-Anteil -/+"
[4E5] 352: PRINT "b / SHIFT + b - Blau-Anteil -/+"
[KB6] 353: PRINT "ESC - PaLitteneditor verlassen"
[N92] 354: DO UNTIL i$ <> ""
[YL1] 355: i$ = INKEY$
[HS0] 356: LOOP
[W94] 357: 'Bildschirm restaurieren
[W72] 358: DEF SEG = &H4000
[KE5] 359: 'Segment auf Beginn VGA-Speicher
[DX2] 360: BLOAD "backup2.$$$", 0
[091] 361: DEF SEG
[XK5] 362: 'Segment auf BASIC zurücksetzen
[651] 363: RETURN
[VA4] 364: '-----
[6E5] 365: 'Speichern der Farbpalette in eine
[F41] 366: 'Datei.
[213] 367: PaletteSpeichern:
[GZ2] 368: GOSUB PaletteKeep
[GM1] 369: LOCATE 20, 1
[PB5] 370: INPUT "Dateiname (SAVE)"; PaLette$
[906] 371: 'Nachdem der Segment-Zeiger auf den
[824] 372: 'Beginn des VGA-Bildspeichers
[UP5] 373: 'gesetzt ist, werden die R, G und B-
[Y85] 374: 'Werte jeder PaLettenfarbe in den
[XK5] 375: 'Beginn des Bildschirmspeichers
[5T5] 376: 'geschrieben (POKE). Anschließend
[5Y4] 377: 'wird dieser Bereich mit BSAVE
[2X1] 378: 'gesichert.
[W72] 379: DEF SEG = &H4000
[ZP2] 380: FOR pLt = 0 TO 255
[4D2] 381: pLtptr = pLt * 3
[RX3] 382: POKE (pLtptr), PaL(pLt, 0)
[B94] 383: POKE (pLtptr + 1), PaL(pLt, 1)
[E94] 384: POKE (pLtptr + 2), PaL(pLt, 2)
[ZS0] 385: NEXT
[263] 386: BSAVE PaLette$, 0, 768
[W94] 387: 'Bildschirm restaurieren
```

```
[ZS2] 388: BLOAD "backup.$$$", 0
[M07] 389: DEF SEG 'Segment auf BASIC zurücksetzen
[UW3] 390: SOUND 800, .3: SOUND 1000, .3
[4W1] 391: GOTO Start
[VA4] 392: '-----
[1G7] 393: 'Unterprogramm Lädt angegebene Farbpalette.
[6X6] 394: 'Welche Farbwerte der zuverigen Palette
[6W6] 395: 'geändert werden, hängt vom definierten
[F84] 396: 'Bereich (Tasten s und e) ab.
[082] 397: PaletteLaden:
[GM1] 398: LOCATE 20, 1
[V95] 399: INPUT "Dateiname (LOAD)"; PaLette$
[ZU5] 400: INPUT "BereichStart"; BereichStart
[845] 401: INPUT "BereichEnde"; BereichEnde
[F56] 402: IF BereichEnde = 0 THEN BereichEnde = 255
[9B7] 403: INPUT "ZielBereichStart"; ZielBereichStart
[W72] 404: DEF SEG = &H4000
[ZL2] 405: BLOAD PaLette$, 0
[E76] 406: FOR pLt = BereichStart TO BereichEnde
[LR5] 407: pLtptr = ZielBereichStart + pLt * 3
[RN2] 408: r = PEEK(pLtptr)
[V13] 409: g = PEEK(pLtptr + 1)
[N13] 410: b = PEEK(pLtptr + 2)
[1Z1] 411: OUT 968, pLt
[7L1] 412: OUT 969, r
[6K1] 413: OUT 969, g
[RH1] 414: OUT 969, b
[ZS0] 415: NEXT
[GZ2] 416: GOSUB PaletteKeep
[W94] 417: 'Bildschirm restaurieren
[W72] 418: DEF SEG = &H4000
[ZS2] 419: BLOAD "backup.$$$", 0
[787] 420: DEF SEG 'Segment auf BASIC zurücksetzen
[UW3] 421: SOUND 800, .3: SOUND 1000, .3
[4W1] 422: GOTO Start
[VA4] 423: '-----
[9Y6] 424: 'Das Unterprogramm Machestandard setzt in
[SC6] 425: 'der aktuellen Palette einige Standard-
[DES] 426: 'farben - und zwar so: die ersten 16
[NR6] 427: 'Farben entsprechen den BASIC-Standard-
[VK6] 428: 'farben, die folgenden 32 Farben erhalten
[F23] 429: 'Graustufenwerte.
[TD2] 430: Machestandard:
[AZ0] 431: a = 968
[GZ0] 432: b = 969
[KB5] 433: OUT a, 0: OUT b, 0: OUT b, 0: OUT b, 0
[7K5] 434: OUT a, 1: OUT b, 0: OUT b, 0: OUT b, 42
[8E5] 435: OUT a, 2: OUT b, 0: OUT b, 42: OUT b, 0
[BG5] 436: OUT a, 3: OUT b, 0: OUT b, 42: OUT b, 42
[ED5] 437: OUT a, 4: OUT b, 42: OUT b, 0: OUT b, 0
[HF5] 438: OUT a, 5: OUT b, 42: OUT b, 0: OUT b, 42
[RN5] 439: OUT a, 6: OUT b, 42: OUT b, 21: OUT b, 0
[7T5] 440: OUT a, 7: OUT b, 42: OUT b, 42: OUT b, 42
[LS5] 441: OUT a, 8: OUT b, 21: OUT b, 21: OUT b, 21
[WS5] 442: OUT a, 9: OUT b, 21: OUT b, 21: OUT b, 63
[5R5] 443: OUT a, 10: OUT b, 21: OUT b, 63: OUT b, 21
[NR5] 444: OUT a, 11: OUT b, 21: OUT b, 63: OUT b, 63
[7R5] 445: OUT a, 12: OUT b, 63: OUT b, 21: OUT b, 21
[RR5] 446: OUT a, 13: OUT b, 63: OUT b, 21: OUT b, 63
[ZR5] 447: OUT a, 14: OUT b, 63: OUT b, 63: OUT b, 21
[FS5] 448: OUT a, 15: OUT b, 63: OUT b, 63: OUT b, 63
[F95] 449: OUT a, 16: OUT b, 0: OUT b, 0: OUT b, 0
[5A5] 450: OUT a, 17: OUT b, 2: OUT b, 2: OUT b, 2
[VA5] 451: OUT a, 18: OUT b, 4: OUT b, 4: OUT b, 4
[HB5] 452: OUT a, 19: OUT b, 6: OUT b, 6: OUT b, 6
[MB5] 453: OUT a, 20: OUT b, 8: OUT b, 8: OUT b, 8
[3P5] 454: OUT a, 21: OUT b, 10: OUT b, 10: OUT b, 10
[NP5] 455: OUT a, 22: OUT b, 12: OUT b, 12: OUT b, 12
[7R5] 456: OUT a, 23: OUT b, 14: OUT b, 14: OUT b, 14
[TR5] 457: OUT a, 24: OUT b, 16: OUT b, 16: OUT b, 16
[BS5] 458: OUT a, 25: OUT b, 18: OUT b, 18: OUT b, 18
[PP5] 459: OUT a, 26: OUT b, 20: OUT b, 20: OUT b, 20
[8R5] 460: OUT a, 27: OUT b, 22: OUT b, 22: OUT b, 22
[UR5] 461: OUT a, 28: OUT b, 24: OUT b, 24: OUT b, 24
[CS5] 462: OUT a, 29: OUT b, 26: OUT b, 26: OUT b, 26
[BS5] 463: OUT a, 30: OUT b, 28: OUT b, 28: OUT b, 28
[YP5] 464: OUT a, 31: OUT b, 30: OUT b, 30: OUT b, 30
[GR5] 465: OUT a, 32: OUT b, 32: OUT b, 32: OUT b, 32
[2S5] 466: OUT a, 33: OUT b, 34: OUT b, 34: OUT b, 34
[MS5] 467: OUT a, 34: OUT b, 36: OUT b, 36: OUT b, 36
[6T5] 468: OUT a, 35: OUT b, 38: OUT b, 38: OUT b, 38
[HR5] 469: OUT a, 36: OUT b, 40: OUT b, 40: OUT b, 40
[3S5] 470: OUT a, 37: OUT b, 42: OUT b, 42: OUT b, 42
[NS5] 471: OUT a, 38: OUT b, 44: OUT b, 44: OUT b, 44
[7T5] 472: OUT a, 39: OUT b, 46: OUT b, 46: OUT b, 46
[6T5] 473: OUT a, 40: OUT b, 48: OUT b, 48: OUT b, 48
[HR5] 474: OUT a, 41: OUT b, 50: OUT b, 50: OUT b, 50
[3S5] 475: OUT a, 42: OUT b, 52: OUT b, 52: OUT b, 52
[NS5] 476: OUT a, 43: OUT b, 54: OUT b, 54: OUT b, 54
[7T5] 477: OUT a, 44: OUT b, 56: OUT b, 56: OUT b, 56
[TT5] 478: OUT a, 45: OUT b, 58: OUT b, 58: OUT b, 58
[4S5] 479: OUT a, 46: OUT b, 60: OUT b, 60: OUT b, 60
[PS5] 480: OUT a, 47: OUT b, 62: OUT b, 62: OUT b, 62
[651] 481: RETURN
[HA1] 482: END SUB
```

»palette.bas« ist ein Paletten-Editor, mit dem Sie die aktuelle Belegung der Farbtabelle verändern können.

Noch erhältlich:

DOS EXTRA Nr. 4

- Desktop-Publishing-Programm »SDTP«
- Kommandoeditor für MS-DOS
- Editor für Disketten und Festplatten
- Speicherresidente Programme verwalten
- Passwort-Schutz für MS-DOS
- Tastatureingabe automatisiert
- Clock-Utility für alle Grafikkarten
- Cursor und Zeichensatz für EGA-Karte
- Fenstertechnik für Turbo C

DOS EXTRA Nr. 5

- 16 Farben für die CGA-Farbgrafikkarte
- Druckerzeichensatz selbstgemacht
- Mausgesteuertes Zeichenprogramm
- Editor in GW-Basic
- Setup-Programm für ATs
- Maschinsprache für GW-Basic

DOS EXTRA Nr. 6

- SBasic-Compiler im Quelltext
- Datenbankprogramm »Parat«
- Fensterfunktionen für GW-Basic und SBasic
- Analyse von Grafikkarten
- Maskengenerator für Binärdateien
- dBase-Datenbanken in Basic
- Superhirn für Quick Basic 4.0

DOS EXTRA Nr. 7

- MS-DOS-Benutzeroberfläche Easydos
- Benutzeroberfläche für SBasic
- Sprite-Editor für Farbgrafikkarten
- DOS-Packer
- Virenschutzprogramm
- Weltkarte im Computer
- Einfacher Prozessor-Emulator

DOS EXTRA Nr. 8

- Makroassembler SAM III
- Exe2bin-Ersatz durch »exe2com«
- Speicherverwaltung unter MS-DOS

DOS EXTRA Nr. 10

- Units-Bibliotheken für Turbo Pascal: Datum, Sound, Bildschirmausgabe
- Snapshot-Utility, Version 2.7
- TSR-Datei-Manager
- DOS-Assembler SAM III, Version 1.1
- Schnelles Kopierprogramm
- Komfortable Dateiumleitung
- Hercules-Grafik in Basic programmiert
- Patch für BGI-Treiber von Turbo Pascal

DOS EXTRA Nr. 11

- Preiswerte HPGL-Erweiterung, der HPGL-Extender
- Leistungsstarker Disketteneditor
- Blockbefehle für den Turbo-Pascal-Editor
- Sound-Erzeugung unter Turbo Pascal
- Speicherresidente Uhr

DOS EXTRA Nr. 12

- Kreative Grafik in Basic zum Mitmachen
- Grafik-Anpassung von Basic-Programmen
- Außergewöhnliche Fraktalgrafiken
- Das ultimative Hardcopy-Utility
- Testbild für Monitore in Basic programmiert

DOS EXTRA Nr. 14

- Komfortables Datensicherungsprogramm
- Zeichensatzeditor für 24-Nadel-Drucker
- Pascal-Quelltexte optimieren
- Utilities zur Bearbeitung von Dateien und Zeichnissen

DOS EXTRA Nr. 15

- Der Nachbrenner für GW-Basic: SBasic Version 2.0
- Der Tastatortreiber »Skeybgr« in der Version 4.0
- »Schiffe versenken« mit einem Quick-Basic-Programm
- Simulation von Tastatureingaben mit »Prokey«

DOS EXTRA Nr. 16

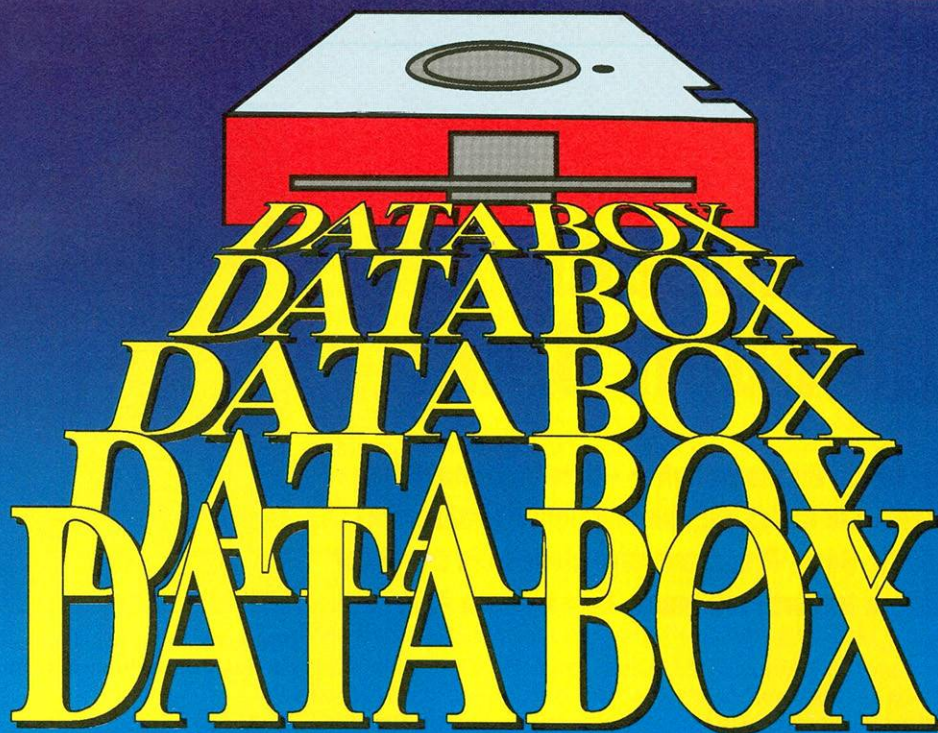
- Batch-Compiler der S-Klasse
- Config-Editor beim Booten
- Assembler-Preprozessor für MASM und TASM
- GEM-API-Unit für Turbo Pascal
- SCheck-Prüfsummenprogramm in der Version 3.0

DOS EXTRA Nr. 17

- Transpiler: Basic nach C
- Flexibles Kopierprogramm für alle Formate
- Formatierer der S-Klasse
- BGI-Treiber für Super-VGA
- Kiloweise Hilfsprogramme für jeden Zweck

DOS EXTRA Nr. 18

- universelle Entwickleroberfläche
- PCX-Bilder in Basic
- Pretty Printer für Turbo Pascal
- Tips, Tricks und Tools
- kiloweise Hilfsprogramme für jeden Zweck



Die Software zu den DOS-EXTRA-Heften

Sagenhaft günstig, extra viel gute Software fürs Geld!

DATABOX:

- mehr als der übliche Softwareservice
- bringt alle Programme der DOS EXTRA auf Datenträger

Für alle PCs unter MS-DOS

Den Aufbau der Databox-Diskette finden Sie in der Datei »read.me« erklärt. Geben Sie deshalb vor dem Start »type read.me« ein. Sämtliche Pascal- und C-Programme liegen auch in einer kompilierten Form für Hercules- und Farbgrafikkarten vor. Es ist allerdings nicht gewährleistet, daß diese Versionen auf Hardware, die nicht vollständig kompatibel zum IBM PC ist, problemlos laufen. In diesem Fall müssen Sie den Quellcode nach der Anpassung mit dem entsprechenden Compiler neu übersetzen.

jeweils 2 Disketten 5 1/4"
1 Diskette 3 1/2"

35,- DM (unverbindliche Preisempfehlung)

Wenn Sie über den DMV-Bestellservice bestellen, gilt folgendes:

Inland:		Ausland:	
Einzelpreis	35,- DM	Einzelpreis	35,- DM
zzgl. Versandkosten	4,- DM	zzgl. Versandkosten	6,- DM
Endpreis	39,- DM	Endpreis	41,- DM

Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland 4,- DM bzw. für das Ausland 6,- DM Porto und Verpackung.

- Bitte benutzen Sie die Bestellkarte. -



DOS EXTRA Nr. 19

- komplettes Raytracing-Programm
- Sprite-Programmierung
- Fraktal-Explorer
- Units zur Darstellung von 2D- und 3D-Grafiken
- Tips & Tricks zur Grafik

Der Film am laufenden Band

Sprites, Fonts, Bildausschnitte – diese Grundbilder entwickeln Sie meist mit kommerziellen Grafikwerkzeugen. Als Basic-Programmierer stehen Sie damit vor erheblichen Import-Problemen. Hier hilft ein Auto-Grabber.

Zu Beginn der Werkstatt wurde bereits erwähnt: Es hat keinen allzu großen Sinn, einen eigenen Sprite-Editor oder gar ein Malprogramm zu entwickeln, wenn Sie lediglich Bilder in eigenen Programmen verwenden möchten. Eine einfache Methode zum Importieren beliebiger Bilder durch Einsatz eines externen Viewers wurde deshalb bereits vorgestellt. Nachdem Sie ein Bild per Viewer-Aufruf über die Shell-Anweisung auf dem Bildschirm haben, läßt es sich problemlos mit der Grafikanweisung »GET« einfangen und speichern. Wer allerdings mit vielen Bildern und Bildausschnitten arbeitet, wird es lästig finden, bei der Get-Ausführung die Bildkoordinaten, Breite und Höhe stets manuell zu ermitteln. Im folgenden lernen Sie ein komfortables Werkzeug kennen, das Bild-Import zum Kinderspiel werden läßt und zudem einige Spezial-Funktionen bereitstellt. Wer ein umfangreiches Programmierprojekt plant, kommt selten um die Programmierung individueller Tools herum, mit denen sich die Gesamtentwicklung erleichtern läßt. Nichts ist lästiger, als bei der Programmmentwicklung immer wieder durch manuelle Standardarbeiten aufgehalten zu werden.

Bild 1 verdeutlicht das Problem, das auf den folgenden Seiten gelöst wird. Zu sehen ist ein Grafikbildschirm, auf dem sich sechs grafische Objekte befinden – ein Bild der Erde und diverse Gebirgstteile. Die blauen Rahmen verdeutlichen lediglich, wie sich die Objekte mit einem Rahmen eingrenzen lassen. Die Grafiken wurden mit einem Zeichenprogramm (Colorix) für ein Spiel angefertigt, bei dem der PC aus fotorealistischen Einzelteilen künstliche Landschaften montiert. Das Problem: Wie konvertiert man dieses Bild möglichst schnell in Basic-einlesbare Grafiken? Natürlich müssen die sechs Objekte in einzelnen Grafikdateien gespeichert werden, die Sie später über »BLOAD« einlesen und mit »PUT« auf den Bildschirm bringen können. Eine einfache Lösung ist die Programmierung eines per Maus oder Pfeiltasten steuerbaren Fadenkreuzes, mit dem Sie Ausschnitte aus dem Gesamtbildschirm markieren und per Tastendruck speichern können – besser, als wenn Sie stets mit Koordinaten-Daten für »GET« herumexperimentieren müßten. Wer eine solche Markierungs-Routine programmiert, wird bei der Anwendung der Routine folgendes feststellen: der Ablauf des Markierens der einzelnen Bildelemente erfolgt stets nach dem gleichen Schema: linken oberen Eckpunkt setzen, dann rechten unteren und schließlich den so definierten rechteckigen Bereich speichern. Haben Sie diese Automatik erkannt, so ist auch schon klar, daß sich eine weitaus komfortablere Lösung programmieren läßt: etwas, das stets nach dem gleichen Schema manuell ausgeführt wird, kann der PC auch automatisch erledigen. Die Zielsetzung ist damit klar: ein Tool muß her, das automa-

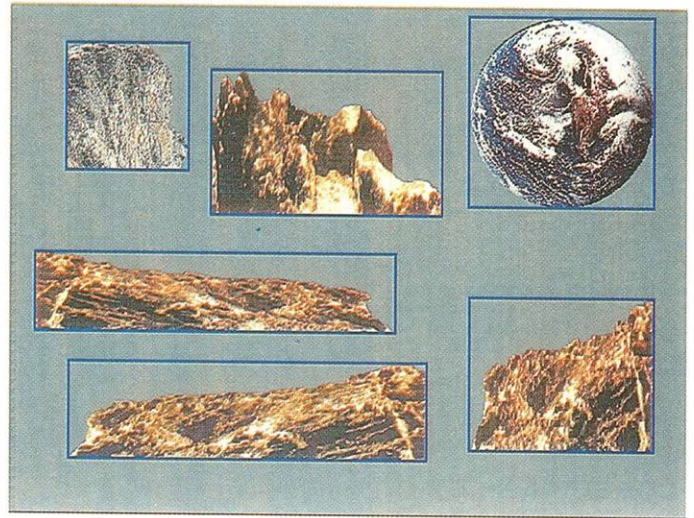


Bild 1. Sechs Grafikobjekte – sie werden vom Auto-Grabber in ein Basic-verträgliches Format konvertiert.

tisch alle Objekte auf einem Bildschirm erkennt und sie einzeln speichert. Wie soll der Rechner aber herausfinden, wie der rechteckige Rahmen um ein beliebiges Bildteil zu setzen ist? Wie soll er nicht-rechteckige Objekte einfangen? Damit Sie dieses Problem lösen können, sollten Sie sich erst einen Plan für die Vorgehensweise erarbeiten:

- Damit ein Bildteil als solcher erkannt werden kann, muß er auf einem einfarbigen Hintergrund plziert sein. Nur so läßt sich automatisch feststellen, wie seine Umrisse verlaufen. Für den Auto-Grabber wird folgendes festgelegt: Der Hintergrund, auf dem die Einzelobjekte plziert sind, muß die Farbe 0 haben – also die erste Farbe in der Palette.
- Wo soll der Rechner mit der Suche beginnen? Als idealer Startpunkt für die Suche stellt sich schnell der erste Bildpunkt des Bildschirms oben links (Koordinaten 0,0) heraus. Schreiben Sie eine Routine, die beim Startpunkt beginnt und das Bild Punkt für Punkt von links nach rechts und Zeile für Zeile abtastet, so wird die Routine früher oder später auf einen Bildpunkt stoßen, der nicht der Hintergrundfarbe 0 entspricht. Die Routine hat damit einen Bildpunkt eines Grafikobjekts erwischt, ein erster Anhaltspunkt ist gefunden. Bild 2 verdeutlicht den Sachverhalt, anhand der stark vergrößerten Darstellung eines Einzelobjekts aus Bild 1. Ziel des Scan-Vorgangs ist, einen rechteckigen Rahmen zu ermitteln, der das Grafikobjekt exakt einschließt. Beim Zeile-für-Zeile-Abtasten zum Finden des Startpunkts ist sicherge-

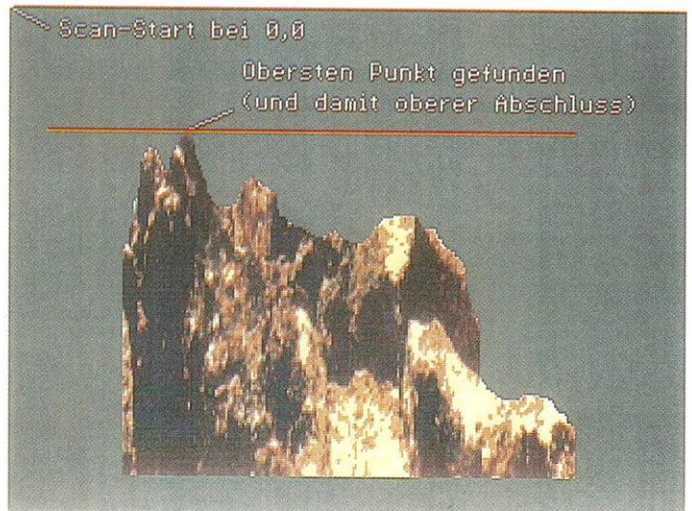


Bild 2. Schritt 1 des Scan-Vorgangs – der oberste Startpunkt wird ermittelt.

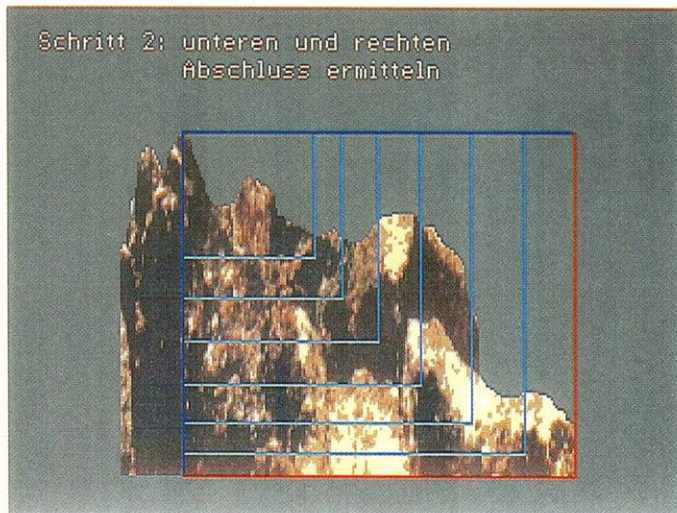


Bild 3. Schritt 2 des Scan-Vorgangs – das Rechteck wird vergrößert, bis der rechte und der untere Abschluß gefunden sind.

stellt, daß der Startpunkt dem obersten Punkt des Grafikobjekts entspricht. Der obere horizontale Abschluß (veranschaulicht durch die rote Linie) ist damit gefunden.

● Wie geht es jetzt aber weiter? Bekannt ist, daß das Grafikobjekt komplett erfaßt ist, wenn alle vier Seiten des Rahmens das Objekt exakt eingrenzen. Es müssen also einige Bedingungen erfüllt sein:

- Die oberste horizontale Rahmenlinie ist dann korrekt, wenn sich in ihr mindestens ein Bildpunkt ungleich der Hintergrundfarbe befindet und die unmittelbare Zeile über ihr ausschließlich Bildpunkte in der Hintergrundfarbe (Farbe 0) enthält.
- Bei der unteren horizontalen Rahmenlinie sieht es ähnlich aus. Sie ist dann perfekt, wenn sie mindestens einen Bildpunkt ungleich der Hintergrundfarbe enthält und die darunter liegende Zeile ausschließlich aus Bildpunkten der Hintergrundfarbe besteht.
- Was oben und unten gilt, läßt sich auch auf den rechten und linken Abschluß übertragen. Beim linken Abschluß muß die vertikale Linie links neben der Rahmenseite ausschließlich aus Bildpunkten der Hintergrundfarbe bestehen, bei der rechten Rahmenseite entsprechend die rechts daneben verlaufende vertikale Linie.

Eines ist noch zu beachten: damit das Zeile-für-Zeile-Suchen des Startpunkts nicht immer an der gleichen Stelle hängenbleibt (erstes Grafikobjekt), muß jedes Objekt, nachdem es eingefangen und gespeichert ist, vom Bildschirm gelöscht werden. Zum Löschen wird einfach ein ausgefülltes Rechteck in der Hintergrundfarbe darübergezeichnet. Findet die Startpunkt-Suchroutine keinen einzigen Bildpunkt ungleich der Hintergrundfarbe mehr, so ist der Bildschirm leereräumt, der Scan-Vorgang wird beendet.

Primäre Scanroutinen

Genug der Theorie, die Programm-Routinen-Entwicklung kann beginnen. Zuerst schreibt man die primären Scan-Test-Routinen: eine findet den vertikalen, eine andere den horizontalen Rahmenabschluß. Sind die Routinen fertig, können sie in die Hauptscanroutine eingebunden werden. Deren Entwicklung erfordert wiederum einige Überlegungen, da folgendes Problem besteht: Es ist nicht möglich, beim horizontalen Test die gesamte Bildschirmbreite beziehungsweise beim vertikalen Test die gesamte Bildschirmhöhe zu berücksichtigen. Das ist nur dann legitim, wenn sich am Bildschirm nur ein einziges Objekt befindet, das gefunden werden soll. Eine perfekte und vor allem komfortable Scan-Routine muß allerdings beliebig viele Einzelobjekte

zulassen. Die Lösung ist ganz einfach: Betrachten Sie Bild 3 – Sie sehen, wie es weitergeht, wenn der Startpunkt durch das Zeile-für-Zeile-Abtasten gefunden ist. Die durch hellblaue Linien dargestellten rechteckigen Bereiche sind willkürlich gesetzt – sie sollen lediglich den Ablauf anschaulicher machen. Die Idee ist, vom obersten Startpunkt aus ein Rechteck so lange zu vergrößern, bis der rechte untere Abschluß des Grafikobjekts gefunden ist. Die obere linke Ecke des Scan-Rechtecks entspricht natürlich dem Startpunkt. Als Scan-Ausgangspunkt wählt man den Punkt, der sich exakt ein Pixel rechts vom Startpunkt und ein Pixel darunter befindet – das ist die Mindestgröße für den Rahmen. Anschließend werden per Schleife die x- und y-Koordinaten des unteren rechten Rechteck-Punktes so lange um den Wert 1 erhöht, bis die vertikale und horizontale Scan-Routine folgendes feststellen: Die rechte vertikale Rechteckseite und die untere horizontale Rechteckseite erfüllen die oben beschriebenen Abschluß-Bedingungen. Die beiden roten Linien in Bild 3 sind entsprechend jene, die beim Vergrößern des Rechtecks ständig von den Routinen gecheckt werden.

Jetzt kann der dritte Scan-Schritt beginnen, bei dem der linke Abschluß gefunden werden muß. Hierbei wird die linke Rechteckseite so lange um 1 nach links verschoben, bis sie ebenfalls die Abschlußbedingung erfüllt (Bild 4). Allerdings reicht es beim Nach-links-Erweitern keinesfalls aus, nur den senkrechten Rechteckabschluß zu prüfen. Das verdeutlicht auch Bild 5: Es kann sein, daß der untere Abschluß noch nicht stimmt. Würden Sie bei Bild 5 den vertikalen Abschluß lediglich nach links verschieben, so würde die Routine das nach unten herausstehende Gebirgsteil schlichtweg abschneiden. Der dritte Scan-Schritt muß also ähnlich wie der zweite ablaufen. Einziger Unterschied: nicht der rechte untere Rechteckpunkt ist entscheidend, sondern der linke untere. Anstatt also nur die linke Rahmen-seite zu verschieben, wird gleichzeitig auch nach unten verschoben.

Bilder, Frames, Sprites und Basic

Somit sind die Grundlagen für ein Scan-Tool geschaffen, das dem Grafikprogrammierer die Entwicklung enorm erleichtert. In Bild 6 sehen Sie, in welcher Reihenfolge der Auto-Grabber Bildobjekte einfängt. Die Reihenfolge ergibt sich zwangsläufig aus der Positionierung der Einzelobjekte. Da die Erdkugel bei der Zeile-für-Zeile-Startpunktsuche zuerst erwischt wird, ist sie natürlich auch das erste eingefangene Objekt. Beim Einsatz des Auto-Grabbers ist noch eines zu beachten: Der Bildschirmrand ist bei der Scan-Abtastung

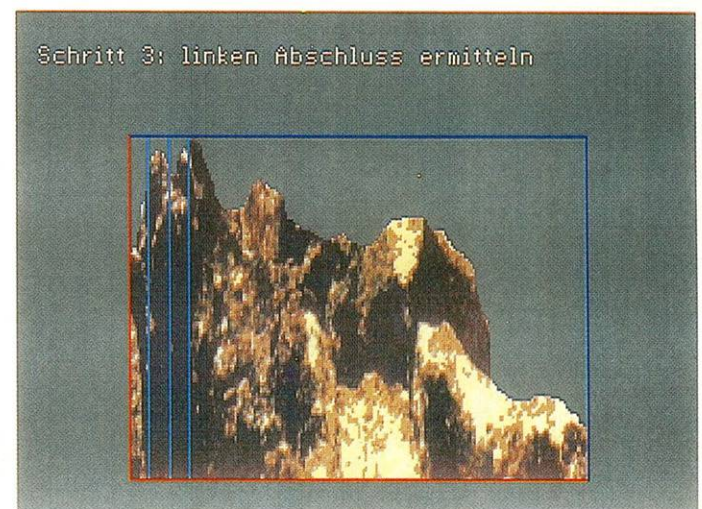


Bild 4. Schritt 3 des Scan-Vorgangs – der linke Abschluß wird ermittelt.

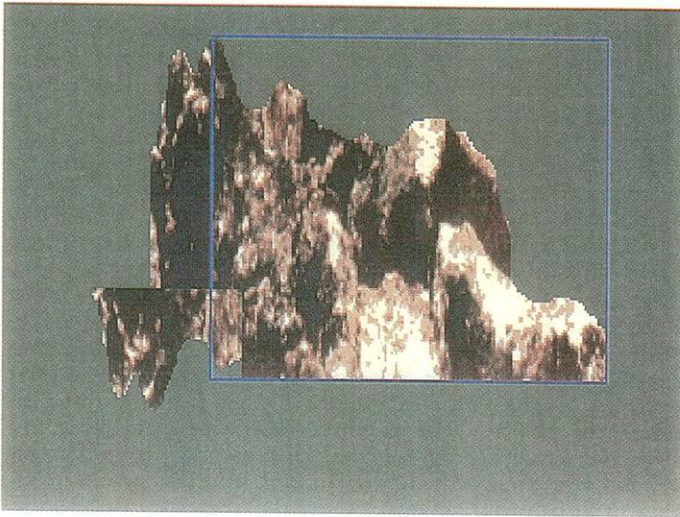


Bild 5. Beim Suchen des linken Abschlusses muß auch der untere horizontale Abschluß berücksichtigt werden.

separat zu berücksichtigen. Stößt ein Bildausschnitt am Bildrand an, so kann daneben natürlich keine Zeile mit freien Punkten in der Hintergrundfarbe bestehen.

Mit dem Auto-Grabber sind recht trickreiche Dinge anzustellen. Die verschiedenen Methoden, Einzelbilder oder Animations-Sequenzen in Arrays einzulesen, wurden im Workshop bereits eingehend beschrieben. Natürlich eignet sich der Auto-Grabber auch prima dazu, auch ganze Bildsequenzen als Datei zu speichern. Die oben beschriebene Scan-Routine ist gut zum Einfangen von Einzelbildteilen geeignet, für das Anfertigen von Sprite-Sequenzen ist sie allerdings nicht sonderlich brauchbar. Stellen Sie sich an dieser Stelle vor, Sie möchten eine Erdkugel ähnlich wie einen Springball eine Linie am Bildschirm entlanghüpfen lassen. Während die Kugel sich nach rechts bewegt, muß sie natürlich hüpfen. Das Hüpfen soll durch verschiedene Einzelbilder realisiert werden. Bild 7 zeigt den dafür erforderlichen Entwurf mit einem Zeichenprogramm. Die blaue Linie dient nur als Hilfe bei der Konstruktion. Sie steht quasi stellvertretend für eine beliebige Bahn, auf der die Erdkugel später hüpfen soll. Das Hüpfen wird durch verschiedenen Abstand der Erdkugel vom Boden erreicht. Bei Bild 7 besteht die Sprungbahn der Erdkugel aus sechs Phasen beziehungsweise Einzelbildern. Beim Ablauf der Animation wird von einem zum nächsten Bild (Frame) weitergeschaltet. Solche Animationen sollten natürlich mit dem Auto-Grabber anzufertigen sein. Beim Erdkugel-Beispiel versagt er allerdings. Bild 8 zeigt, was passiert, wenn er die Einzelbilder einfängt: sie werden schlichtweg in der falschen Reihenfolge eingesammelt. Zudem zieht die Scan-Routine um jede Erdkugel ein exaktes Rechteck – damit sind praktisch alle Einzelbilder identisch, das Bewegungs-Moment (Hüpfen) wird beim Scannen nicht berücksichtigt.

Einsammeln der Bilder

Aus diesem Grund ist im Auto-Grabber eine zweite Scan-Routine (Sprite-Scan) integriert, deren Funktionsweise sich so versteht (Bild 9): Die Suche eines Startpunkts für den Beginn des Scan-Vorgangs entfällt. Statt dessen läuft eine horizontale Scan-Zeile, beginnend in der ersten Bildschirmzeile, so lange nach unten, bis sie auf einen Bildpunkt ungleich der Hintergrundfarbe stößt. An dieser Stelle bleibt sie stehen, und in der Zeile darunter startet eine zweite Scan-Zeile, die so lange nach unten läuft, bis sie den unteren Abschluß gefunden hat. Die vertikale Eingrenzung der Objekte ist nun kein Problem mehr: Die erste vertikale Scanzeile (Länge = obere horizontale Zeile bis untere horizontale Zeile) läuft so

lange nach rechts, bis sie auf ein Objekt stößt. Dann startet in der Spalte daneben eine zweite, die den rechten Objekt-Abschluß findet. Nach diesem Verfahren werden alle Objekte von links nach rechts eingesammelt – unabhängig von ihrer Höhe. Das bringt einen weiteren Vorteil, der durch die roten Punkte in Bild 9 verdeutlicht wird: beim zweiten Scan-Verfahren werden die Einzelbilder nicht mit einem exakten Rahmen umschlossen, sondern die x-Koordinaten der linken oberen Ecken sind für alle Einzelbilder identisch. Diese Ecken können dann beim Animations-Ablauf in einem Programm für eine Hüpf-Bahn angegeben werden. Die Methode hat allerdings einen Nachteil. Die gespeicherten Bilder beanspruchen weitaus mehr Speicherplatz als notwendig. Erinnern Sie sich in diesem Zusammenhang an das Kapitel über »PUT« und »GET«, wo auf individuelle Bild-Array-Datenformate eingegangen wird. Werden mehrere Bilder in einem Array gemeinsam angelegt, so können Sie in diesem problemlos weitere Daten (Position) unterbringen – entsprechend sind Sie an dieser Stelle aufgefordert, den Auto-Grabber nach eigenen Bedürfnissen abzuändern. Sämtliche Routinen sind einfach und übersichtlich gestaltet. Für den Einstieg empfiehlt es sich, das Listing einfach abzutippen und mit dem Auto-Grabber zu experimentieren. Beachten Sie, daß Sie abhängig von den Grafikformaten, in denen Ihre Bilder vorliegen, einen geeigneten Viewer benötigen, der sich über die DOS-Shell aufrufen läßt. Im Auto-Grabber ist für GIF-Bilder der Viewer Picem und für LBM-Bilder (Deluxe Paint) der Viewer Vpic eingetragen. Beide sind Shareware-Programme und können problemlos im Shareware-Handel bezogen werden. Von Haus aus kann der Auto-Grabber lediglich Basic-BLD/PLT-Bilder einlesen.

Befehlsübersicht Auto-Grabber

Der Auto-Grabber kann in Verbindung mit einem geeigneten externen View-Programm jedes Bild mit einer Auflösung von 320 x 200 und 256 Farben bearbeiten. Die Ausgabe sämtlicher Bilddaten erfolgt im Basic-BLOAD-Dateiformat. Abhängig von der jeweiligen Einstellung, erzeugt der Auto-Grabber Dateien mit unterschiedlichen Dateieendungen. Diese Dateien können mit den Routinen der VGA-Toolbox (am Ende des Workshops) verarbeitet werden:

- **PLT** – Palettendatei. Die Palettendaten werden stets in der Reihenfolge Farbe 0 bis 255 gespeichert, mit jeweils getrennten R-, G- und B-Werten. Die PLT-Dateien lassen sich natürlich mit der Sub-Routine »PaletteLaden« des Workshops einlesen.



Bild 6. In dieser Reihenfolge werden die Objekte beim Scan-Vorgang erkannt. Entscheidend ist immer der oberste Punkt eines Objekts.

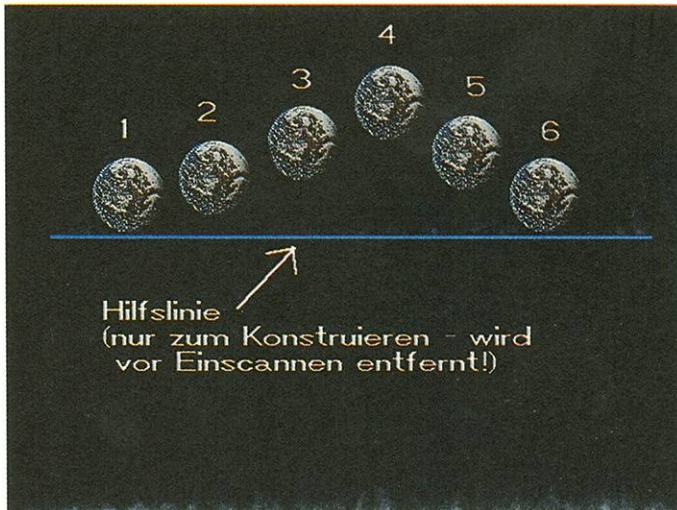


Bild 7. Entwurf für eine Animation mit hüpfender Erdkugel

- **PUT** – Bilddatendatei. Hier werden die Bilddaten im Standard-Array-Format ohne Zusatzinformationen gespeichert:

Array(1) = Breite; Array(2) = Höhe

Array(3) - Array(n) Bilddaten

- **DAT** – Bilddatendatei mit Positionsdaten. Bei diesem Format wird die Position des linken oberen Eckpunktes eines Bildes mitgespeichert. Wird eine solche Bilddatei mit einer entsprechenden Routine geladen (Sub-Routine »Pos-Clip« im Listing), so erscheint das Bild an exakt der Stelle, an der es der Auto-Grabber auch eingefangen hat.

Die Array-Daten sind wie folgt angelegt:

Array(1) x-Bildschirmposition

Array(2) y-Bildschirmposition

Array(3) ist unbenutzt

Array(4) = Breite; Array(5) = Höhe

- **CLP** – Diese Dateien enthalten bis zu 99 Einzelbilder. In den ersten 99 Einträgen des Arrays ist jeweils vermerkt, an welcher Stelle im Array sich die Einzelbilder befinden. In Array(1) steht beispielsweise, ab welchem Array-Eintrag das erste Einzelbild steht. Das erste Bild steht standardmäßig ab Array(100). Wer mehr als 99 Einzelbilder in einem Array wünscht, kann das Format leicht abändern. CLP-Dateien können Einzelbilder unterschiedlicher Größe enthalten, sie müssen allerdings über eine gemeinsame Farbpalette verfügen. Möchten Sie beispielsweise das zehnte Bild eines geladenen CLP-Arrays darstellen, so funktioniert das folgendermaßen:

```
PUT X%, Y%, (Array(Array(10)), PSET
```

- **INF** – Info-Dateien im ASCII-Format. Sie enthalten Informationen zu den Einzelbildern (Höhe, Breite) einer CLP-Datei.

Bild 10 zeigt das Hauptmenü, das nach Start des Auto-Grabbers erscheint. Geben Sie hier einfach die gewünschte Ziffer ein, und betätigen Sie zur Ausführung die Enter-Taste. Beachten Sie, daß im Auto-Grabber keine Sicherheitsmechanismen eingebaut sind (um das Listing möglichst kompakt zu halten). Wird beispielsweise eine nicht vorhandene Bilddatei angegeben, so stürzt das Programm mit der Fehlermeldung »File not found« ab.

Im folgenden werden die Funktionen des Hauptmenüs etwas ausführlicher erläutert:

- **Total Scan** – hinter dieser Option steckt die erste Scan-Routine, bei der Einzelobjekte durch Ermitteln eines Startpunkts gefunden werden. Total Scan ist ideal zum Speichern von Einzelbildern, die in keinem Sprite-Animations-Zusammenhang miteinander stehen.

- **Sprite Scan** – die zweite Scan-Routine, die im Beispiel Erdkugel erläutert wurde. Sie eignet sich insbesondere dann, wenn Animations-Phasen gespeichert werden. Die

Sprite-Scan-Routine läßt sich unter anderem auch einsetzen, um trickreich vielfarbige Fonts beliebiger Größe auf den Schirm zu bringen. Die Fonts werden einfach mit einem Grafikprogramm editiert, und dann werden per Auto-Grabber die Buchstaben der Reihenfolge nach eingelesen. Wird der Buchstaben-Satz in einem CLP-Bild-Array gespeichert, so zeigt der Array-Index auf die Position der Buchstaben; ihre jeweilige Breite – wichtig für das Positionieren bei der Ausgabe – können Sie ebenfalls problemlos an den entsprechenden Array-Positionen ermitteln.

Für einen Scan-Vorgang müssen Sie sich zu Beginn stets für eine der beiden obigen Scan-Routinen entscheiden – welche eingesetzt wird, hängt stets vom geplanten Vorhaben ab. Nach Anwahl einer Scan-Routine, werden weitere Optionen bereitgestellt (Bild 11). Unter anderem müssen Sie hier den Dateinamen des Quellbildes angeben. Dabei dürfen Sie die Dateierweiterung nicht vergessen – anhand dieser kann das Programm feststellen, welcher externe Grafikview zum Einsatz kommen muß. Anschließend fragt der Auto-Grabber nach einem Frame-/Clip-Dateinamen. Hier können Sie einen beliebigen angeben, ohne Dateierweiterung. Zu Beginn eines jeden Scannens speichert der Auto-Grabber die Farbpalette des Bildes in einer PLT-Datei ab. Die PLT-Datei ist mit dem angegebenen Dateinamen bezeichnet. Achten Sie beim Angeben eines Frame-/Clip-Dateinamens darauf, daß die Dateien bei Einzelbild-Speicherung durchnummeriert werden (bild1.dat, bild2.dat, bild3.dat etc.). Möchten Sie also hundert Einzelbilder nacheinander speichern, so darf der angegebene Dateiname maximal fünf Zeichen lang sein.

Die Ausgangsoptionen legen außerdem fest, mit welchem Verfahren die Einzelbilder gespeichert werden (Nachfrage entfällt bei der Funktion »ZOOM-Perspektive«):

- **Einzelframes** – hier speichert der Auto-Grabber jedes eingefangene Bild separat im PUT-Dateiformat. Der angegebene Dateiname wird mit einer Numerierung versehen.

- **Clipping-Puffer** – hier werden alle Einzelbilder in einem Array abgelegt, das am Ende des Scan-Vorgangs als eine einzige CLP-Datei gespeichert wird.

- **Data-Clip-Art** – wie Einzelframes, allerdings erfolgt die Speicherung der Bilddaten im DAT-Format, bei dem auch die Position des Bildausschnitts abgelegt wird.

Sie können – von wenigen Ausnahmen abgesehen – also stets unabhängig vom Scan-Verfahren festlegen, in welchem Bilddatei-Typ die Einzelbilder abgelegt werden.

Die Funktion »DPaint-Animation-Import« gestattet es Ihnen, beliebige Animationen des kommerziellen Grafikwerkzeugs Deluxe Paint Animation von Basic aus abzuspielen. Animationen bestehen bei Deluxe aus Einzelbildern

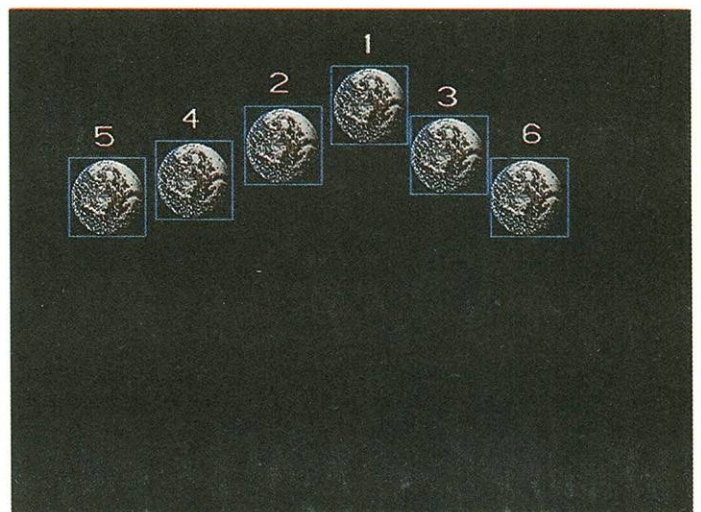


Bild 8. Hier die Reihenfolge, in der die erste Scan-Hauptroutine die Einzelbilder speichert

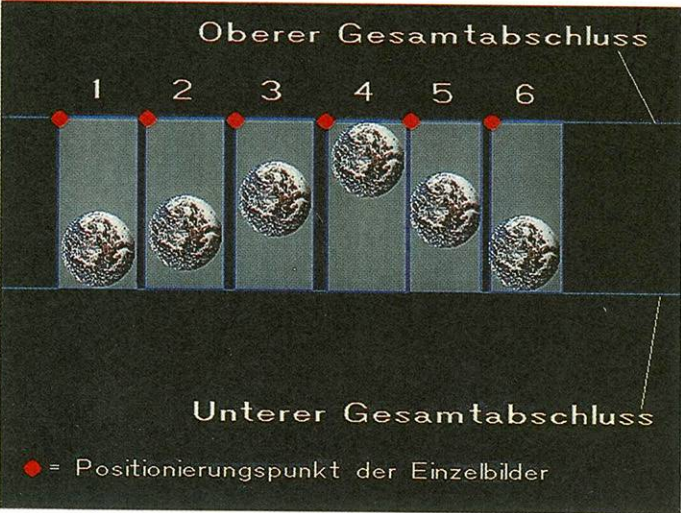


Bild 9. Arbeitsweise der zweiten Scan-Hauptroutine – sie sucht nach einem gemeinsamen oberen und unteren horizontalen Abschluß.

(Frames), die in Folge abgespielt werden. Beim Speichern einer Animation legt Deluxe eine Datei mit Endung »anm« an, in der alle Einzelbilder gespeichert sind. ANM-Dateien liegen in einem komprimierten Format vor, das seinem Namen leider nicht gerecht wird: rechnen Sie die unkomprimierte Datenmenge eines Einzelbildes aus (ein kompletter Bildschirm mit 320 x 200 Bildpunkten und 256 Farben benötigt 64 000 Byte), so zeigt sich, daß das ANM-Format teils sogar größere Datenmengen produziert. Was liegt also näher, als alle Einzelbilder einfach im Basic-Format abzuspeichern und sie dann Bild für Bild schnell von der Festplatte (am besten mit Cache oder RAM-Disk) abzuspielen. Durch die Konvertierung der ANM-Dateien in Basic-Einzelbilder gewinnen Sie sogar einen weiteren enormen Vorteil: Sie können jederzeit einzelne Bilder einer Animation laden oder gar einen bestimmten Bildbereich einer Animation abspielen (einen interaktiv steuerbaren Player finden Sie im Rahmen der VGA-Toolbox). Und mehr: Stellen Sie sich den altbekannten Vogel vor, der über ein festes Hintergrundbild flattert. Um eine solche Animation anzufertigen, ist bei Deluxe Paint Animation so vorzugehen: Zuerst müssen Sie festlegen, aus wie vielen Einzelbildern die Animation bestehen soll.

DPaint-Animation-Import

Anschließend zeichnen oder importieren Sie das Hintergrundbild und kopieren es auf alle verfügbaren Einzelbilder. Die Ausgangssituation für die Vogel-Animation belegt also bereits eine ganze Menge Speicherkapazität. Zuletzt werden die Einzelbilder durchgeschaltet und der Vogel in seiner Flugbahn versetzt daraufkopiert. Dann liegen jede Menge praktisch identische Bilder vor, die sich lediglich durch die Position des Vogels unterscheiden. Wollen Sie das Problem mit dem Deluxe-Paint-Animation-Konverter des Auto-Grabbers und diversen Routinen der VGA-Toolbox lösen, so sieht das folgendermaßen aus: Erst müssen Sie ein einziges Mal den Bildhintergrund laden. Anschließend bewegen Sie eine CLP-Datei mit den Einzelbewegungen des Vogels auf einer definierten Bahn über den Bildschirm. Am Bildschirm sieht dann alles aus, als ob die Original-ANM-Datei abliefe, die Speicherkapazitäts-Einsparung ist allerdings enorm. Was für Deluxe Paint Animation gilt, trifft auch für alle ähnlichen Animationsprogramme zu, wie den Autodesk-Animator. Der Auto-Grabber unterstützt von Haus aus zwar nur Deluxe-Paint-Animationen, Sie können ihn allerdings mit geringem Aufwand für jedes andere Animationsprogramm an-

passen. Hierfür muß nur die Lade-Routine für die Einzelbilder entsprechend verändert werden. Voraussetzung für das jeweilige Animations-Programm ist, daß es gestattet, anstelle einer kompletten Animations-Datei einzelne durchnummerierte Bilder zu speichern, die sich dann mit einem Viewer auf den Bildschirm bringen lassen. Bild 12 verdeutlicht den grundsätzlichen Vorgang bei Deluxe. Nachdem eine Animation fertiggestellt beziehungsweise geladen ist, wird über die Pull-down-Leiste die Option »Pict – Save Picture« gewählt. Am Bildschirm erscheint eine Dialogmaske, bei der Sie hinter »File name:« den gewünschten Namen für die Einzelbilder angeben. Anschließend markieren Sie die Option »Save Sequenz« und geben den gewünschten Einzelbilderbereich an. Dann speichert Deluxe Paint Animation die Einzelbilder unter dem angegebenen Dateinamen (zuzüglich Numerierung) einzeln ab. Im Beispiel wurde als Dateiname »pano« und als Bereich 1 bis 20 angegeben. Nach Abschluß der Einzelbilderspeicherung und Verlassen von Deluxe Paint Animation finden Sie die Einzelbilder in folgenden Dateien: »pano0001.lbm« bis »pano0020.lbm«.

Bewegungsphasen eines Sprites

Diese Einzelbilder können anschließend mit dem Auto-Grabber eingelesen und in einem Basic-Bildformat gespeichert werden. Wählen Sie hierzu im Auto-Grabber-Hauptmenü zunächst die Auswahloption »DPaint-Animation-Import« an. Anschließend werden Sie nach der gewünschten Bilddatenart gefragt. Einzel-Frames bieten sich an, wenn die gleichgroßen Einzelbilder einer Animation stets an der gleichen Bildschirmposition erscheinen, DAT-Bilddateien (Data-Clip-Art), wenn Einzelbilder an unterschiedlichen Positionen auftauchen. Der Einsatz von CLP-Dateien (Clipping-Puffer) eignet sich, um Bewegungsphasen eines Sprites in einem gemeinsamen Array abzulegen. In Verbindung mit der Vga-Toolbox lassen sich CLP-Dateien dann ähnlich einsetzen wie der Animations-Pinsel von Deluxe Paint Animation. Beachten Sie allerdings, daß eine CLP-Datei maximal 64 000 Byte an Bilddaten enthalten kann. Beim Versuch, mehr Bilddaten einzulesen, stürzt der Auto-Grabber zwangsläufig ab. Nach Auswahl eines Bilddateiformats werden Sie aufgefordert, einen Bilddateinamen anzugeben – hier ist der Dateiname der Einzelbilder ohne Numerierung und Dateiendung anzugeben. Bei der anschließenden Frage nach dem Zieldateinamen können Sie einen beliebigen Namen angeben – allerdings muß im Dateinamen noch ausreichend Platz für das Durchnummerieren vorhanden sein. Werden etwa 50 Einzelbilder gewünscht, so darf

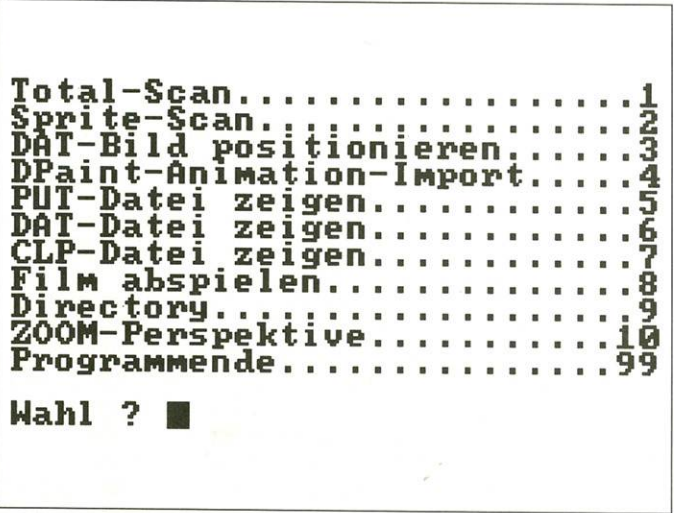


Bild 10. Das Hauptmenü des Auto-Grabber-Utility

Total-Scan.....	1
Sprite-Scan.....	2
DAT-Bild positionieren.....	3
DPaint-Animation-Import.....	4
PUT-Datei zeigen.....	5
DAT-Datei zeigen.....	6
CLP-Datei zeigen.....	7
Film abspielen.....	8
Directory.....	9
ZOOM-Perspektive.....	10
Programmende.....	99
Wahl ? 1	

Bild 11. Optionen und Einstellungen für den eigentlichen Scan-Vorgang

der Dateiname maximal sechs Buchstaben groß sein. Schließlich verlangt die Konvertierungs-Routine noch die Angabe der Bild-Anzahl, anschließend läuft die komplette Konvertierung automatisch ab.

Die gespeicherten Bildfolgen können dann mit der Option »Film abspielen« des Auto-Grabber-Hauptmenüs betrachtet werden.

Über die Funktion »PUT-Datei zeigen« lassen sich PUT-Dateien am Bildschirm betrachten. Nach Anwahl der Option gibt die Routine zunächst per DOS-Shell alle Dateien des aktuellen Verzeichnisses aus, die mit »put« enden. Diese Art der Dateiauswahl ist keineswegs komfortabel, allerdings müssen Sie berücksichtigen, daß das Auto-Grabber-Programm sehr kompakt gestaltet wurde, um Ihnen Tipparbeit zu ersparen. Nach Durchlauf der Dateianzeige, ist die gewünschte Datei anzugeben, wobei auf die Angabe der Dateierweiterung verzichtet werden kann. Anschließend erscheint die PUT-Datei am Bildschirm, bis eine Taste zum Abbruch der Darstellung betätigt wird. Am unteren Bildschirmrand wird zudem eine Informationszeile ausgegeben, aus der Bild-Dateiname, Höhe und Breite hervorgehen (Bild 14).

Die Option »DAT-Datei zeigen« funktioniert wie »PUT-Datei zeigen«, ist jedoch für DAT-Bilddateien gedacht.

Der Viewer »CLP-Datei zeigen« arbeitet ähnlich wie die beiden zuvor beschriebenen. Da eine CLP-Datei mehrere Bilder enthalten kann, ist die View-Routine allerdings entsprechend darauf vorbereitet. Nach Angabe des Quelldateinamens erscheint das erste Bild der CLP-Datei am Bildschirm, die Infozeile informiert zusätzlich über die Bildnummer (SpriteNR). Wird eine beliebige Taste gedrückt, so schaltet der Viewer zur Darstellung des nächsten Bilds in der CLP-Datei um. Ist das letzte Bild betrachtet, erfolgt ein Rücksprung ins Hauptmenü. Beim Darstellen von CLP-Dateien legt der Auto-Grabber im aktuellen Verzeichnis automatisch eine Datei an, deren Namen dem der CLP-Datei entspricht, allerdings die Dateierweiterung »inf« trägt. In dieser ASCII-Datei sind zu jedem Einzelbild der CLP-Datei die Höhe und Breite eingetragen. Sie benötigen diese Info-Dateien insbesondere dann, wenn Sie Schriftfonts in CLP-Dateien ablegen.

Einsatz des Auto-Grabbers als Viewer

Wenn Sie den Auto-Grabber kompilieren und als ausführbares Programm bereithalten, läßt er sich auch als Viewer für PUT-, DAT- und CLP-Dateien verwenden, wobei die Steuerung über Aufrufparameter erfolgen kann:

```
grabber.exe bild.put
```

zeigt dann beispielsweise die Bilddatei »bild.put« an. Die Dateierweiterung muß beim Kommandozeilen-Aufruf mitangegeben werden. Zum Einsatz kommen die gleichen Darstellungsroutinen wie im Auto-Grabber-Hauptmenü. Allerdings beendet das Programm auf Tastendruck die Bilddarstellung.

Mit der Funktion »DAT-Bild positionieren« können Sie Bilder, die im DAT-Format gespeichert sind, positionieren, und nach Angabe des gewünschten Bilddateinamens erscheint ein leerer Bildschirm mit dem Bildteil. Er kann mit den Pfeiltasten an eine beliebige Bildschirmposition be-

wegt werden. Beachten Sie, daß es zum Programmabsturz kommt, wenn Sie das Bild über die Grenzen des sichtbaren Bildschirmbereichs hinausbewegen, da die Grafik-Anweisungen »PUT« und »GET« kein Positionieren über die Bildränder hinaus gestatten. Mit Druck auf [Return] wird die neue Bildposition ins Array eingetragen und das Bild gespeichert.

Die Option »Film abspielen« dient zum Bild-für-Bild-Abspielen gespeicherter Bildsequenzen, die vom Auto-Grabber durchnummeriert wurden. Dies sind in der Regel Bilddateien der Option »DPaint-Animation-Import« oder »ZOOM-Perspektive«. Bildsequenzen dieser Art liegen entweder im PUT- oder DAT-Format vor. »Film-Abspielen« unterstützt dementsprechend DAT- und PUT-Bildsequenzen. Das Bild-für-Bild-Abspielen dient zur Kontrolle, ob ein Animationseffekt beziehungsweise die Einzel-Frames einer Animation in Ordnung sind, Nach Anwahl von »Film abspielen« im Hauptmenü müssen Sie zunächst einen Dateinamen angeben. Geben Sie hier den Sequenz-Dateinamen ohne Numerierung, allerdings mit Extension an. Ohne Extension kann der Auto-Grabber nicht feststellen, ob DAT- oder PUT-Dateien abzuspielen sind. Beispiel: Liegen drei Bilder mit den Dateinamen »pano1.put«, »pano2.put« und »pano3.put« vor, so geben Sie als Dateinamen fürs Abspielen einfach »pano.put« ein. Anschließend möchte der Auto-Grabber mit seiner Nachfrage »Anzahl der Bilder« wissen, aus wie vielen Einzelbildern die Animation besteht.

Geben Sie hier den entsprechenden Wert an. Es ist natürlich auch gestattet, weniger Bilder anzugeben als vorhanden sind – dann werden die letzten Bilder beim Abspielen einfach ausgelassen, die Gesamtanimationsdauer verkürzt sich. Nach der Bildzahlangabe beginnt die Animation automatisch. Bei Animationsstart ist die Durchlaufrichtung auf vorwärts gestellt. Mit Druck auf die Taste [r] können Sie die Abspielrichtung jederzeit umschalten. Mit der Taste [f] wird die Anzeige der Nummer des aktuellen Frames ein- und ausgeschaltet. Ist die Numerierung aktiviert, so ist die Framenummer permanent in der linken oberen Bildschirm-ecke zu sehen. Die Abspielgeschwindigkeit wird durch die beiden Tasten [+] und [-] reguliert. [+] erhöht die Wartezeit zwischen den Einzelbildern, [-] erniedrigt sie. Standardmäßig ist die Warteschleife auf 0 eingestellt – also maximale Geschwindigkeit. Die Abspielgeschwindigkeit (dies gilt auch für die VGA-Toolbox-Routinen am Ende dieses Workshops) hängt von den Faktoren PC-Leistung (Speicher, Grafikkarte, Festplatte) und Bildgröße ab. Je größer ein Bild ist, umso größer ist die Nachladezeit für Einzelbilder – berücksichtigen Sie dies, wenn Sie ein großes Projekt

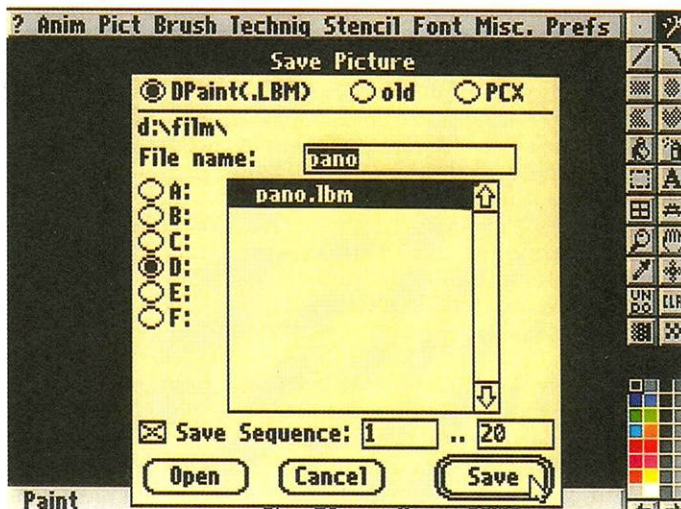


Bild 12. So werden Einzelbilder bei Deluxe Paint Animation mit automatischer Numerierung gespeichert.

Total-Scan.....	1
Sprite-Scan.....	2
DAT-Bild positionieren.....	3
DPaint-Animation-Import.....	4
PUT-Datei zeigen.....	5
DAT-Datei zeigen.....	6
CLP-Datei zeigen.....	7
Film abspielen.....	8
Directory.....	9
ZOOM-Perspektive.....	10
Programmende.....	99

Wahl ? 4	
EinzelFrames.....	1
Clipping-Puffer.....	2
Data-Clip-Art.....	3
Wahl ? 1	
Bilddatei (.bld)? pano	
Frame-/Clip-Dateiname? pano	
Anzahl der Bilder?? 20	

Bild 13. Konvertieren von Deluxe-Paint-Animation-Bildfolgen mit dem Auto-Grabber

vorhaben, bei dem Animationen in unterschiedlich großen Bildschirmbereichen ablaufen. Mit Druck auf die Escape-Taste wird das Abspielen beendet, das Hauptmenü erscheint wieder.

- Directory – zeigt sämtliche PUT-, DAT- und CLP-Dateien des aktuellen Laufwerks an.
- ZOOM-Perspektive – diese Funktion steht in keinem direkten Zusammenhang mit dem Auto-Grabber. Allerdings hat sich ihre Integration angeboten, da in Verbindung mit den Auto-Grabber-Routinen hier schnell und einfach ein interessanter Effekt geboten wird: Zoom mit Perspektive. Eine ausführliche Beschreibung finden Sie im folgenden Beitrag dieses Workshops.

Der Auto-Grabber ist mehr für den universellen Einsatz als für höchstmögliche Ausführungsgeschwindigkeit konzipiert. Lassen Sie dem Auto-Grabber Zeit, wenn am Bildschirm eine Zeitlang scheinbar nichts passiert. Der Auto-Grabber benötigt etwas Zeit, um beispielsweise festzustellen, daß sich kein Grafikobjekt mehr am Bildschirm befindet. Beachten Sie außerdem, daß einige MS-DOS-Bildviewer erst dann zurückspringen, wenn eine Taste gedrückt wird. Daß der Auto-Grabber arbeitet, können Sie an den durchlaufenden Scan-Linien sehen.

Beschreibung des Listings

Haben Sie sich entschieden, für welchen Zweck Sie den Auto-Grabber einsetzen möchten, so sollten Sie ihn entsprechend modifizieren. In diesem Zusammenhang empfiehlt sich ein weiterer Blick auf die Zoom-Funktion, bei der zwar beliebig viele Einzelbilder entstehen, die allerdings alle stets die gleiche Größe haben. Wer das nicht ändern will, kann darauf verzichten, für jedes Speichern eines Einzelbildes die Total-Scan-Routine aufzurufen, da die Position und Größe des zu speichernden Bildes ja bekannt sind. Es reicht aus, die Total-Scan-Routine nur einmal zu Beginn zu durchlaufen und Bildposition und Größe gegebenenfalls in Variablen zu speichern. Dann können die Einzelbilder nach jeder Berechnung direkt mit der Sub-Routine »SAVE« gespeichert werden. Beim Auto-Grabber-Listing finden Sie zudem einige geringfügig modifizierte Sub-Routinen der VGA-Toolbox. Hier abschließend eine kurze Beschreibung des Listings. Grundlagen zu den einzelnen Routinen finden Sie im Beitrag zur VGA-Toolbox. Beachten Sie bei der Eingabe des Listings, daß zu Beginn sämtlicher Sub-Routinen die Anweisung »DEFINT A-Z« stehen muß.

In den Zeilen 1 bis 58 legen Sie die Deklaration der Sub-Routinen und die Programminitialisierung fest, dimensionieren erforderliche Arrays und legen Ausgangswerte fest. In dem anschließenden If-then-Block bis Zeile 82 steckt eine Abfrage von Command\$. Command\$ ist eine Variable, die zu Beginn eines Programmstarts die angegebenen Aufrufparameter enthält. So können Programmeinstellungen per Stapeldatei automatisiert werden. In den Zeilen 59 bis

82 verzweigt das Programm – falls ein Aufrufparameter angegeben wurde – zu der entsprechenden View-Routine und beendet nach der Darstellung seine Ausführung. Die Hauptroutine und das Auswahlmenü finden Sie in den Zeilen 83 bis 107.

Eine Routine spielt Bildsequenzen (Zeile 108 bis 176) ab, die als Einzelbilder im PUT- oder DAT-Bildformat gespeichert sind.

In den Zeilen 177 bis 256 finden Sie die Routinen zum Laden und Darstellen einer PUT-Bilddatei, einer DAT-Bilddatei und einer CLP-Bilddatei, in den Zeilen 257 bis 264 die Routine zum Aufruf der Sub-Routine »PosClip« zum Positionieren eines Bildes am Bildschirm.

In den Zeilen 265 bis 277 wird das Verzeichnis ausgegeben, anschließend findet eine Abfrage statt, in welchem Format die Bilder gespeichert werden sollen.

Eine Abfrage der Optionen und Einstellungen für eine Zoom-Sequenz oder eine Bildausschnitt-Vergrößerung finden Sie in den Zeilen 288 bis 295.

Die Initialisierung eines Clipping-Arrays sehen Sie in den Zeilen 296 bis 300 (falls erforderlich).

Eingabe von Bild-Quelldatei und Bild-Zieldatei und die Ausführung des Unterprogramms »DPaintImport« finden Sie in Zeile 301 bis 305.

In den Zeilen 306 bis 327 finden Sie eine Routine zum Laden eines Quellbildes. Hier kommen externe DOS-Grafik-Viewer zum Einsatz. Welcher benötigt wird, erkennt das Programm an der Dateiendung – sie ist also stets mit anzugeben. In diesem Programmbereich können Sie beliebige andere DOS-Grafik-Viewer eintragen. Achtung: bei Programmentwicklung in der Basic-Shell steht unter Umständen nicht genug Speicher zum Ausführen eines DOS-Programms via Shell-Anweisung zur Verfügung. Das macht sich insbesondere dann bemerkbar, wenn Sie einen Viewer einsetzen, der das Rollen beliebig großer Bilder in der Sichtauflösung zuläßt. Bei Speicherproblemen (sie können sich auch lediglich durch kommentarlos Nichtfunktionieren des externen Viewers äußern) hilft es in der Regel, ein ausführbares Programm zu kompilieren und von DOS aus zu starten.

In Zeile 328 bis 330 findet ein Aufruf der Superzoom-Routine statt.

Den Aufruf der ersten und zweiten Scanroutine sehen Sie in Zeile 331 bis 335.

Zeile 336 bis 378: Routine für perspektivisches Zoomen und Speichern von Einzelbildern mit automatischer Durchnumerierung als Dateinamenszusatz. Am Ende der Routine erfolgt ein Sprung zu »InstantPlay«.

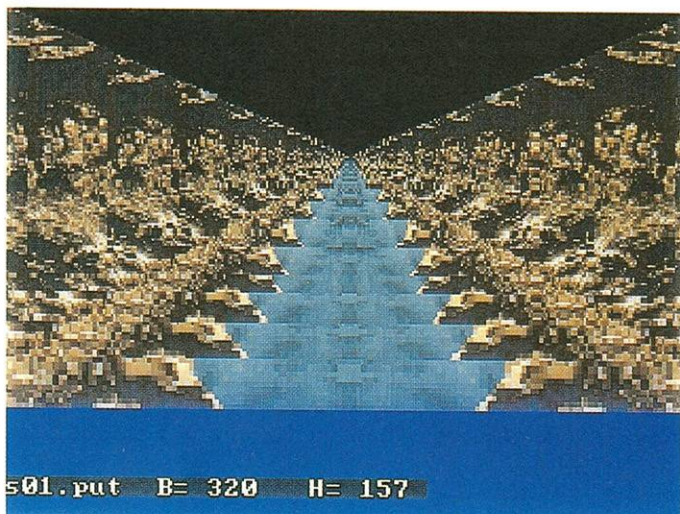


Bild 14. Der Auto-Grabber beinhaltet Viewer für PUT-, DAT- und CLP-Bilddateien, die auch über Aufrufparameter aktivierbar sind.

Infodatei:test.clp

```

SpriteNR = 1  Breite= 62  Höhe= 35
SpriteNR = 2  Breite= 62  Höhe= 35
SpriteNR = 3  Breite= 62  Höhe= 35
SpriteNR = 4  Breite= 62  Höhe= 35
SpriteNR = 5  Breite= 62  Höhe= 35
SpriteNR = 6  Breite= 62  Höhe= 35
SpriteNR = 7  Breite= 62  Höhe= 35
SpriteNR = 8  Breite= 62  Höhe= 35
SpriteNR = 9  Breite= 62  Höhe= 35
    
```

Bild 15. INF-Dateien helfen, Einzelbilder in CLP-Bilddateien zu finden.

Die DPaintImport-Routine finden Sie in Zeile 379 bis 405 – sie konvertiert LBM-Bildsequenzen, die von Deluxe-Paint Animation PC gespeichert worden sind. Diese Routine arbeitet nur in Verbindung mit einem externen LBM-Viewer, der sich über DOS-Aufruf-Parameter starten läßt. Hier wurde »vpic« verwendet – für einen anderen Viewer muß Zeile 399 geändert werden.

In Zeile 406 bis 475 sehen Sie die Total-Scan-Routine (Scan-Routine 1), die alle Objekte des Bildschirms speichert, unter dem Zieldateinamen mit Nummer (ziel1.dat, ziel2.dat).

Zeile 476 bis 533 zeigt die Sprite-Scan-Routine (Scan-Routine 2). Sie holt sich alle Objekte mit gemeinsamem Zeilenabschluß auf den Bildschirm und speichert sie unter dem Zieldateinamen mit Nummer (ziel1.dat, ziel2.dat etc).

Der Programmteil in Zeile 534 bis 558 testet, wo sich der nächste belegte Punkt befindet, oben links beginnend, und liefert die Ergebnisse in die Variablen »FundZeile%« und »FundSpalte%«. Enthalten beide Variablen ihren Aus-

gangswert -1, so befindet sich am Bildschirm kein einziger Punkt mehr, ungleich der Hintergrundfarbe (Farbe 0).

Zeile 559 bis 568: Die Sub-Routine »AddFrame« hängt einem Sprite-Array das nächste Bildelement an und berechnet die Array-Position für das nächste Element.

Zeile 569 bis 576: Die Sub-Routine »InitClipping« bereitet den gewählten Puffer für ein neues Clipping vor.

Die Funktion »KillNummer\$« (Zeile 577 bis 593) dient nur für den Auto-Grabber. Sie entfernt aus dem angegebenen Bilddateinamen Bildnumerierungen, um dann den zugehörigen Paletten-Dateinamen zu ermitteln. Die Dateiendung wird ebenfalls abgeschnitten, falls sie vorhanden ist.

Zeile 595 bis 615 zeigt die Sub-Routine »LadeBild« zum Einlesen eines im BLD-Format gespeicherten Bilds im Bildschirmmodus 13 (VGA, Auflösung 320 x 200 in 256 Farben). Der Aufruf erfolgt mit »LadeBild Dateiname\$«. Die Endungen für die Bilddatei (»bild«) und Palettedatei (»plt«) werden automatisch angefügt – Sie müssen den Dateinamen also ohne Extension angeben.

Zeile 616 bis 654: Die Sub-Routine »LOAD« lädt eine Bilddatei in ein Array und plaziert das Bild am Bildschirm.

Weiterhin werden folgende Sub-Routinen vereinbart: Zeile 655 bis 662: »LoadClipping« – lädt eine CLP-Bilddatei in ein Array.

Zeile 663 bis 679: »LoadDataClip« – lädt eine DAT-Bilddatei in ein Array.

Zeile 680 bis 709: »PaletteLaden«

Zeile 710 bis 726: »PaletteSpeichern«

Zeile 727 bis 736: »pause« – wartet auf Tastendruck.

Zeile 737 bis 785: »PosClip« – zum (Um-)Positionieren einer DAT-Bilddatei

Zeile 786 bis 810: »SAVE« – lädt eine PUT-Bilddatei in ein Array und speichert es dann.

Zeile 811 bis 836: »SaveClipping« und »SaveDataClip« – laden und speichern ein CLP-Array.

Zeile 837 bis 1094: Unterprogramme für die Scan-Routinen. – In den letzten Zeilen finden Sie schließlich noch die Sub-Routine »VgaZoom«. Sie zoomt ein Bild-Array von seinem Mittelpunkt heraus um den angegebenen Faktor.

(Michael Nickles/uk)

```

[V95] 1: 'Programm: grabber.bas
[L77] 2: 'Funktion: Scant VGA-Bilder und speichert'
[GX6] 3: 'sämtliche Bildelemente einzeln ab.
[855] 4: 'Sprache: QuickBasic
[CD5] 5: 'Autor: Michael Nickles
[XM5] 6: '(c) 1992 DMV Widuch GmbH & Co. KG
[000] 7:
[TL1] 8: DEFINT A-Z
[UNV] 9: DECLARE SUB VgaZoom (Array() AS INTEGER, XZ
oomFaktor AS SINGLE, YZoomFaktor AS SINGLE,
X AS INTEGER, y AS INTEGER, BildNeuBreite
AS INTEGER, BildNeuHoehe AS INTEGER)
[XYB] 10: DECLARE SUB PaletteLaden (VonFarbe%, BisFar
be%, ZielBereichStart%, Datei$)
[G76] 11: DECLARE SUB PaletteSpeichern (Datei$)
[WY6] 12: DECLARE FUNCTION KillNummer$ (Dateiname$)
[573] 13: DECLARE SUB pause ()
[2R3] 14: DECLARE SUB PosClip ()
[A09] 15: DECLARE SUB SaveDataClip (X%, y%, Breite%,
Hoehe%, Datei$)
[9E5] 16: DECLARE SUB LoadDataClip (Datei$)
[GB5] 17: DECLARE SUB SavePalette (Datei$)
[117] 18: DECLARE SUB SaveClipping (Array%(), Datei$)
[8Z6] 19: DECLARE SUB LoadClipping (Array%(), Datei$)
[1V5] 20: DECLARE SUB InitClipping (Array%())
[LD8] 21: DECLARE SUB AddFrame (X%, y%, Breite%, Hoeh
e%, Array%())
[AZ5] 22: DECLARE SUB LOAD (X%, y%, Art%, Datei$)
[GK7] 23: DECLARE SUB SAVE (X%, y%, Breite%, Hoehe%,
Datei$)
[XH4] 24: DECLARE SUB LadeBild(Datei$)
[E27] 25: DECLARE FUNCTION TestArea1% (x1%,y1%,x2%,y2
%)
[H27] 26: DECLARE FUNCTION TestArea2% (x1%,y1%,x2%,y2
%)
[958] 27: DECLARE FUNCTION TestVerLine% (X%,YStart%,Y
Ende%)
[F18] 28: DECLARE FUNCTION TestHorLine% (XStart%,y%,X
Ende%)
    
```

```

[SV4] 29: COMMON SHARED ClippingStart%
[9V4] 30: COMMON SHARED ClipInfoStart%
[W94] 31: COMMON SHARED FrameAnzahl%
[X24] 32: COMMON SHARED PufferArt%
[S94] 33: COMMON SHARED Einrahmen%
[YB3] 34: Start% = 1: Ende% = 32100
[TK5] 35: DIM SHARED Puffer(Start% TO Ende%)
[Y46] 36: 'Puffer für PUT/GET Grafik-Befehle
[EX4] 37: DIM SHARED TempPuffer(1 TO 1000)
[XE4] 38: 'Puffer für Scan-Tracing
[6G4] 39: DIM SHARED ClipInfo%(1 TO 255)
[GA3] 40: DIM SHARED Pal%(255, 2)
[S35] 41: 'Puffer für VGA-Farbpalette
[TL4] 42: CONST FramePufferArt% = 1
[FY3] 43: CONST ClipPufferArt% = 2
[ZC3] 44: CONST DataClipArt% = 3
[DV2] 45: 'FramePufferArt:
[0K6] 46: 'Hier werden alle eingefangenen Einzel-
[YD7] 47: 'objekte einzeln abgespeichert (als BLD/PLT
[SA6] 48: 'Dateien). Der angegebene Zieldateiname
[567] 49: 'wird mit jedem Einzelbild durchnummeriert
[697] 50: '(Numerierung wird am Dateinamen angehängt
[Wb6] 51: '(bild1.dat,bild2.dat ...) – bei Angabe des
[Y87] 52: 'Zieldateinamens ausreichend Platz für die
[3A4] 53: 'Numerierung frei lassen.
[6T2] 54: 'ClipPufferArt:
[HY6] 55: 'Speicherung von maximal 99 Einzelbildern
[R24] 56: 'in einem einzigen Array.
[SK4] 57: 'Speicherung als CLP-Datei.
[7C1] 58: SCREEN 13
[0C6] 59: 'Durch den folgenden IF-THEN-Block wird
[KW5] 60: 'der Auto-Grabber auch als Viewer für
[486] 61: 'DAT, CLP und PUT Dateien tauglich, die
[SE5] 62: 'einfach beim Programmstart als
[PE5] 63: 'Aufrufparameter anzugeben sind.
[286] 64: DO UNTIL i$ = " 'Tastaturpuffer leeren
[YL1] 65: i$ = INKEY$
    
```

»grabber.bas« scant ein Bild mehrfach ein und läßt diese gespeicherten Bilder anschließend als »Film« ablaufen.


```
[HS0] 66: LOOP
[X53] 67: IF COMMAND$ <> "" THEN
[843] 68:   i$ = LTRIM$(COMMAND$)
[632] 69:   i$ = RTRIM$(i$)
[AR2] 70:   ProgrammEnde% = 1
[B82] 71:   Dateiname$ = i$
[GM3] 72:   SELECT CASE RIGHT$(i$, 4)
[2G2] 73:     CASE IS = ".PUT"
[472] 74:       GOTO PutView
[ED2] 75:     CASE IS = ".DAT"
[322] 76:       GOTO DatView
[XD2] 77:     CASE IS = ".CLP"
[N42] 78:       GOTO ClpView
[GS1] 79:   END SELECT
[4M0] 80: END
[801] 81: END IF
[014] 82: '-----
[922] 83: HauptMenue:
[GC1] 84: PALETTE
[E46] 85: 'Standardpalette einschalten, damit
[FX5] 86: 'die ersten 16 Farben stimmen - sonst
[S25] 87: 'sehen Sie den Menütext nicht!
[5F4] 88: IF ProgrammEnde% = 1 THEN END
[PN0] 89: CLS
[HD3] 90: PRINT "Auto-Grabber "
[GX0] 91: PRINT
[T24] 92: PRINT "Total-Scan.....1"
[335] 93: PRINT "Sprite-Scan.....2"
[MT5] 94: PRINT "DAT-Bild positionieren.....3"
[BW5] 95: PRINT "DPaint-Animation-Import....4"
[YB5] 96: PRINT "PUT-Datei zeigen.....5"
[BA5] 97: PRINT "DAT-Datei zeigen.....6"
[NA5] 98: PRINT "CLP-Datei zeigen.....7"
[095] 99: PRINT "Film abspielen.....8"
[W15] 100: PRINT "Directory.....9"
[3L5] 101: PRINT "ZOOM-Perspektive.....10"
[X85] 102: PRINT "Programmende.....99"
[GX0] 103: PRINT
[0N3] 104: INPUT "Wahl "; Auswahl%
[252] 105: Auswertung:
[G02] 106: DConvert% = 0
[BS3] 107: IF Auswahl% = 99 THEN END
[9L4] 108: '-----
[T67] 109: 'Routine zum Abspielen einer gespeicherten
[AM5] 110: 'durchnummerierten Film-Sequenz
[9L4] 111: '-----
[923] 112: IF Auswahl% = 8 THEN
[9D6] 113:   INPUT "Dateiname (PUT/DAT)"; QuellDatei$
[YZ3] 114:   i$ = RIGHT$(QuellDatei$, 4)
[8G4] 115:   IF i$ = ".dat" THEN PlayArt = 1
[CK4] 116:   IF i$ = ".put" THEN PlayArt = 2
[0W7] 117:   QuellDatei$=LEFT$(QuellDatei$,LEN(QuellDatei$)-4)
[X96] 118:   IF QuellDatei$ = "" THEN GOTO HauptMenue
[6G6] 119:   INPUT "Anzahl der Bilder?"; BilderAnzahl%
[R36] 120:   PaletteLaden 0,255,0,QuellDatei$+".plt"
[192] 121: InstantPlay:
[192] 122:   temp$ = "00000000"
[RV5] 123:   temp$=LEFT$(temp$,8-LEN(QuellDatei$))
[PD2] 124:   StartFrame% = 1
[FW3] 125:   AktFrame% = StartFrame%
[U14] 126:   EndFrame% = BilderAnzahl%
[PN0] 127:   CLS
[GW2] 128:   DO UNTIL Taste$ = ""
[1C2] 129:     Taste$ = INKEY$
[HS0] 130:   LOOP
[VR1] 131:   Runner% = 1
[XB4] 132:   'bestimmt Abspielrichtung
[K32] 133:   FrameShow% = 1
[AT4] 134:   'Framennummer Anzeige ein/aus
[5S3] 135:   DO UNTIL Taste$ = CHR$(27)
[FM2] 136:     'Abbruch mit ESC
[1C2] 137:     Taste$ = INKEY$
[U55] 138:     IF Taste$="+" AND Delay!<100000 THEN
[HR2] 139:       Delay! = Delay! + 50
[TS1] 140:       SOUND 600, .1
[801] 141:     END IF
[924] 142:     IF Taste$ = "-" AND Delay! > 0 THEN
[NR2] 143:       Delay! = Delay! - 50
[TS1] 144:       SOUND 600, .1
[801] 145:     END IF
[8Y2] 146:     IF Taste$ = "r" THEN
[TS1] 147:       SOUND 600, .1
[PF3] 148:     Runner% = Runner% * (-1)
[801] 149:   END IF
[4X2] 150:   IF Taste$ = "f" THEN
[TS1] 151:     SOUND 600, .1
[CF4] 152:   FrameShow% = FrameShow% * (-1)
[801] 153:   END IF
[P75] 154:   Nummer$ = LTRIM$(STR$(AktFrame%))
[FY2] 155:   IF PlayArt = 1 THEN
[PG8] 156:     'Nummer$=LEFT$(temp$,LEN(temp$)-LEN(Nummer$)+1)
[LK6] 157:     LoadDataClip QuellDatei$+Nummer$+".dat"
[801] 158:   END IF
[KY2] 159:   IF PlayArt = 2 THEN
[LK5] 160:     LOAD 0,0,0,QuellDatei$+Nummer$+".put"
[801] 161:   END IF
[HH3] 162:   IF FrameShow% = -1 THEN
[8B4] 163:     LOCATE 1, 1: PRINT AktFrame%
```

```
[801] 164:   END IF
[AV3] 165:   FOR i! = 1 TO Delay!: NEXT
[DC5] 166:   'Warteschleife für Verzögerung
[315] 167:   AktFrame% = AktFrame% + Runner%
[DP6] 168:   IF Runner%=1 AND AktFrame% > EndFrame% THEN
[FW3] 169:     AktFrame% = StartFrame%
[801] 170:   END IF
[BF7] 171:   IF Runner% = -1 AND AktFrame% < StartFrame% THEN
[BC3] 172:     AktFrame% = EndFrame%
[801] 173:   END IF
[HS0] 174:   LOOP
[FU2] 175:   GOTO HauptMenue
[801] 176: END IF
[VA4] 177: '-----
[K95] 178: 'Laden und Darstellen einer PUT-
[DU1] 179: 'Bilddatei.
[VA4] 180: '-----
[X13] 181: IF Auswahl% = 5 THEN
[RZ2] 182:   SHELL "DIR *.put /p"
[CK4] 183:   INPUT "Dateiname"; Dateiname$
[V86] 184:   IF Dateiname$ = "" THEN GOTO HauptMenue
[WH1] 185:   PutView:
[W08] 186:   PaletteLaden 0,255,0,KillNummer$(Dateiname$)+".plt"
[MY2] 187:   CLS : PAINT (0, 0), 1
[6T4] 188:   IF INSTR(Dateiname$, ".")=0 THEN
[ST4] 189:     Dateiname$ = Dateiname$ + ".put"
[MC3] 190:   LOAD 0, 0, 0, Dateiname$
[F71] 191:   COLOR 15
[SM1] 192:   LOCATE 24, 1
[4F8] 193:   PRINT Dateiname$; " B="; Puffer(1) / 8; "
H="; Puffer(2);
[MW0] 194:   pause
[FU2] 195:   GOTO HauptMenue
[801] 196: END IF
[VA4] 197: '-----
[Y65] 198: 'Laden und Darstellen einer DAT-
[DU1] 199: 'Bilddatei.
[VA4] 200: '-----
[123] 201: IF Auswahl% = 6 THEN
[3W2] 202:   SHELL "DIR *.dat /p"
[CK4] 203:   INPUT "Dateiname"; Dateiname$
[8F1] 204:   DatView:
[V86] 205:   IF Dateiname$ = "" THEN GOTO HauptMenue
[W08] 206:   PaletteLaden 0,255,0,KillNummer$(Dateiname$)+".plt"
[PN0] 207:   CLS
[6T4] 208:   IF INSTR(Dateiname$, ".")=0 THEN
[AE4] 209:     Dateiname$=Dateiname$+".dat"
[2Y3] 210:   LoadDataClip Dateiname$
[F71] 211:   COLOR 15
[SM1] 212:   LOCATE 24, 1
[2F8] 213:   PRINT Dateiname$; " B= ; Puffer(1) / 8; "
H="; Puffer(2);
[MW0] 214:   pause
[FU2] 215:   GOTO HauptMenue
[801] 216: END IF
[VA4] 217: '-----
[D75] 218: 'Laden und Darstellen einer CLP-
[DU1] 219: 'Bilddatei.
[VA4] 220: '-----
[523] 221: IF Auswahl% = 7 THEN
[XD2] 222:   SHELL "DIR *.clp"
[CK4] 223:   INPUT "Dateiname"; Dateiname$
[RF1] 224:   ClpView:
[V86] 225:   IF Dateiname$ = "" THEN GOTO HauptMenue
[723] 226:   Dateinummer% = FREEFILE
[W08] 227:   PaletteLaden 0,255,0,KillNummer$(Dateiname$)+".plt"
[RCA] 228:   OPEN KillNummer$(Dateiname$) + ".inf" FOR
OUTPUT AS #Dateinummer
[PN0] 229:   CLS
[6T4] 230:   IF INSTR(Dateiname$, ".")=0 THEN
[LR4] 231:     Dateiname$ = Dateiname$ + ".clp"
[KW6] 232:   PRINT #Dateinummer,"Infodatei:";Dateiname$
[FE5] 233:   LoadClipping Puffer(), Dateiname$
[TZ1] 234:   AktFrame% = 1
[KS4] 235:   DO UNTIL Puffer(AktFrame%) = 0
[KG6] 236:     PUT (0,0),Puffer(Puffer(AktFrame%)),PSET
[F71] 237:     COLOR 15
[PM1] 238:     LOCATE 23, 1
[925] 239:     i$ = "SpriteNR=" + STR$(AktFrame)
[UM2] 240:     i$ = i$ + " Breite="
[C96] 241:     i$ = i$+STR$(Puffer(Puffer(AktFrame))/8)
[HL2] 242:     i$ = i$ + " Höhe="
[U76] 243:     i$ = i$+STR$(Puffer(Puffer(AktFrame)+1))
[5H3] 244:     PRINT #Dateinummer, i$
[BR2] 245:     PRINT Dateiname$
[SM1] 246:     LOCATE 24, 1
[WN4] 247:     PRINT "SpriteNR="; AktFrame%;
[SD6] 248:     PRINT " B="; Puffer(Puffer(AktFrame))/8;
[9E6] 249:     PRINT " H="; Puffer(Puffer(AktFrame)+1);
[MW0] 250:     pause
[PN0] 251:     CLS
[8U3] 252:     AktFrame% = AktFrame% + 1
[HS0] 253:   LOOP
[AL3] 254:   CLOSE #Dateinummer
[FU2] 255:   GOTO HauptMenue
```



```
[801] 256: END IF
[6Y3] 257: '-----
[VE6] 258: 'Aufruf der SUB-Routine PosClip zum
[KV6] 259: 'Positionieren eines Bildes am Bildschirm
[6Y3] 260: '-----
[N13] 261: IF Auswahl% = 3 THEN
[FE1] 262:   PosClip
[FU2] 263:   GOTO HauptMenu
[801] 264: END IF
[6Y3] 265: '-----
[MB6] 266: 'Listet sämtliche Bilddaten-Dateien
[AP5] 267: 'des aktuellen Verzeichnisses auf.
[6Y3] 268: '-----
[D23] 269: IF Auswahl% = 9 THEN
[SC2] 270:   SHELL "DIR *.dat"
[MW0] 271:   pause
[CG2] 272:   SHELL "DIR *.put"
[MW0] 273:   pause
[XD2] 274:   SHELL "DIR *.clp"
[MW0] 275:   pause
[FU2] 276:   GOTO HauptMenu
[801] 277: END IF
[W93] 278: IF Auswahl% <> 10 THEN
[CM5] 279:   'Eingabe/Abfrage der Scan-Optionen
[095] 280:   PRINT "EinzelFrames....(put)...1"
[LT2] 281:   'FramePufferArt
[H95] 282:   PRINT "Clipping-Puffer...(clp)...2"
[YH2] 283:   'ClipPufferArt
[PZ4] 284:   PRINT "Data-Clip-Art...(dat)...3"
[N12] 285:   'DataClipArt
[DW3] 286:   INPUT "Wahl "; PufferArt%
[801] 287: END IF
[S43] 288: IF Auswahl% = 10 THEN
[HR4] 289:   PufferArt% = FramePufferArt%
[CY5] 290:   INPUT "Anzahl Frames?"; BilderAnzahl%
[RR5] 291:   INPUT "X-ZOOM (0.1-n.0)"; XZoomFaktor!
[WR5] 292:   INPUT "Y-ZOOM (0.1-n.0)"; YZoomFaktor!
[TN6] 293:   INPUT "Zielbild-Breite"; ZielBildBreite%
[D66] 294:   INPUT "Zielbild-Höhe"; ZielBildHoehe%
[801] 295: END IF
[LS5] 296: IF PufferArt% = ClipPufferArt% THEN
[LN3] 297:   InitClipping Puffer()
[801] 298: END IF
[SP5] 299: INPUT "Bilddatei (.bld)"; QuellDatei$
[U66] 300: INPUT "Frame-/Clip-Dateiname"; ZielDatei$
[T13] 301: IF Auswahl% = 4 THEN
[K02] 302:   DConvert% = 1
[H73] 303:   GOSUB DPaintImport
[FU2] 304:   GOTO HauptMenu
[801] 305: END IF
[DT6] 306: 'Bei der Angabe des Quellbild-Dateinamens
[NW6] 307: 'muß die Dateiendung mitangegeben werden
[LB6] 308: '(GIF, LBM oder BLD). Nach der Dateiendung
[707] 309: 'entscheidet das Programm in der folgenden
[2E5] 310: 'SELECT-CASE-Abfrage, mit welchem
[VF6] 311: 'externen Viewer das Bild geladen wird.
[KB7] 312: 'Der/die Viewer müssen im Verzeichnis oder
[4V6] 313: 'im PATH sein. Ohne Zusatz-Viewer können
[536] 314: 'nur Bilder im BASIC-Format (BLD/PLT)
[VM2] 315: 'geladen werden.
[AA5] 316: SELECT CASE RIGHT$(QuellDatei$, 3)
[P72] 317:   CASE IS = "gif"
[GP4] 318:     i$ = "picem /v:1 " + QuellDatei$
[VT5] 319:     'benötigt externen Viewer PICEM !!!
[Y81] 320:     SHELL i$
[282] 321:   CASE IS = "lbm"
[L04] 322:     i$ = "vpic /r " + QuellDatei$
[9H5] 323:     'benötigt externen Viewer VPIC !!!
[Y81] 324:     SHELL i$
[H72] 325:   CASE IS = "bld"
[167] 326:     LadeBild LEFT$(QuellDatei$, LEN(QuellDatei$)-4)
[GS1] 327: END SELECT
[S43] 328: IF Auswahl% = 10 THEN
[LN2] 329:   COTO Superzoom
[801] 330: END IF
[0P5] 331: IF Auswahl% = 1 THEN GOSUB TotalScan
[MD5] 332: IF Auswahl% = 2 OR Auswahl% = 3 THEN
[HV2] 333:   GOSUB SpriteScan
[801] 334: END IF
[FU2] 335: GOTO HauptMenu
[9L4] 336: '-----
[3C7] 337: 'Routine für perspektivisches Zoomen und
[647] 338: 'speichern durchnummerierter Einzelbilder.
[WX6] 339: 'Achtung: Superzoom kann nur PUT-Dateien
[YW1] 340: 'speichern!
[4X4] 341: '-----
[FX1] 342: Superzoom:
[K02] 343: DConvert% = 1
[DT4] 344: SchreibDatei$ = ZielDatei$ + ".1"
[835] 345: UrDatei$ = SchreibDatei$ + ".put"
[ES2] 346: GOSUB TotalScan
[ZM7] 347: IF ZielBildBreite% <> 0 AND ZielBildHoehe% <
> 0 THEN
[RN0] 348:   BEEP
[MCD] 349:   VgaZoom Puffer(), 1+XZoomFaktor!, 1+YZoomFak
tor!, 0, 0, ZielBildBreite%, ZielBildHoehe%
[DT4] 350:   SchreibDatei$ = ZielDatei$ + ".1"
[ES2] 351:   GOSUB TotalScan
```

```
[801] 352: END IF
[SW5] 353: PaletteSpeichern ZielDatei$ + ".plt"
[PN0] 354: CLS
[SN3] 355: PUT (0, 0), Puffer, PSET
[5L4] 356: FOR izn = 2 TO BilderAnzahl%
[X13] 357:   LOAD 0, 0, 0, UrDatei$
[PN0] 358:   CLS
[GS4] 359:   z1! = 1 + (izn - 1) * (XZoomFaktor!)
[MS4] 360:   z2! = 1 + (izn - 1) * (YZoomFaktor!)
[667] 361:   'Werden für ZielBild-Höhe und ZielBild-
[M16] 362:   'Breite keine Angaben gemacht, so wird
[4R6] 363:   'die Größe des Quellbildes beibehalten.
[Y04] 364:   IF ZielBildBreite% = 0 THEN
[WT4] 365:     ZielBildBreite% = Puffer(1) / 8
[V14] 366:   IF ZielBildHoehe% = 0 THEN
[564] 367:     ZielBildHoehe% = Puffer(2)
[VT9] 368:   VgaZoom Puffer(), z1!, z2!, 0, 0, ZielBildBreit
e%, ZielBildHoehe%
[5B4] 369:   Nummer$ = LTRIM$(STR$(izn))
[LU5] 370:   SchreibDatei$ = ZielDatei$ + Nummer$
[ES2] 371:   GOSUB TotalScan
[ZS0] 372:   NEXT
[PD2] 373:   StartFrame% = 1
[FW3] 374:   AktFrame% = StartFrame%
[U14] 375:   EndFrame% = BilderAnzahl%
[VY3] 376:   QuellDatei$ = ZielDatei$
[BV1] 377:   PlayArt = 2
[SW2] 378:   GOTO InstantPlay
[9L4] 379: '-----
[X17] 380: 'DPaintImport-Routine - konvertiert LBM-
[8X5] 381: 'Bildsequenzen die von Deluxe-Paint
[096] 382: 'Animation PC gespeichert worden sind.
[SW6] 383: 'Diese Routine arbeitet nur in Verbindung
[EP6] 384: 'mit einem externen LBM-Viewer, der sich
[PD7] 385: 'über DOS-Aufruf-Parameter starten läßt.
[0M6] 386: 'Hier wurde VPIC verwendet - für einen
[HB6] 387: 'anderen Viewer die Zeile unten bitte
[V24] 388: 'entsprechend abändern.
[9L4] 389: '-----
[BB2] 390: DPaintImport:
[G66] 391: INPUT "Anzahl der Bilder?"; BilderAnzahl%
[192] 392: temp$ = "00000000"
[NB6] 393: temp$ = LEFT$(temp$, 8 - LEN(QuellDatei$))
[1L4] 394: FOR izn = 1 TO BilderAnzahl%
[5B4] 395:   Nummer$ = LTRIM$(STR$(izn))
[LU5] 396:   SchreibDatei$ = ZielDatei$ + Nummer$
[VA8] 397:   Nummer$ = LEFT$(temp$, LEN(temp$) - LEN(Nummer$)
) + Nummer$
[C62] 398:   PRINT Nummer$
[G96] 399:   i$ = "vpic /r " + QuellDatei$ + Nummer$ + ".lbm"
[Y81] 400:   SHELL i$
[ES2] 401:   GOSUB TotalScan
[662] 402:   IF izn = 1 THEN
[SW5] 403:     PaletteSpeichern ZielDatei$ + ".plt"
[ZS0] 404:   NEXT
[651] 405:   RETURN
[2F4] 406: '-----
[4Z5] 407: 'Total-Scan-Routine - holt sich alle
[LH6] 408: 'Objekte des Bildschirms und speichert
[UC5] 409: 'sie unter ZielDateiName + Nummer
[RR4] 410: '(ziell.dat, ziel2.dat etc) ab.
[2F4] 411: '-----
[EUI] 412: TotalScan:
[283] 413: IF DConvert% = 0 THEN
[N53] 414: 'Palette speichern
[AS3] 415: GET (0, 0)-(319, 10), Puffer
[SW5] 416: PaletteSpeichern ZielDatei$ + ".plt"
[SN3] 417: PUT (0, 0), Puffer, PSET
[801] 418: END IF
[P46] 419: FrameNummer% = 1 'Bildzähler auf Bild 1
[ZH3] 420: StartPunktSucher! = 0
[NB0] 421: DO
[075] 422:   '1. Obersten Startpunkt suchen
[RT3] 423:   GOSUB SucheStartPunkt
[2B6] 424:   IF FundZeile% = -1 AND FundSpalte% = -1 THEN
[KAL] 425:     EXIT DO
[VG3] 426:   XStart% = FundSpalte%
[VC3] 427:   YStart% = FundZeile%
[D96] 428:   '2. Rechten unteren Abschluß suchen
[ET6] 429:   'Der rechte untere Eckpunkt muß min-
[9A5] 430:   'destens jeweils 1 von XStart% und
[2U5] 431:   'YStart% entfernt sein, also diese
[SR2] 432:   'Ausgangswerte:
[SZ2] 433:   XEnde% = XStart% + 1
[XZ2] 434:   YEnde% = YStart% + 1
[AG5] 435:   AllesKlar1% = 0 'Prüfvariable
[DG5] 436:   AllesKlar2% = 0 'Prüfvariable
[D75] 437:   DO UNTIL POINT(XEnde%, YEnde%) <> 0
[0R2] 438:     XEnde% = XEnde% + 1
[5R2] 439:     YEnde% = YEnde% + 1
[HS0] 440:   LOOP
[248] 441:   DO UNTIL TestArea2%(XStart%, YStart%, XEnde%
, YEnde%)
[WN9] 442:     IF TestHorLine%(XStart%, YEnde%, XEnde%) = 1
AND YEnde% < 199 THEN
[5R2] 443:       YEnde% = YEnde% + 1
```

»grabber.bas« scannt ein Bild mehrfach ein und läßt diese ge-
speicherten Bilder anschließend als »Film« ablaufen.


```
[801] 444: END IF
[GH9] 445: IF TestVerLine%(XEnde%,YStart%,YEnde%)=1
AND XEnde% < 319 THEN
[0R2] 446: XEnde% = XEnde% + 1
[801] 447: END IF
[HS0] 448: LOOP
[9M5] 449: '3. Linken oberen Abschluß suchen
[Z38] 450: DO UNTIL TestAreal%(XStart%,YStart%,XEnde%
,YEnde%)
[FGA] 451: IF TestVerLine%(XStart%, YStart%, YEnde%)
<> 0 AND XStart% <> 0 THEN
[283] 452: XStart% = XStart% - 1
[801] 453: END IF
[CU9] 454: IF TestHorLine%(XStart%,YEnde%,XEnde%)<>0
AND YEnde% < 199 THEN
[5R2] 455: YEnde% = YEnde% + 1
[801] 456: END IF
[HS0] 457: LOOP
[T26] 458: '4. Ausschnitt speichern und dann vom
[3K3] 459: ' Bildschirm löschen
[AF6] 460: i$=ZielDatei$+LTRIM$(STR$(FrameNumber%))
[866] 461: IF DConvert% = 1 THEN i$ = SchreibDatei$
[3P5] 462: IF XEnde% <> 319 THEN XEnde% = XEnde%-1
[3R5] 463: IF YEnde% <> 199 THEN YEnde% = YEnde%-1
[N16] 464: IF XStart% <> 0 THEN XStart% = XStart%+1
[AM9] 465: SAVE XStart%,YStart%,XEnde%-XStart%+1,YEn
de%-YStart%+1,i$
[GT6] 466: LINE (XStart%,YStart%)-(XEnde%,YEnde%),0,B
F
[515] 467: FrameNumber% = FrameNumber% + 1
[ZD4] 468: IF DConvert% = 1 THEN EXIT DO
[KN6] 469: 'Bei DPaint-Konvertierung wird ehe Bild
[823] 470: 'für Bild geladen!
[HS0] 471: LOOP
[1S5] 472: IF PufferArt% = ClipPufferArt% THEN
[LL6] 473: SaveClipping Puffer(), ZielDatei$ + ".clp"
[801] 474: END IF
[651] 475: RETURN
[9L4] 476: '-----
[956] 477: 'Sprite-Scan-Routine - holt sich alle
[LE5] 478: 'Objekte mit gemeinsamen Zeilen-
[297] 479: 'Abschluß auf dem Bildschirm und speichert
[UC5] 480: 'sie unter ZielDateiname + Nummer
[RR4] 481: '(ziell.dat, ziel2.dat etc) ab.
[9L4] 482: '-----
[142] 483: SpriteScan:
[6L2] 484: FrameNumber% = 1
[KP6] 485: StartZeile% = 1 'Start bei zweiter Zeile
[076] 486: Endzeile% = 2 'Ende bei dritter Zeile
[074] 487: DO UNTIL StartZeile% > 199
[GW6] 488: '1. Oberen horizontalen Abschluß suchen
[074] 489: DO UNTIL StartZeile% > 199
[5NH] 490: IF TestHorLine%(0,StartZeile%,319)<>0 AND
TestHorLine%(0,StartZeile%-1,319)=0 OR Sta
rtZeile% > 199 THEN EXIT DO
[664] 491: StartZeile%=StartZeile%+1
[HS0] 492: LOOP
[R37] 493: '2. Unteren horizontalen Abschluß suchen
[074] 494: DO UNTIL StartZeile% > 199
[WTG] 495: IF TestHorLine%(0,Endzeile%,319)<>0 AND T
estHorLine%(0,Endzeile% + 1, 319) = 0 OR E
ndzeile% > 199 THEN EXIT DO
[6S3] 496: Endzeile% = Endzeile% + 1
[HS0] 497: LOOP
[HY4] 498: IF StartZeile% > 199 THEN EXIT DO
[337] 499: 'Jetzt die Prüfungen für die vertikalen
[722] 500: 'Abschlüsse
[FM2] 501: StartSpalte% = 1
[A22] 502: EndSpalte% = 2
[KF4] 503: DO UNTIL StartSpalte% > 319
[RC6] 504: '3. Linken vertikalen Abschluß suchen
[V36] 505: ' zwischen StartZeile und Endzeile
[KF4] 506: DO UNTIL StartSpalte% > 319
[UE9] 507: IF TestVerLine%(StartSpalte%, StartZeile
%, Endzeile%) <> 0 AND TestVerLine%(StartSp
alte% - 1, StartZeile%, Endzeile%) = 0 THEN
EXIT DO
[325] 508: StartSpalte% = StartSpalte% + 1
[HS0] 509: LOOP
[026] 510: '4. Rechten vertikalen Abschluß suchen
[V36] 511: ' zwischen StartZeile und Endzeile
[DK4] 512: EndSpalte% = StartSpalte% + 1
[RT3] 513: DO UNTIL EndSpalte% > 318
[GKM] 514: IF TestVerLine%(EndSpalte%, StartZeile%,
Endzeile%) <> 0 AND TestVerLine%(EndSpalte
% + 1, StartZeile%, Endzeile%) = 0 THEN EXI
T DO
[A44] 515: EndSpalte% = EndSpalte% + 1
[HS0] 516: LOOP
[824] 517: IF EndSpalte% > 319 THEN EXIT DO
[W26] 518: '5. Ausschnitt speichern und dann vom
[3K3] 519: ' Bildschirm löschen
[5U7] 520: i$ = ZielDatei$ + LTRIM$(STR$(FrameNumber
%)) + ".dat"
[EME] 521: SAVE StartSpalte%, StartZeile%, EndSpalte
% - StartSpalte% + 1, Endzeile% - StartZeil
e% + 1, i$
[0T9] 522: LINE (StartSpalte%, StartZeile%)-(EndSpal
```

```
te%, Endzeile%), 0, B
[515] 523: FrameNumber% = FrameNumber% + 1
[P02] 524: ' SOUND 9000, .1
[UL4] 525: StartSpalte% = EndSpalte% + 1
[DK4] 526: EndSpalte% = StartSpalte% + 1
[HS0] 527: LOOP
[A74] 528: StartZeile% = Endzeile% + 1
[P84] 529: Endzeile% = StartZeile% + 1
[HS0] 530: LOOP
[1S5] 531: IF PufferArt% = ClipPufferArt% THEN
[LL6] 532: SaveClipping Puffer(), ZielDatei$ + ".clp"
[651] 533: RETURN
[2F4] 534: '-----
[Y17] 535: 'Unterprogramm testet wo sich der nächste
[YZ5] 536: 'belegte Punkt beginnend oben links
[ZE6] 537: 'befindet und liefert die Ergebnisse in
[3U6] 538: 'den Variablen FundZeile% und FundSpalte%.
[P65] 539: 'Enthalten beide Variablen ihren
[0V5] 540: 'Ausgangswert -1, so befindet sich am
[5X6] 541: 'Bildschirm kein einziger Punkt ungleich
[EUS] 542: 'der Hintergrundfarbe (Farbe 0) mehr!
[2F4] 543: '-----
[923] 544: SucheStartPunkt:
[V92] 545: FundZeile% = -1
[RE2] 546: FundSpalte% = -1
[7A1] 547: Check% = 0
[VW2] 548: FOR Zeile% = 0 TO 199
[183] 549: FOR Spalte% = 0 TO 319
[ZF5] 550: IF POINT(Spalte%, Zeile%) <> 0 THEN
[P23] 551: FundZeile% = Zeile%
[LE3] 552: FundSpalte% = Spalte%
[WV5] 553: Zeile% = 200 'Bewirkt Abbruch beider
[RF5] 554: Spalte% = 320 'FOR-NEXT-Schleifen.
[801] 555: END IF
[ZS0] 556: NEXT
[ZS0] 557: NEXT
[651] 558: RETURN
[AH6] 559: 'Hängt einem Sprite-Array das nächste
[AG5] 560: 'Bildelement an und berechnet die
[CE7] 561: 'Array-Position für das nächste Element.
[EG6] 562: SUB AddFrame(X%,Y%,Breite%,Hoehe%,Array())
[CF6] 563: Groesse!=4+INT(((Breite%)*(8)+7)/8)*1*Hoe
he%
[3G9] 564: GET (X%, Y%)-(X%+Breite%-1,Y%+Hoehe%-1), A
rray(ClippingStart%)
[NX4] 565: FrameAnzahl% = FrameAnzahl% + 1
[F26] 566: Array(FrameAnzahl%) = ClippingStart%
[857] 567: ClippingStart%=ClippingStart%+(Groesse!/2)
+8
[HA1] 568: END SUB
[FL6] 569: 'Bereitet den gewählten Puffer für ein
[HD3] 570: 'neues Clipping vor.
[7B4] 571: SUB InitClipping (Array())
[Z21] 572: ERASE Array
[NE3] 573: REDIM Array(1 TO 32000)
[8G2] 574: FrameAnzahl% = 0
[733] 575: ClippingStart% = 100
[HA1] 576: END SUB
[8A7] 577: 'Function dient nur für den Auto-Grabber.
[K65] 578: 'Sie entfernt aus dem angegebenen
[X37] 579: 'Bilddateinamen Bildnummerierungen, damit
[PG6] 580: 'dann der zugehörige Palettendateiname
[0R6] 581: 'ermittelt werden kann. Die Dateieindung
[UT5] 582: '(.???) wird ebenfalls abgeschnitten,
[123] 583: 'falls sie vorhanden ist.
[1M5] 584: FUNCTION KillNumber$ (Dateiname$)
[ZY1] 585: i$ = "0123456789"
[AC2] 586: i1$ = Dateiname$
[WU0] 587: i2$ = ""
[L13] 588: FOR i = 1 TO LEN(i1$)
[5B8] 589: IF INSTR(i$, MID$(i1$, i, 1)) = 0 THEN i2
$ = i2$ + MID$(i1$, i, 1)
[ZS0] 590: NEXT
[447] 591: IF INSTR(i2$, ".") THEN i2$ = LEFT$(i2$, L
EN(i2$) - 4)
[HN2] 592: KillNumber$ = i2$
[R52] 593: END FUNCTION
[000] 594:
[WK1] 595: DEFMSG A-Z
[AB3] 596: SUB LadeBild (Datei$)
[NR6] 597: 'SUB-Routine zum Einlesen eines im .bld
[W86] 598: 'Format gespeicherten Bilds im SCREEN
[E25] 599: 'Modus 13 (VGA 320x200 in 256 Farben.
[4E5] 600: 'Aufruf mit Ladebild Dateiname$
[NY5] 601: 'ACHTUNG: Die Endungen für Bilddatei
[715] 602: '(.bld) und Palettendatei (.plt)
[Y47] 603: 'werden automatisch angefügt - Dateinamen
[NP4] 604: 'also ohne Extension angeben!
[2H5] 605: PaletteLaden 0, 255, 0, Datei$ + ".plt"
[W22] 606: DEF SEG = &H0000
[D85] 607: 'Beginn des VGA-Videospeichers
[ZY3] 608: Dateiname$ = Datei$ + ".bld"
[4V3] 609: 'Dateiname der Bilddatei
[013] 610: BLOAD Dateiname$, 0
[UZ5] 611: 'Direktes Laden der Bilddatei in den
[ZB6] 612: 'Videospeicher der VGA-Karte. Bild baut
[Z67] 613: 'sich somit bereits während dem Einlesen
[G43] 614: 'am Bildschirm auf.
```



```
[HA1] 615: END SUB
[TL1] 616: DEFINT A-Z
[FM4] 617: SUB LOAD (X%, Y%, Art%, Datei$)
[SB6] 618: 'SUB-Routine zum Laden eines mit SAVE-
[LS5] 619: 'SUBRoutine gespeicherten Bildes.
[LV6] 620: 'Puffer-Array Puffer muß im Hauptmodul
[DU5] 621: 'als COMMON SHARED deklariert sein!
[2K6] 622: 'Aufruf der Routine: LOAD x%, y%, Dateis
[3U5] 623: 'x% / y% = Position des Bildschirms
[D72] 624: 'Art% = PSET
[OT1] 625: '1 = PRESET
[011] 626: '2 = AND
[T01] 627: '3 = OR
[X31] 628: '4 = XOR
[8F5] 629: 'Dateis = Dateiname mit Extension
[HE4] 630: DEF SEG = VARSEG(Puffer(1))
[A45] 631: BLOAD Dateis, VARPTR(Puffer(1))
[4P6] 632: DEF SEG ' Stelle BASIC-Segment wieder her.
[M83] 633: Breite = Puffer(1) / 8
[SL2] 634: Hoehe = Puffer(2)
[BP2] 635: SELECT CASE Art%
[VR1] 636: CASE IS = 0
[H04] 637: PUT (X%, Y%), Puffer, PSET
[ZR1] 638: CASE IS = 1
[NB4] 639: PUT (X%, Y%), Puffer, PRESET
[3S1] 640: CASE IS = 2
[KP3] 641: PUT (X%, Y%), Puffer, AND
[7S1] 642: CASE IS = 3
[6M3] 643: PUT (X%, Y%), Puffer, OR
[BS1] 644: CASE IS = 4
[BU3] 645: PUT (X%, Y%), Puffer, XOR
[FS1] 646: CASE IS = 5
[EM4] 647: 'Hundert Prozent Durchlauf
[2D3] 648: FOR i = 1 TO 101 STEP 2
[PP4] 649: Puffer(1) = (i / 100) * Breite% * 8
[M44] 650: Puffer(2) = (i / 100) * Hoehe%
[UY3] 651: PUT (X, Y), Puffer%, PSET
[ZS0] 652: NEXT
[GS1] 653: END SELECT
[HA1] 654: END SUB
[KZ6] 655: 'SUB-Routine lädt ein Array, das mehrere
[6T5] 656: 'Bilder enthält in den angegebenen
[3A2] 657: 'Array-Puffer.
[BD5] 658: SUB LoadClipping (Array(), Dateis)
[344] 659: DEF SEG = VARSEG(Array(1))
[WT4] 660: BLOAD Dateis, VARPTR(Array(1))
[0V6] 661: DEF SEG 'Stelle BASIC-Segment wieder her.
[HA1] 662: END SUB
[WS5] 663: 'Lädt eine DAT-Bilddatei, bei der im
[IM6] 664: 'Array-Header Ziel-X und -Y eingetragen
[5T0] 665: 'ist.
[E34] 666: SUB LoadDataClip (Dateis)
[HE4] 667: DEF SEG = VARSEG(Puffer(1))
[A45] 668: BLOAD Dateis, VARPTR(Puffer(1))
[4P6] 669: DEF SEG ' Stelle BASIC-Segment wieder her.
[4Y1] 670: X = Puffer(1)
[7Y1] 671: Y = Puffer(2)
[8D2] 672: Art = Puffer(3)
[3F6] 673: 'hier ungenutzt - reserviert um in den
[DS6] 674: 'Clip-Bilddaten gleich mitzuspeichern, in
[1U5] 675: 'welcher Art das Bild gePUTet wird.
[U83] 676: Breite = Puffer(4) / 8
[VL2] 677: Hoehe = Puffer(5)
[5B4] 678: PUT (X%, Y%), Puffer(4), PSET
[HA1] 679: END SUB
[1R6] 680: 'Routine zum Laden einer Farbpalette im
[A75] 681: 'BASIC-PUT-Format (256 Farben, RGB).
[6T5] 682: 'Die beiden ersten Aufrufparameter
[9A5] 683: 'VonFarbe% und BisFarbe% geben den
[NK6] 684: 'Bereich der zu ladenden Palette an, der
[8U5] 685: 'in die aktuell vorhandene Palette
[Y96] 686: 'übertragen werden soll. Der dritte
[245] 687: 'Parameter - ZielBereichStart% -
[6B5] 688: 'gibt an ab welcher Position der
[TT5] 689: 'aktuellen Palette der neue Bereich
[082] 690: 'geladen wird.
[C15] 691: 'Um eine komplett neue Palette
[8N4] 692: '(alle 256 Farben) zu laden, ist
[HB5] 693: 'folgender Aufruf erforderlich:
[S94] 694: 'PaletteLaden 0,255,0,Dateis
[2MA] 695: SUB PaletteLaden (VonFarbe%, BisFarbe%, ZielBereichStart%, Dateis)
[W72] 696: DEF SEG = &H0000
[NA2] 697: BLOAD Dateis, 0
[Z75] 698: FOR Plt = VonFarbe% TO BisFarbe%
[LR5] 699: pltptr = ZielBereichStart% + Plt * 3
[RN2] 700: r = PEEK(pltptr)
[V13] 701: g = PEEK(pltptr + 1)
[N13] 702: b = PEEK(pltptr + 2)
[LZ1] 703: OUT 968, Plt
[7L1] 704: OUT 969, r
[6K1] 705: OUT 969, g
[RH1] 706: OUT 969, b
[ZS0] 707: NEXT
[091] 708: DEF SEG
[HA1] 709: END SUB
[Y07] 710: 'Speichert eine komplette VGA-Farbpalette
[0D3] 711: 'im BASIC Plt-Format.
[NW4] 712: SUB Palettespeichern (Dateis)
```

```
[W72] 713: DEF SEG = &H0000
[ZP2] 714: FOR Plt = 0 TO 255
[4D2] 715: pltptr = Plt * 3
[P42] 716: OUT &H3C7, Plt
[W02] 717: r = INP(&H3C9)
[602] 718: g = INP(&H3C9)
[WZ1] 719: b = INP(&H3C9)
[SF2] 720: POKE pltptr, r
[HV2] 721: POKE pltptr + 1, g
[6V2] 722: POKE pltptr + 2, b
[ZS0] 723: NEXT
[AV2] 724: BSAVE Dateis, 0, 768
[091] 725: DEF SEG
[HA1] 726: END SUB
[WK1] 727: DEFSEG A-Z
[H54] 728: 'Warten auf Tastendruck
[1L1] 729: SUB pause
[152] 730: DO UNTIL is = ""
[YL1] 731: is = INKEY$
[HS0] 732: LOOP
[N92] 733: DO UNTIL is <> ""
[YL1] 734: is = INKEY$
[HS0] 735: LOOP
[HA1] 736: END SUB
[TL1] 737: DEFINT A-Z
[XE6] 738: 'Routine zum (Um-)Positionieren einer
[BE2] 739: 'DAT-Bilddatei.
[W42] 740: SUB PosClip
[MR5] 741: INPUT "Bilddatei (.dat)"; QuellDateis
[L73] 742: Dateis = QuellDateis
[B84] 743: IF INSTR(Dateis, ".") = 0 THEN
[X93] 744: Dateis = Dateis + ".dat"
[801] 745: END IF
[U97] 746: PaletteLaden 0,255,0,KillNummer$(Dateis)+".plt"
[PN0] 747: CLS
[073] 748: LoadDataClip Dateis
[4Y1] 749: X = Puffer(1)
[7Y1] 750: Y = Puffer(2)
[8D2] 751: Art = Puffer(3)
[U83] 752: Breite = Puffer(4) / 8
[VL2] 753: Hoehe = Puffer(5)
[4R6] 754: Groesse! = 4 + INT(((Breite% * 8) + 7) / 8) * 1 * Hoehe% + 6
[K64] 755: PUT (X%, Y%), Puffer(4), XOR
[S56] 756: 'Abfrage der Pfeiltasten zum Bewegen des
[YB6] 757: 'Bildausschnitts. Achtung: Ausschnitt
[E26] 758: 'darf nicht über den Bildschirmrand
[YM5] 759: 'bewegt werden, da sonst Programm-
[BC5] 760: 'Absturz. Falls gewünscht, hier
[UL5] 761: 'Sicherheits-Abfragen einfügen.
[KK3] 762: DO UNTIL Wert$ = CHR$(13)
[072] 763: Wert$ = INKEY$
[VE6] 764: IF LEN(Wert$) > 1 THEN Wert$ = RIGHT$(Wert$, 1)
[ZT2] 765: SELECT CASE Wert$
[KU1] 766: CASE IS = "X"
[T82] 767: X% = X% - 1: links
[MU1] 768: CASE IS = "M"
[1F2] 769: X% = X% + 1: rechts
[FU1] 770: CASE IS = "H"
[342] 771: Y% = Y% - 1: rauf
[RU1] 772: CASE IS = "P"
[DH2] 773: Y% = Y% + 1: runter
[GS1] 774: END SELECT
[L12] 775: Puffer(1) = X%
[P12] 776: Puffer(2) = Y%
[K64] 777: PUT (X%, Y%), Puffer(4), XOR
[K64] 778: PUT (X%, Y%), Puffer(4), XOR
[865] 779: LOCATE 1, 1: PRINT "X="; X; "Y="; Y
[HS0] 780: LOOP
[HE4] 781: DEF SEG = VARSEG(Puffer(1))
[CN6] 782: BSAVE Dateis, VARPTR(Puffer(1)), Groesse!
[091] 783: DEF SEG
[4C5] 784: ' Stelle BASIC-Segment wieder her.
[HA1] 785: END SUB
[207] 786: SUB SAVE (X%, Y%, Breite%, Hoehe%, Dateis) STAT IC
[1L5] 787: 'SUB-Routine zum Speichern eines
[S76] 788: 'Bildausschnitts, der zuvor mit GET
[UU5] 789: 'in Array Puffer eingelesen wurde.
[LV6] 790: 'Puffer-Array Puffer muß im Hauptmodul
[906] 791: 'als COMMON SHARED deklariert sein!
[8K3] 792: 'Aufruf der Routine:
[UB5] 793: 'SAVE x%, y%, Breite%, Hoehe%, Dateis
[LL6] 794: 'x%/y%=Obere linke Ecke d. zu speichernden
[MX2] 795: ' Ausschnitts
[8X6] 796: 'Breite% / Hoehe% = Größe des Ausschnitts
[8F5] 797: 'Dateis = Dateiname mit Extension
[8P3] 798: SELECT CASE PufferArt%
[S44] 799: CASE IS = FramePufferArt%
[CF6] 800: Groesse! = 4 + INT(((Breite% * 8) + 7) / 8) * 1 * Hoehe% + 6
[NM6] 801: GET (X%, Y%) - (X% + Breite% - 1, Y% + Hoehe% - 1), Puffer
[HE4] 802: DEF SEG = VARSEG(Puffer(1))
```

»grabber.bas« scannt ein Bild mehrfach ein und läßt diese gespeicherten Bilder anschließend als »Film« ablaufen.


```
[357] 803: BSAVE Datei$+".put",VARPTR(Puffer(1)),Gr
oesse!
[456] 804: DEF SEG 'Stelle BASIC-Segment wieder her
[V24] 805: CASE IS = ClipPufferArt%
[G96] 806: AddFrame X%, y%,Breite%,Hoehe%,Puffer()
[YH3] 807: CASE IS = DataClipArt%
[7B6] 808: SaveDataClip X%,y%,Breite%,Hoehe%,Datei$
[GS1] 809: END SELECT
[HA1] 810: END SUB
[ZV6] 811: 'Speichert das Gesamtbildarray, das durch
[G57] 812: 'zufügen von Bildteilen mit der Routine
[R44] 813: 'AddFrame entstanden ist.
[4F5] 814: SUB SaveClipping (Array(), Datei$)
[YL5] 815: Groesse!=(ClippingStart%-100)*2+100
[C14] 816: Array(FrameAnzahl% + 1) = 0
[344] 817: DEF SEG = VARSEG(Array(1))
[TE6] 818: BSAVE Datei$, VARPTR(Array(1)), Groesse!
[456] 819: DEF SEG 'Stelle BASIC-Segment wieder her
[HA1] 820: END SUB
[EB6] 821: 'Ähnlich wie Routine SAVE, allerdings
[GB5] 822: 'werden hier X- und Y-Werte der
[Ø86] 823: 'Ausschnittposition im Array-Header
[XU2] 824: 'mitgespeichert.
[FN7] 825: SUB SaveDataClip (X%, y%, Breite%, Hoehe%,
Datei$)
[L12] 826: Puffer(1) = X%
[Pl2] 827: Puffer(2) = y%
[PV1] 828: Puffer(3) = 0
[L56] 829: 'reserviert für Art (PSET, PUT etc)
[PV2] 830: '(hier ungenutzt!)
[4R6] 831: Groesse! = 4 + INT(((Breite%)*(8+7)/8)*1*Hoehe
e%+6
[B47] 832: GET (X%,y%)-(X%+Breite%-1,y%+Hoehe%-1), Pu
ffer(4)
[HE4] 833: DEF SEG = VARSEG(Puffer(1))
[F37] 834: BSAVE Datei$+".dat",VARPTR(Puffer(1)),Gro
esse!
[US6] 835: DEF SEG 'Stelle BASIC-Segment wieder her
[HA1] 836: END SUB
[WK1] 837: DEFNG A-Z
[V76] 838: 'Funktion prüft, ob der rechteckige
[RX6] 839: 'Bildschirmausschnitt von oberem linken
[P96] 840: 'Eckpunkt (x1%,y1%) bis unterem rechten
[264] 841: 'Eckpunkt (x2%,y2%) folgende
[TP3] 842: 'Bedingungen erfüllt:
[PP6] 843: '1. Alle Bildpunkte des Rechteckrahmens
[KU5] 844: ' enthalten den Farbwert 0, oder der
[S35] 845: ' Bildschirmrand ist erreicht.
[FR6] 846: '2. Legen Sie innerhalb des Rahmens einen
[N56] 847: ' zweiten Rahmen, dessen vier Seiten
[S25] 848: ' exakt an den ersten Rahmen grenzen
[X26] 849: ' (also ein zweiter Rahmen im ersten
[4U5] 850: ' Rahmen), so muß jede Seite des
[816] 851: ' zweiten Rahmens mindestens einen
[DH6] 852: ' Bildpunkt enthalten der ungleich dem
[S24] 853: ' Farbwert 0 (entspricht
[X14] 854: ' Hintergrundfarbe) ist.
[N76] 855: ' Sind beide Bedingungen erfüllt, so
[516] 856: ' sind die Exkkoordinaten gefunden, die
[TW4] 857: ' ein Grafik-Objekt mit einem
[376] 858: ' rechteckigen Rahmen umschließen.
[AD5] 859: 'TestArea% liefert den Wert 0 wenn
[U25] 860: 'Bedingung nicht erfüllt ist, 1 wenn
[3N1] 861: 'erfüllt.
[LU5] 862: FUNCTION TestArea1% (x1%, y1%, x2%, y2%)
[HU6] 863: 'Erst mal davon ausgehen, daß Bedingung
[PB2] 864: 'erfüllt ist:
[522] 865: TestArea1% = 1
[NØ6] 866: '1. Erste Testphase - Prüfen ob alle
[917] 867: 'Bildpunkte des äußeren Rahmens der
[MM6] 868: 'Hintergrundfarbe (Farbe 0) entsprechen:
[T35] 869: '1.1 Obere äußere horizontale
[MR3] 870: ' Rahmenlinie testen
[832] 871: IF y1% <> 0 THEN
[2U2] 872: FOR i% = x1% TO x2%
[B74] 873: IF POINT(i%, y1% - 1) <> 0 THEN
[222] 874: TestArea1% = 0
[FA2] 875: EXIT FUNCTION
[8Ø1] 876: END IF
[E61] 877: NEXT i%
[8Ø1] 878: END IF
[VR5] 879: '1.2 Untere äußere horizontale
[MR3] 880: ' Rahmenlinie testen
[3C2] 881: IF y2% <> 199 THEN
[2U2] 882: FOR i% = x1% TO x2%
[RV3] 883: IF POINT(i%, y2%) <> 0 THEN
[222] 884: TestArea1% = 0
[FA2] 885: EXIT FUNCTION
[8Ø1] 886: END IF
[E61] 887: NEXT i%
[8Ø1] 888: END IF
[SW4] 889: '1.3 Linke äußere vertikale
[MR3] 890: ' Rahmenlinie testen
[732] 891: IF x1% <> 0 THEN
[7U2] 892: FOR i% = y1% TO y2%
```

```
[PW3] 893: IF POINT(x1%, i%) <> 0 THEN
[222] 894: TestArea1% = 0
[FA2] 895: EXIT FUNCTION
[8Ø1] 896: END IF
[E61] 897: NEXT i%
[8Ø1] 898: END IF
[E95] 899: '1.4 Rechte äußere vertikale
[MD3] 900: 'Rahmenlinie testen
[2C2] 901: IF x2% <> 319 THEN
[7U2] 902: FOR i% = y1% TO y2%
[UW3] 903: IF POINT(x2%, i%) <> 0 THEN
[222] 904: TestArea1% = 0
[FA2] 905: EXIT FUNCTION
[8Ø1] 906: END IF
[E61] 907: NEXT i%
[8Ø1] 908: END IF
[R25] 909: '2. Zweite Testphase - Prüfen ob min-
[166] 910: 'destens ein Bildpunkte des inneren
[A16] 911: 'Rahmens einem Farbwert ungleich der
[RG6] 912: 'Hintergrundfarbe (Farbe 0) entspricht.
[GL4] 913: '2.1 Obere innere horizontale
[MR3] 914: ' Rahmenlinie testen
[PX4] 915: Checker% = 0 'Prüfvariable
[KK3] 916: FOR i% = x1% + 1 TO x2% - 1
[574] 917: IF POINT(i%, y1% + 1) <> 0 THEN
[FT1] 918: Checker% = 1
[8Ø1] 919: END IF
[E61] 920: NEXT i%
[DX2] 921: IF Checker% = 0 THEN
[222] 922: TestArea1% = 0
[FA2] 923: EXIT FUNCTION
[8Ø1] 924: END IF
[LØ5] 925: '2.2 Untere innere horizontale
[MR3] 926: ' Rahmenlinie testen
[PX4] 927: Checker% = 0 'Prüfvariable
[KK3] 928: FOR i% = x1% + 1 TO x2% - 1
[F74] 929: IF POINT(i%, y2% - 1) <> 0 THEN
[FT1] 930: Checker% = 1
[8Ø1] 931: END IF
[E61] 932: NEXT i%
[DX2] 933: IF Checker% = 0 THEN
[222] 934: TestArea1% = 0
[FA2] 935: EXIT FUNCTION
[8Ø1] 936: END IF
[FC4] 937: '2.3 Linke innere vertikale
[MR3] 938: ' Rahmenlinie testen
[ET1] 939: Checker% = 0
[RK3] 940: FOR i% = y1% + 1 TO y2% - 1
[884] 941: IF POINT(x1% + 1, i%) <> 0 THEN
[FT1] 942: Checker% = 1
[8Ø1] 943: END IF
[ZSØ] 944: NEXT
[DX2] 945: IF Checker% = 0 THEN
[222] 946: TestArea1% = 0
[FA2] 947: EXIT FUNCTION
[8Ø1] 948: END IF
[6K4] 949: '2.4 Rechte innere vertikale
[MR3] 950: ' Rahmenlinie testen
[ET1] 951: Checker% = 0
[RK3] 952: FOR i% = y1% + 1 TO y2% - 1
[K84] 953: IF POINT(x2% - 1, i%) <> 0 THEN
[FT1] 954: Checker% = 1
[8Ø1] 955: END IF
[ZSØ] 956: NEXT
[DX2] 957: IF Checker% = 0 THEN
[222] 958: TestArea1% = 0
[FA2] 959: EXIT FUNCTION
[8Ø1] 960: END IF
[R52] 961: END FUNCTION
[AØ6] 962: 'Wie FUNCTION TestArea1%, allerdings
[Y55] 963: 'wird bei der Prüfung die linke
[2F5] 964: 'vertikale Rahmenseite (innen und
[H35] 965: 'außen) nicht berücksichtigt!
[PUS] 966: FUNCTION TestArea2% (x1%, y1%, x2%, y2%)
[HU6] 967: 'Erst mal davon ausgehen, daß Bedingung
[PB2] 968: 'erfüllt ist:
[722] 969: TestArea2% = 1
[NØ6] 970: '1. Erste Testphase - Prüfen ob alle
[917] 971: 'Bildpunkte des äußeren Rahmens der
[8M6] 972: 'Hintergrundfarbe (Farbe 0) entsprechen.
[T35] 973: '1.1 Obere äußere horizontale
[MR3] 974: ' Rahmenlinie testen
[832] 975: IF y1% <> 0 THEN
[2U2] 976: FOR i% = x1% TO x2%
[B74] 977: IF POINT(i%, y1% - 1) <> 0 THEN
[K74] 978: TestArea2% = 0: 'SOUND 1000, 3
[FA2] 979: EXIT FUNCTION
[8Ø1] 980: END IF
[E61] 981: NEXT i%
[8Ø1] 982: END IF
[VR5] 983: '1.2 Untere äußere horizontale
[MR3] 984: ' Rahmenlinie testen
[3C2] 985: IF y2% <> 199 THEN
[2U2] 986: FOR i% = x1% TO x2%
[RV3] 987: IF POINT(i%, y2%) <> 0 THEN
```



```
[422] 988: TestArea2% = 0
[FA2] 989: EXIT FUNCTION
[801] 990: END IF
[E61] 991: NEXT i%
[801] 992: END IF
[SW4] 993: '1.3 Linke äußere vertikale
[MR3] 994: ' Rahmenlinie testen
[6F4] 995: ' Entfällt bei TestArea2 !
[E95] 996: '1.4 Rechte äußere vertikale
[MR3] 997: ' Rahmenlinie testen
[2C2] 998: IF x2% <> 319 THEN
[7U2] 999: FOR i% = Y1% TO y2%
[UW3] 1000: IF POINT(x2%, i%) <> 0 THEN
[422] 1001: TestArea2% = 0
[FA2] 1002: EXIT FUNCTION
[801] 1003: END IF
[E61] 1004: NEXT i%
[801] 1005: END IF
[RZ5] 1006: '2. Zweite Testphase - Prüfen ob min-
[166] 1007: 'destens ein Bildpunkte des inneren
[AL6] 1008: 'Rahmens einem Farbwert ungleich der
[RG6] 1009: 'Hintergrundfarbe (Farbe 0) entspricht.
[GL4] 1010: '2.1 Obere innere horizontale
[MR3] 1011: ' Rahmenlinie testen
[PX4] 1012: Checker% = 0 'Prüfvariable
[KK3] 1013: FOR i% = x1% + 1 TO x2% - 1
[574] 1014: IF POINT(i%, Y1% + 1) <> 0 THEN
[FT1] 1015: Checker% = 1
[801] 1016: END IF
[E61] 1017: NEXT i%
[DX2] 1018: IF Checker% = 0 THEN
[422] 1019: TestArea2% = 0
[FA2] 1020: EXIT FUNCTION
[801] 1021: END IF
[L05] 1022: '2.2 Untere innere horizontale
[MR3] 1023: ' Rahmenlinie testen
[PX4] 1024: Checker% = 0 'Prüfvariable
[KK3] 1025: FOR i% = x1% + 1 TO x2% - 1
[F74] 1026: IF POINT(i%, y2% - 1) <> 0 THEN
[FT1] 1027: Checker% = 1
[801] 1028: END IF
[E61] 1029: NEXT i%
[DX2] 1030: IF Checker% = 0 THEN
[422] 1031: TestArea2% = 0
[FA2] 1032: EXIT FUNCTION
[801] 1033: END IF
[FC4] 1034: '2.3 Linke innere vertikale
[MR3] 1035: ' Rahmenlinie testen
[6F4] 1036: ' Entfällt bei TestArea2 !
[6K4] 1037: '2.4 Rechte innere vertikale
[MR3] 1038: ' Rahmenlinie testen
[ET1] 1039: Checker% = 0
[RK3] 1040: FOR i% = Y1% + 1 TO y2% - 1
[K84] 1041: IF POINT(x2% - 1, i%) <> 0 THEN
[FT1] 1042: Checker% = 1
[801] 1043: END IF
[E61] 1044: NEXT i%
[DX2] 1045: IF Checker% = 0 THEN
[422] 1046: TestArea2% = 0
[FA2] 1047: EXIT FUNCTION
[801] 1048: END IF
[R52] 1049: END FUNCTION
[SY5] 1050: 'Prüft, ob die horizontale Linie von
[8P5] 1051: '(XStart%,y%)-(XEnde%,y%) mindestens
[4T4] 1052: 'einen Bildpunkt ungleich der
[926] 1053: 'Hintergrundfarbe (Farbe 0) enthält.
[116] 1054: 'TestHorLine% liefert den Wert 0 wenn
[U25] 1055: 'Bedingung nicht erfüllt ist, 1 wenn
[3N1] 1056: 'erfüllt.
[YX6] 1057: FUNCTION TestHorLine% (XStart%, y%, XEnde%)
[3G3] 1058: IF y% = 199 OR y% = 0 THEN
[465] 1059: TestHorLine% = 1: EXIT FUNCTION
[801] 1060: END IF
[4M2] 1061: TestHorLine% = 0
[084] 1062: FOR i% = XStart% TO XEnde%
[4S3] 1063: IF POINT(i%, y%) <> 0 THEN
[5M2] 1064: TestHorLine% = 1
[DZ1] 1065: i% = XEnde% + 1
[801] 1066: END IF
[ZS0] 1067: NEXT
[HN6] 1068: GET (XStart%,y%)-(XEnde%,y%),TempPuffer%
[KT4] 1069: LINE (XStart%, y%)-(XEnde%, y%), 1
[5X1] 1070: 'SOUND 600, .1
[XW5] 1071: PUT (XStart%, y%), TempPuffer%, PSET
[R52] 1072: END FUNCTION
[LG5] 1073: 'Prüft, ob die vertikale Linie von
[7T5] 1074: '(x%,YStart%)-(x%-YEnde%) mindestens
[4T4] 1075: 'einen Bildpunkt ungleich der
[926] 1076: 'Hintergrundfarbe (Farbe 0) enthält.
[316] 1077: 'TestVerLine% liefert den Wert 0 wenn
[U25] 1078: 'Bedingung nicht erfüllt ist, 1 wenn
[3N1] 1079: 'erfüllt.
[SV6] 1080: FUNCTION TestVerLine% (X%, YStart%, YEnde%)
[DF3] 1081: IF X% = 319 OR X% = 0 THEN
[265] 1082: TestVerLine% = 1: EXIT FUNCTION
```

```
[801] 1083: END IF
[2M2] 1084: TestVerLine% = 0
[584] 1085: FOR i% = YStart% TO YEnde%
[1S3] 1086: IF POINT(X%, i%) <> 0 THEN
[3M2] 1087: TestVerLine% = 1
[GZ1] 1088: i% = YEnde% + 1
[801] 1089: END IF
[ZS0] 1090: NEXT
[ZM6] 1091: GET (X%, YStart%)-(X%, YEnde%), TempPuffer%
[CY4] 1092: LINE (X%, YStart%)-(X%, YEnde%), 1
[7V5] 1093: PUT (X%, YStart%), TempPuffer%, PSET
[R52] 1094: END FUNCTION
[TL1] 1095: DEFINT A-Z
[T97] 1096: 'SUB-Routine für Perspektive-Zoom. Zoomt
[F66] 1097: 'ein Bildarray von seinem Mittelpunkt
[BF5] 1098: 'heraus um den angegebenen Faktor.
[ZV2] 1099: 'Aufrufparameter:
[R85] 1100: ' Array() = Array mit den Bilddaten
[HT5] 1101: ' ZoomFaktor! = Zoomfaktor in n-fach
[7R4] 1102: ' (gültige Werte ab 1.0)
[EK3] 1103: ' X = X-Koordinate der
[005] 1104: ' Zielbildschirm-Position
[GK3] 1105: ' Y = Y-Koordinate der
[005] 1106: ' Zielbildschirm-Position
[BB6] 1107: ' BildNeuBreite = Neue Breite des Bildes
[HN5] 1108: ' BildNeuHöhe = Neue Höhe des Bildes
[S56] 1109: ' (die Ausgangshöhe und Breite wird
[AB5] 1110: ' automatisch vom Array-Header
[8C2] 1111: ' ermittelt.)
[ZAU] 1112: SUB VgaZoom (Array() AS INTEGER, XZoomFakto
r AS SINGLE, YZoomFaktor AS SINGLE, X AS IN
TEGER, y AS INTEGER, BildNeuBreite AS INTEG
ER, BildNeuHoehe AS INTEGER)
[HA4] 1113: BildAltBreite = Array(1) / 8
[PV5] 1114: 'BildBreite aus Array(1) ermitteln
[WP3] 1115: BildAltHoehe = Array(2)
[YD5] 1116: 'Bildhöhe aus Array(2) ermitteln
[9R7] 1117: AusschnittBreite=BildAltBreite/XZoomFaktor!
[GX5] 1118: 'bestimmt die Breite des Bildbereichs
[834] 1119: 'der herausgezogen wird.
[8E7] 1120: AusschnittHoehe=BildAltHoehe/YZoomFaktor!
[PG5] 1121: 'bestimmt die Höhe des Bildbereichs
[U14] 1122: 'der herausgezogen wird
[HL7] 1123: StartZeile=(BildAltHoehe-AusschnittHoehe)/2
[TP4] 1124: 'bestimmt die Startzeile des
[SW4] 1125: 'herauszuzoomenden Bereichs
[8M5] 1126: '(gerechnet ab der 0,0 Koordinate des
[G32] 1127: ' Bild-Arrays)
[N38] 1128: StartSpalte=(BildAltBreite-AusschnittBreite
)/2
[HU4] 1129: 'bestimmt die Startspalte des
[SW4] 1130: 'herauszuzoomenden Bereichs
[8M5] 1131: '(gerechnet ab der 0,0 Koordinate des
[G32] 1132: ' Bild-Arrays)
[344] 1133: DEF SEG = VARSEG(Array(1))
[846] 1134: 'Segment auf Bildarray-Beginn setzen
[M77] 1135: FaktorHoehe!=AusschnittHoehe/BildNeuHoehe
[HK5] 1136: 'Anhand dieses Faktors erfolgt die
[R56] 1137: 'Abbildung der Quellbild-Zeilenzahl
[UB5] 1138: 'auf die Zielbild-Zeilenzahl.
[GK5] 1139: IStart = 0: IEnde = BildNeuHoehe - 1
[GZ4] 1140: 'Die folgende "Zeilen"-Schleife
[PH6] 1141: 'urchläuft das "Zielbild" zeilenweise,
[AR5] 1142: 'beginnend bei Zeile 1, endend bei der
[3D6] 1143: 'Bildhöhe (=maximale Zeilenanzahl des
[5X1] 1144: 'Zielbilds)
[UU3] 1145: FOR i = IStart TO IEnde
[EK2] 1146: ZielZeile = X + i
[WZ5] 1147: 'Zur ZielZeile wird die gewünschte
[774] 1148: 'X-Startposition addiert
[9G6] 1149: UrsprungsZeilenNR = i * FaktorHoehe! + 1
[606] 1150: 'Errechnen der Ursprungszeile durch
[T76] 1151: 'Multiplizieren der aktuellen Zeile
[HY2] 1152: 'mit FaktorHoehe!.
[U56] 1153: ic1! = StartZeile + UrsprungsZeilenNR
[C53] 1154: ic2! = BildAltBreite
[X13] 1155: Poker! = ic1! * ic2! + 4
[546] 1156: 'Position des Zeilenbeginns im Array
[CR1] 1157: 'errechnen
[MS7] 1158: FaktorBreite!=AusschnittBreite/BildNeuBrei
te
[HK5] 1159: 'Anhand dieses Faktors erfolgt die
[9P6] 1160: 'Abbildung der Quellbild-Spaltenanzahl
[MF5] 1161: 'auf die Zielbild-Spaltenanzahl.
[BR5] 1162: AStart = 0: AEnde = BildNeuBreite - 1
[H55] 1163: 'Bereich für Spalten-Schleife
[ZM5] 1164: PokerStart! = Poker! + StartSpalte
[US3] 1165: FOR a = AStart TO AEnde
[8E6] 1166: UrsprungsSpaltenNR = a * FaktorBreite!
[TKA] 1167: PSET (a + y, ZielZeile), PEEK(PokerStart
! + UrsprungsSpaltenNR)
[ZS0] 1168: NEXT
[ZS0] 1169: NEXT
[S17] 1170: DEF SEG 'Segment zurück auf BASIC setzen
[HA1] 1171: END SUB
```

»grabber.bas« scannt ein Bild mehrfach ein und läßt diese ge-
speicherten Bilder anschließend als »Film« ablaufen.

Zoom mit Perspektive

Perspektivisches Zoomen dürfen Sie keineswegs mit dem Vergrößern oder Verkleinern von Bildausschnitten verwechseln. Was manuell eine Menge Arbeit bereitet, läßt sich mit einer Spezialfunktion einfach und schnell realisieren.

Einer der interessantesten Effekte und zudem eine der größten Herausforderungen für den Grafikprogrammierer sind dreidimensionale Animationsabläufe, wie sie beispielsweise bei vielen modernen Computerspielen zu finden sind. Stellen Sie sich vor, eine Figur soll sich aus dem Bildschirm heraus auf den Betrachter zu bewegen. Wie viele Einzelbilder brauchen Sie dafür? Prinzipiell wird nur die Vorwärtsbewegung der Figur benötigt – abwechselnd linkes und rechtes Bein vor. Damit die Figur näherkommt, wird sie nach jedem Schrittablauf einfach am Bildschirm vergrößert. Solche 3D-Animationen werden durch einfaches Vergrößern und Verkleinern erreicht, perspektivisches Zoomen ist nicht erforderlich. Anders ist das bei der Darstellung in Bild 1. Hier ist ein Gebirgs Panorama zu sehen.

Ziel des Perspektive-Zooms soll sein, einen Fernglas-Effekt zu erreichen. Der Betrachter soll weit in die Darstellung hineinzoomen können. Das Originalbild bleibt in seiner Größe unverändert. Das Perspektive-Zentrum entspricht der Bildmitte – also etwa der Bergspitze in der Mitte von Bild 1. Beim Hineinzoomen in dieses Bild wird folglich die Bergspitze immer größer, die äußeren Bildinformationen werden mit jedem Zoom-Schritt aus dem Bildschirmrand hinausgeschoben. Zunächst etwas Theorie zur Zoom-Routine. Bild 2 zeigt den

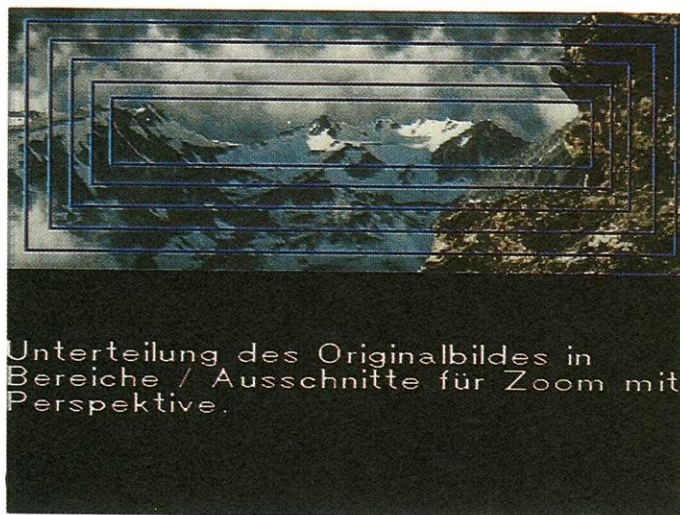


Bild 2. Theorie des Perspektive-Zooms – Ausschnitte des Gesamtbildes werden auf Originalbildgröße gebracht.

prinzipiellen Zoom-Vorgang. Im Quellbild sind mehrere blaue Rahmen zu sehen. Nehmen Sie beispielsweise den Ausschnitt, der durch den zweiten Rahmen eingegrenzt ist, und vergrößern ihn auf Originalbildgröße, so wandert die Darstellung einen Schritt »näher« zum Betrachter. Jeder blaue Rahmen entspricht also einem Zoom-Schritt.

Noch deutlicher wird der Effekt in Bild 3. Sie sehen, welche Bilder sich ergeben, wenn viermal gezoomt wird. Der Rechengang zum Zoomen eines Bildes ist zwar nicht sonderlich kompliziert, allerdings sehr rechenintensiv – ein Perspektive-Zoom in Echtzeit läßt sich mit Basic kaum bewerkstelligen. Also bietet es sich an, die erforderlichen Einzelbilder einfach vorzuberechnen und dann von der Festplatte einzeln nachzuladen. Die Bilder 1 bis 3 wurden im VGA-Modus 640×480/256 angefertigt. Betrachten Sie Bild 4 – dort wurde das Gesamt-Panoramabild in die Auflösung 320×200/256 umgerechnet. Beim Gesamtbild ist die Darstellungsqualität noch äußerst gut, wenn Sie berücksichtigen, daß die Auflösung auf weniger als ein Viertel reduziert wurde. Wird der innere Ausschnitt von Bild 4 auf Vollbildformat vergrößert, entsteht das Ergebnis von Bild 5. Die Vergrößerung der Einzelpixel ist deutlich erkennbar, von »Bildqualität« kann kaum die Rede sein. Den Lösungs-



Bild 1. Ein Gebirgs Panorama – Ausgangsbild für den intensiven Test der Zoom-Routine des Auto-Grabbers

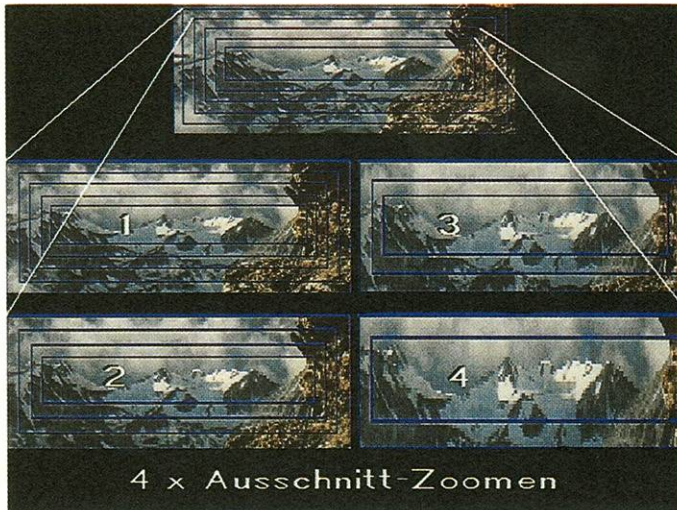


Bild 3. Bei dieser Fotomontage wurden vier Innenbereiche des Quellbildes herausgezoomt.

ansatz bringt Bild 6. Dort wurde der innere Ausschnitt im Grafikmodus $640 \times 480/256$ gezoomt. Das Ergebnis ist deutlich besser als bei Bild 5. Schließlich zeigt Bild 7, was passiert, wenn Sie Bild 6 in das $320 \times 200/256$ -Format umrechnen. Das Ergebnis in Bild 7 entspricht Bild 5, die Darstellungsqualität ist allerdings deutlich besser. Damit wird folgendes klar:

Das Zoomen eines Bildbereichs sollte in der Regel stets auf dem Originalbild basieren. Der zweite Zoom-Schritt darf also nicht aus dem Ergebnis des ersten errechnet werden.

Wenn Sie mit größeren Bildern als $320 \times 200/256$ in Basic arbeiten möchten, müssen Sie diese Bilder in der Regel ins 320×200 -Format konvertieren (eine Aufgabe der zu Beginn der Werkstatt vorgestellten Grafikviewer und -konverter). Bei der Konvertierung haben Sie die Wahl, entweder das Komplettbild auf die kleinere Auflösung zu übertragen oder nur einen Ausschnitt davon. Damit existiert ein Trick, der tiefes Zoomen in Bilder gestattet, ohne daß viel von der Darstellungsqualität verlorengeht. Vorausgesetzt wird dabei natürlich, daß ein Bild in höherer Auflösung als 320×200 vorliegt. Gehen Sie folgendermaßen vor:

- Rechnen Sie das hochauflösende Bild per Zeichenprogramm oder Grafik-Utility in die 320×200 -Darstellung um. Anschließend führen Sie an diesem Bild so viele Zoom-Schritte durch, wie ohne deutlichen Qualitätsverlust realisierbar sind, und speichern die Einzelbilder. Merken Sie

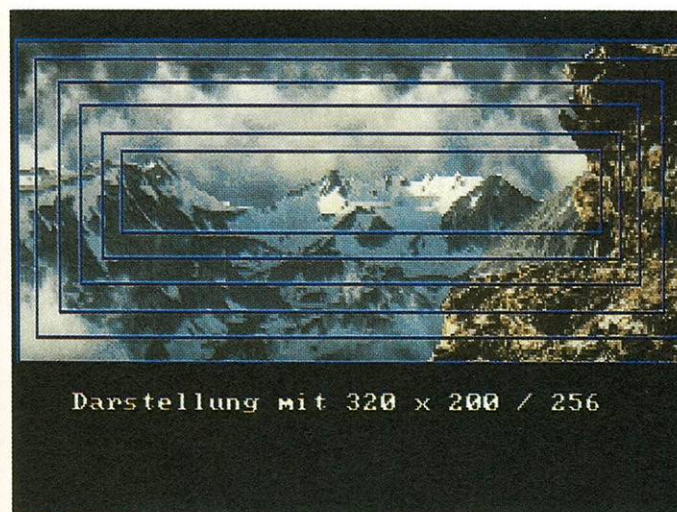


Bild 4. Auch bei der Auflösung $320 \times 200/256$ sind gute fotorealistische Zooms machbar.

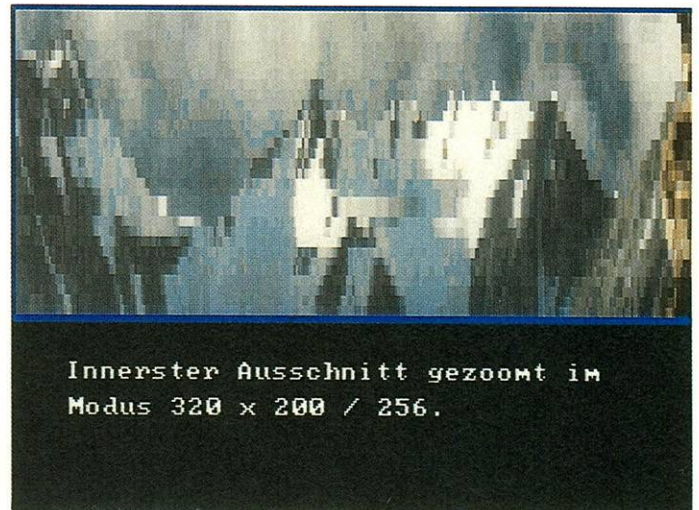


Bild 5. Vergrößert man den inneren Ausschnitt im Grafikmodus $320 \times 200/256$, ist der Verlust an Darstellungsqualität eindeutig zu groß.

sich die Bildposition (den Zoom-Ausschnitt), bei der die Darstellungsqualität deutlich nachläßt.

- Konvertieren Sie jetzt aus dem hochauflösenden Originalbild lediglich den Ausschnitt ins 320×200 -Format, den Sie sich am Ende von Schritt 1 notiert haben. Jetzt liegt ein Ausschnitt in brauchbarer Qualität für die nächsten Zoom-Schritte vor.

- Bei einem $640 \times 480/256$ -Bild reichen in der Regel zwei konvertierte Quell-Ausschnitte aus, höhere Auflösungen erlauben entsprechend weitere.

Sämtliche Zoom-Operationen mit Perspektive in der $320 \times 200/256$ -Darstellung können mit einer Spezialfunktion des Auto-Grabbers aus dem vorhergehenden Beitrag erledigt werden. Im Hauptmenü des Auto-Grabbers ist dafür die Option »Zoom-Perspektive« zu wählen. Anschließend müssen Sie mehrere Einstellungen vornehmen (Bild 8):

- Anzahl Frames: Hier geben Sie an, wie viele Zoom-Schritte (Einzelbilder) benötigt werden.
- X-Zoom: der Zoom-Faktor für die Bildbreite. Er legt fest, in welcher Schrittweite der Zoom erfolgen soll. Der Zoom-Faktor wird mit jedem Einzelbild um den angegebenen Wert erhöht. Zulässige Faktoren beginnen bei 0.1.
- Y-Zoom: wie X-Zoom, allerdings auf die Bildhöhe bezogen. Höhen- und Breiten-Zoom-Faktor können getrennt angegeben werden, damit sich Korrekturen am Darstellungsverhältnis durchführen lassen. Schließlich ist es nicht Sinn



Bild 6. Beim 640×480 -Modus ist die Qualität des inneren Ausschnitts nach dem Zoomen noch recht passabel.

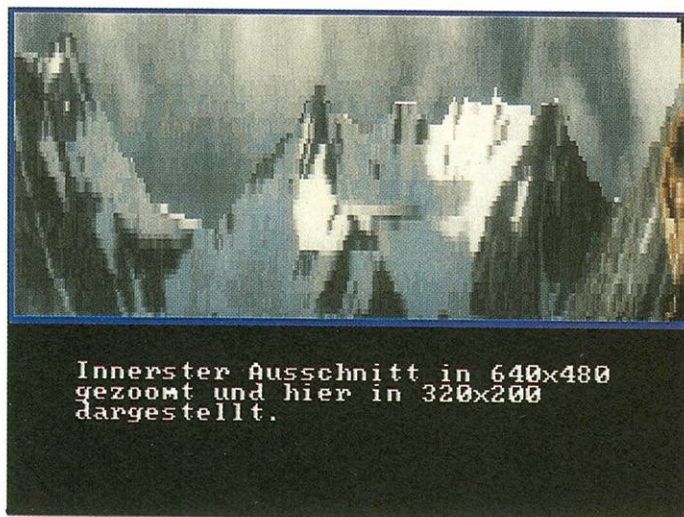


Bild 7. Ein deutlicher Unterschied zu Bild 5 – dank höherer Quellauflösung wird die Darstellung hier besser.

und Zweck eines Zooms, daß Bilder sich verformen beziehungsweise in Breite oder Höhe ausgedehnt erscheinen.

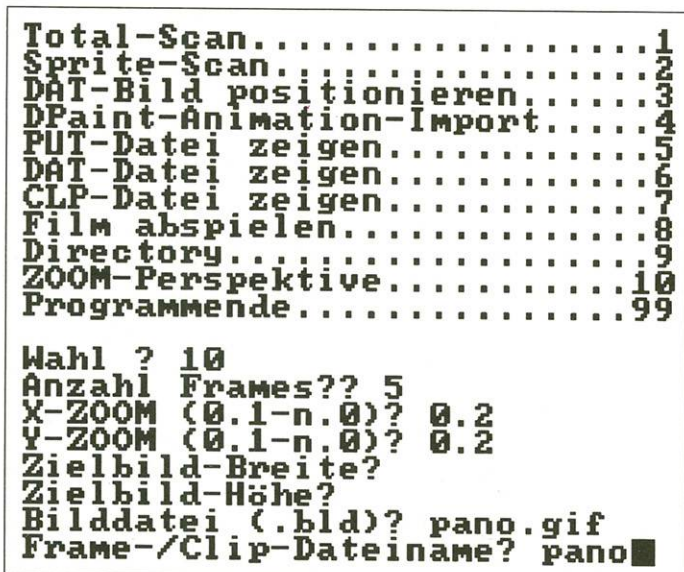


Bild 8. Diese Parameter müssen vor einem Zoom-Vorgang beim Auto-Grabber eingestellt werden.

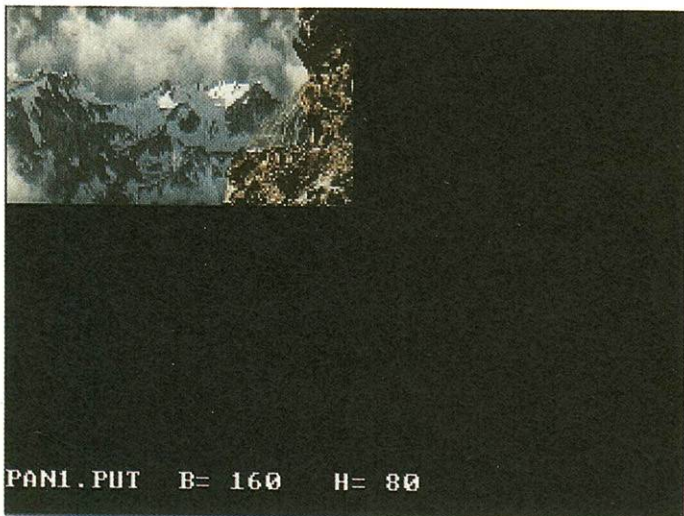


Bild 9. Verkleinerung eines Originalbilds mit 320 x 100 Bildpunkten auf 160 x 80 Bildpunkte



Bild 10. Selbst bei Vergrößerung auf volles Bildformat ist die Darstellungsqualität noch passabel.

- Zielbildbreite: Breite des Zielbilds. Die Zoom-Routine kann gleichzeitig zum Zoomen auch die Gesamtdarstellung vergrößern oder verkleinern. Je kleiner das Zielbild, desto schneller geht das Zoomen vonstatten, da weniger Einzelbildpunkte zu berechnen sind. Wird bei Zielbildbreite kein Wert angegeben, so behält die Zoom-Routine die Größe des Originalbildes bei.

- Zielbildhöhe: Einstellung der Höhe des Zielbilds. Auch hier wird die Originalhöhe beibehalten, wenn auf eine Angabe verzichtet wird.

- Bilddatei: Name der zu zoomenden Bilddatei

- Frame-/Clip-Dateiname: ein beliebiger Dateiname. Beachten Sie, daß er nicht zu lang ist, damit eine Numerierung angefügt werden kann. Bilddateien werden hier ausschließlich im PUT-Format gespeichert. Ein anderes Format würde keinen Sinn machen, da die Einzelbilder ja stets die gleiche Größe haben und auch an jeweils der gleichen Bildschirmposition erscheinen müssen.

Wird die letzte Eintragung mit [Enter] bestätigt, beginnt der Zoom-Vorgang. Abhängig von der gewünschten Bilderanzahl dauert es eine Weile. Sind alle Einzelbilder gespeichert, aktiviert der Auto-Grabber automatisch seine Abspiel-Funktion, damit Sie das Zoom-Ergebnis sofort am Bildschirm begutachten können.

Um optimale Ergebnisse zu erreichen (insbesondere, wenn das Zielbild auch noch in der Größe verändert wird), sollten

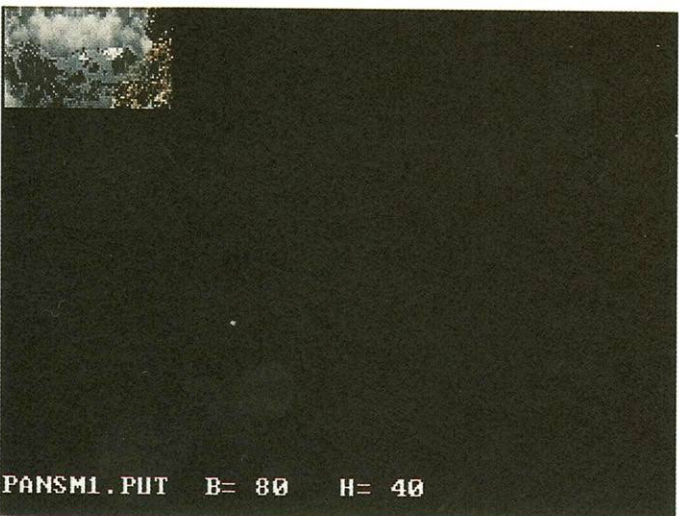


Bild 11. Wesentliche Details im Bild sind selbst bei extremer Verkleinerung noch erkennbar.

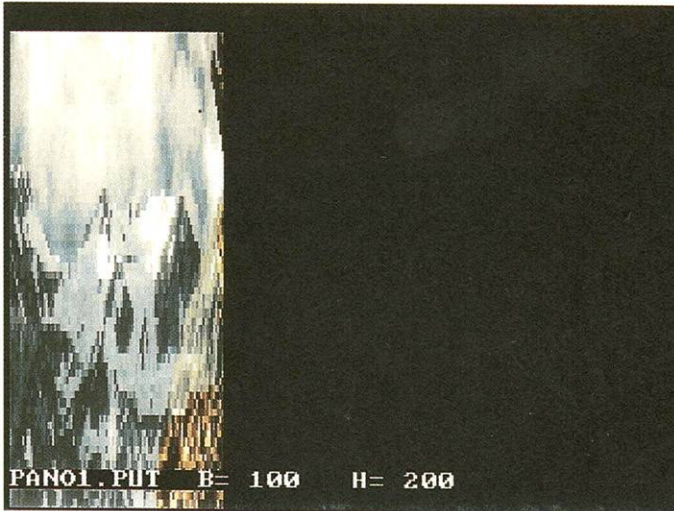


Bild 12. Die Zoom-Routine des Auto-Grabbers gestattet, in einem Zug zu zoomen und die Bildgröße beliebig zu verändern.

Sie mit unterschiedlich großen y- und x-Zoom-Werten experimentieren. Bei der Auflösung $320 \times 200/256$ gibt es kein korrektes Bildschirm-Darstellungsverhältnis – Einzelbildpunkte sind unterschiedlich breit und hoch.

Mit »Zoom-Perspektive« können Sie Bilder auch lediglich vergrößern oder verkleinern. Dazu wird als y- und x-Zoomfaktor jeweils 0 angegeben und dann die gewünschte Zielgröße bei Zielbildbreite und Zielbildhöhe eingetragen. Bei solchen Operationen sollten Sie bei »Anzahl Frames« stets den Wert 1 eintragen, denn es macht keinen Sinn, die Größenkonvertierung mehrfach durchzuführen, da stets das gleiche Ergebnis entsteht.

Die Bilder 9 bis 12 zeigen mehrere Beispielanwendungen der Auto-Grabber Zoom-Routine:

Ausgangsbild für die Bilder 9 bis 12 war das Gebirgspanorama aus Bild 4 (hier allerdings ein wenig in der Höhe gestaucht). In Bild 9 wurde das Quellbild ohne Zoom-Effekt (x-Faktor und y-Faktor jeweils 0) in die Bildgröße 160×80 umgerechnet. Bild 10 zeigt, daß selbst bei maximaler Vergrößerung auf volle Bildschirmgröße eine eindrucksvolle Darstellung erhalten bleibt. Bild 11 minimiert die Größe ins Extreme. Immerhin zeigt Bild 11, daß die Zoom- beziehungsweise Größenveränderungs-Routine auch bei Minidarstellung ausreichend viele Bilddetails beibehält. Bei Bild 12 wurde das Bild extrem in der Höhe gestreckt und gleichzeitig um den Faktor 0.5 gezoomt. Die Beispiele sollen

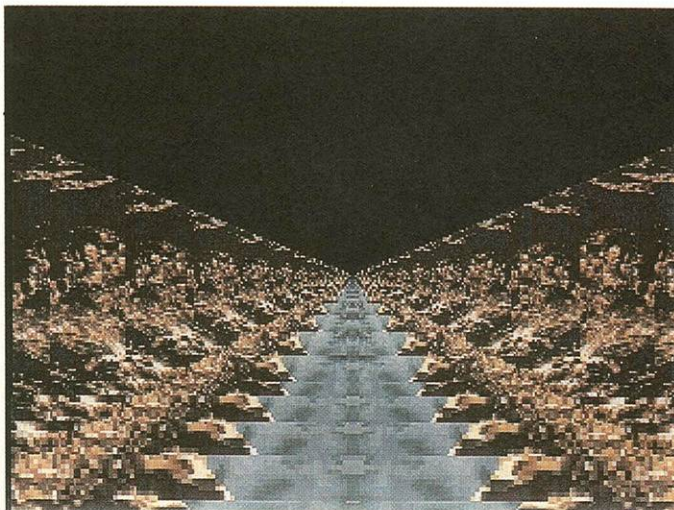


Bild 13. Ein dreidimensionaler fotorealistischer Tunnel – eines von zwei Bildern für einen fantastischen Animationseffekt in voller Bildschirmgröße

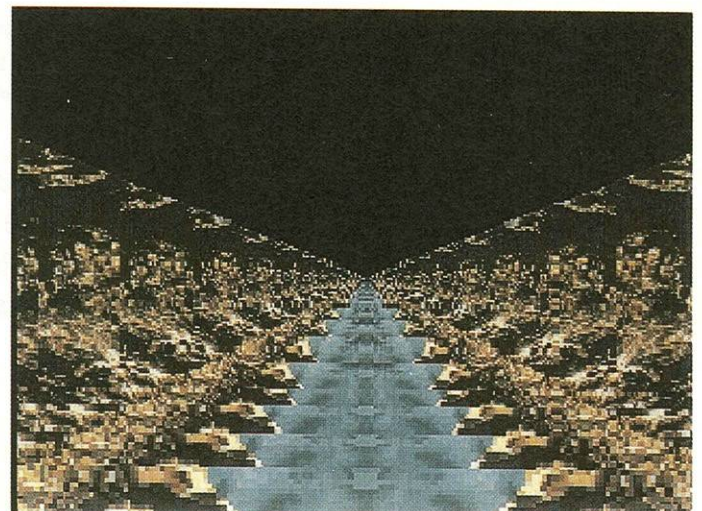


Bild 14. Fast wie Bild 13 – allerdings mit einem entscheidenden Unterschied

Ihnen verdeutlichen, welche Veränderungen sich in Verbindung mit Vergrößern/Verkleinern und Zoomen ergeben – viel Vergnügen bei eigenen Experimenten.

Die Varianten der Zoom-Funktionen sind damit noch längst nicht erschöpft. Nur soviel zum Schluß: Ein weiterer Trick ist aus den Bildern 13 und 14 ersichtlich. Beide sehen auf den ersten Blick identisch aus, wer genau hinschaut, wird allerdings Unterschiede erkennen. Natürlich wurden beide Bilder aus ein und demselben Quellbild unter Einsatz der Zoom-Routine berechnet. Isoliert betrachtet, machen die Bilder kaum etwas her, spielt man sie allerdings schnell hintereinander ab, so entsteht ein äußerst eindrucksvoller 3D-Effekt: der Flug durch einen fotorealistischen Tunnel. Solche 3D-Flüge lassen sich mit der Zoom-Funktion schnell und einfach anfertigen. Voraussetzung ist allerdings ein geeignetes Quellbild – nicht jedes beliebige Bild eignet sich für dieses Vorhaben. Man kann indessen aus jedem Bild ein geeignetes berechnen – vorausgesetzt, man erkennt den entscheidenden Trick, der hier nicht verraten wird. Nur soviel: Es ist kein Zufall, daß das Gebirgspanorama und der »Tunnel« die gleiche Farbgebung aufweisen. Der Groschen fällt beim Blick auf Bild 15. Im Rahmen der Grafikwerkstatt finden Sie sämtliche Utilities und Routinen, die für solche 3D-Effekte erforderlich sind – viel Vergnügen beim Abflug!

(Michael Nickles/uk)

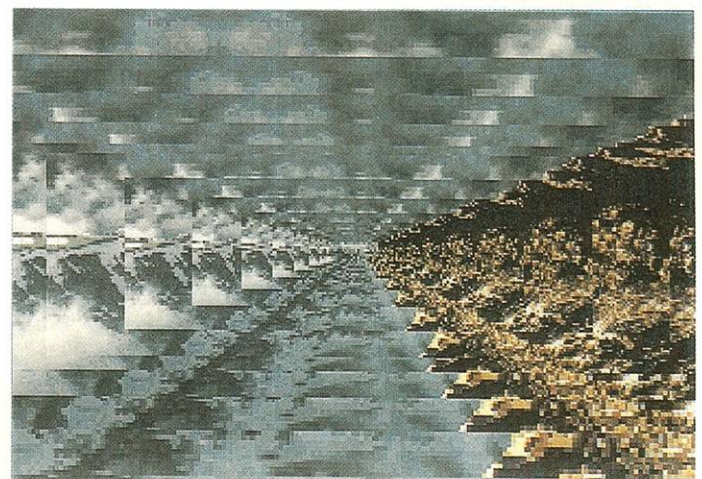


Bild 15. Der »Tunnel« entstand durch Bildmontage des »Gebirgspanoramas« – hier der Zwischenschritt, der vermuten läßt, wie der 3D-Bewegungseffekt zustandekommt.

Tips & Tricks zur Grafik

Was Sie auf etwa zwanzig Seiten an schöner, bunter Welt sehen, läßt sich kaum zusammenfassen: eine Diashow, drehende Spiralen, schnelle Fraktale, Fraktale mit Zoom, Gittermuster aus Linien, Bilder vom Atari im PC, Farbenmischung mit Wertanzeige. Weitere Hilfen wie ein VGA-Emulator, eine Dateizerteilmaschine und ein beachtliches Assemblertuning für die Farbausgabe in Turbo Pascal vervollständigen die Sammlung.

Der VGA-Monochrome-Emulator

»monexp.bas« (Listing 1) entwickelt eine monochrome VGA-Anpassung mit dem Ziel, den monochromen Monitor zu allen Farbprogrammen vollständig kompatibel zu machen. Es gibt nämlich monochrome Monitore, bei denen das Problem auftritt, daß diese den Farben nicht die richtigen Graustufen zuordnen. Daraus folgten viele Probleme. So konnten Sie, wenn Sie im Windows-Setup VGA mit 16 Farboptionen eingestellt hatten, beim Reversi-Spiel die eigenen und die gegnerischen Steine nicht mehr unterscheiden. Sie mußten dann zwangsläufig im Setup die langweilige VGA-Monochrom-Option anwählen. Infolgedessen sah man aber keine dreidimensionalen Schaltknöpfe mehr.

Dem kann abgeholfen werden. Um »monexp.bas« korrekt kompilieren zu können, rufen Sie Quick-Basic mit dem Parameter »/l« auf:

qb /l

Jetzt geben Sie nur noch »monexp on« ein, und schon sind allen Farben die richtigen Grauwerte zugeordnet, so daß Sie sämtliche Graustufen im 16-Farben-Modus ausführen können.

Zum Schluß noch ein kurzer Blick auf die Funktionsweise: das Programm ist sehr einfach aufgebaut, denn es fragt nur die Parameter ab und ruft dann den Interrupt 10hex des VGA-BIOS mit den entsprechenden Registerinhalten auf.

(Karsten Fischer/et)

```
[GXØ] 36: PRINT
[CC1] 37: PRINT v$
[FM5] 38: PRINT "Syntax: MONEXP {on|off}"
[4MØ] 39: END
```

Listing 1. »monexp.bas« stellt Ihnen auch auf monochromen VGA-Bildschirmen Reversi mit dreidimensionalen Schaltknöpfen dar.

Linien zeichnen Muster

In »linien.pas« (Listing 2) zeichnen zwei tanzende Linien eine dreidimensionale bewegte Grafik (Bild 1). Dabei sehen Sie flächige Muster, die verschiedene Schraffierungen aufweisen.

Lassen Sie die Klammerung der Zeilen 33 und 35 aus, erhalten Sie nur die Linien. Bevor Sie das Programm ablaufen lassen, achten Sie darauf, daß Sie auf den passenden Treiber für Ihre Grafikkarte wie »egavga.bgi« zugreifen können. Die Farbzuzuweisung für die Linien ändern Sie in den Zeilen 24 und 25 durch die Zahlenwerte von »SetRGBPalette«.

(Thorsten Claus/et)

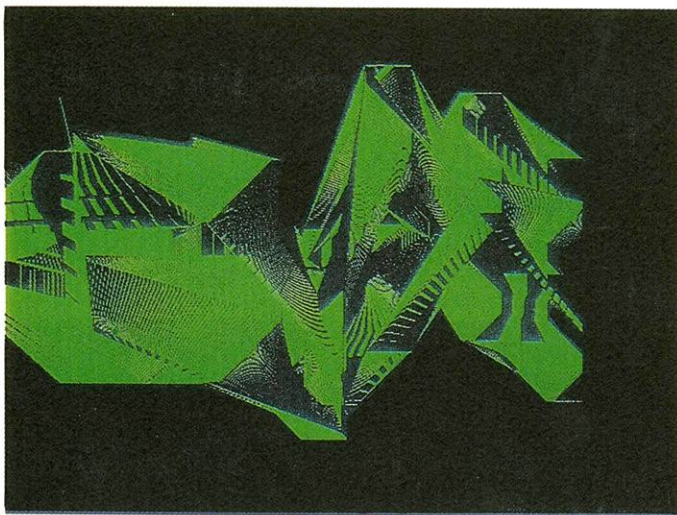


Bild 1. »linien.pas« zeichnet Gebirge aus zwei Linien.

```
[CE3] 1: 'Programm: monexp.bas
[BT5] 2: 'Funktion: VGA-Monochrome-Expander
[N46] 3: 'Sprache : Quick-Basic 4.5 (Parameter /L)
[R24] 4: 'Autor : Karsten Fischer
[EP4] 5: '(c)1991 DMV Widuch GmbH & Co. KG
[ØØØ] 6:
[182] 7: TYPE regtype
[Z12] 8: ax AS INTEGER
[122] 9: bx AS INTEGER
[322] 10: cx AS INTEGER
[522] 11: dx AS INTEGER
[912] 12: bp AS INTEGER
[P12] 13: si AS INTEGER
[SØ2] 14: di AS INTEGER
[PL2] 15: flags AS INTEGER
[WE1] 16: END TYPE
[CW6] 17: DIM inregs AS regtype, outregs AS regtype
[SR4] 18: v$="VGA-MONOCROME-EXPANDER"
[EW2] 19: a$=LCASE$(COMMAND$)
[C65] 20: IF a$<>"on" AND a$<>"off" GOTO help
[5C1] 21: SCREEN 12
[ZS2] 22: IF a$="on" THEN al%=Ø
[823] 23: IF a$="off" THEN al%=1
[WG4] 24: ah%=&H12:bl%=&H33:ax%=ah%*256+al%
[LK4] 25: inregs.ax=ax%:inregs.bx=bl%
[RP6] 26: CALL INTERRUPT(&H1Ø,inregs,outregs):CLS
[CC1] 27: PRINT v$
[4E2] 28: PRINT "STATUS ";
[W78] 29: IF a$="on" THEN PRINT "INSTALLED" ELSE PRINT "REMOVED"
[GXØ] 30: PRINT
[ØB3] 31: PRINT "PRESS ANY KEY"
[KS1] 32: d$=INPUT$(1)
[4MØ] 33: END
[ØØØ] 34:
[LUØ] 35: help:
```

```
[SM3] 1: {Programm: linien.pas
[V96] 2: Funktion: Grafik von Linien/Mustern
[LV5] 3: Sprache: Turbo Pascal ab Version 5.5
[PM3] 4: Autor: Thorsten Claus
[XX4] 5: (c)1992 DMV Widuch GmbH & Co. KG
[9P2] 6: uses Crt,Graph;
[RE2] 7: const p=Pi/18Ø;
[UZ5] 8: var x,y: array[1..1Ø,1..2]of Integer;
[6KØ] 9: a,e,
[A22] 10: m,d: Integer;
[V33] 11: procedure graphic;
[MUØ] 12: begin
```



```
[HF1] 13: m:=9;d:=2;
[6S2] 14: InitGraph(m,d,'');
[ZN0] 15: end;
[3Y2] 16: procedure linien;
[MU0] 17: begin
[YF2] 18: for a:=0 to 9 do
[MU0] 19: begin
[4A2] 20: x[a,1]:=a*64;
[HU3] 21: y[a+1,1]:=Random(200)+150;
[NU3] 22: y[a+1,2]:=Random(200)+150;
[ZN0] 23: end;
[3N3] 24: SetRGBPalette(2,0,63,0);
[RM3] 25: SetRGBPalette(1,0,43,0);
[P31] 26: repeat
[151] 27: Inc(e);
[ZF2] 28: for a:=1 to 9 do
[MU0] 29: begin
[P75] 30: y[a,1]:=Round(y[a,1]+Sin(a*e*p));
[NP5] 31: y[a,2]:=Round(y[a,2]+Sin(a*0.7*e*p));
[6N5] 32: x[a,1]:=Round(x[a,1]+Sin(a*0.4*p*e));
[RX0] 33: end;
[WE3] 34: for a:=1 to 9 do begin
[M62] 35: SetColor(1);
[GR0] 36: Line
[1F5] 37: (x[a,1],y[a,1],x[a+1,1],y[a+1,1]);
[XY1] 38: SetColor(2);
[GR0] 39: Line
[7F5] 40: (x[a,1],y[a,2],x[a+1,1],y[a+1,2]);
[ZN0] 41: end;
[XA1] 42: Delay(3);
[RY1] 43: SetColor(0);
[ZF2] 44: for a:=1 to 9 do
[MU0] 45: begin
[GR0] 46: Line
[1F5] 47: (x[a,1],y[a,1],x[a+1,1],y[a+1,1]);
[GR0] 48: Line
[7F5] 49: (x[a,1],y[a,2],x[a+1,1],y[a+1,2]);
[ZN0] 50: end;
[K13] 51: until KeyPressed;
[ZN0] 52: end;
[MU0] 53: begin
[AC1] 54: graphic;
[MA1] 55: linien;
[522] 56: CloseGraph;
[KN0] 57: end.
```

Listing 2. »linien.pas« läßt durch zwei Linien merkwürdige Muster aufbauen.

Betrachten von PCX-Dateien

Das PCX-Format von PC-Paintbrush gehört zu den weit verbreiteten Grafikformaten. Mit »pcx_view.c« (Listing 3) betrachten Sie solcherart kodierte Bilder beliebiger Auflösung und Farbanzahl, ohne Paintbrush bemühen zu müssen. Die einfache Komprimierungsmethode des PCX-Formats gestattet dabei eine hohe Anzeigegeschwindigkeit. Aber auch unkomprimierte PCX-Bilder, die selten vorkommen, werden korrekt angezeigt.

Zwar gibt es bei manchen Benutzeroberflächen wie PC-Shell oder dem Norton Commander eine Anzeige, die vorgibt, dasselbe zu leisten, doch sind die Fehlermeldungen, die Sie oftmals erhalten, und das, was Sie zu sehen bekommen – falls die Anzeige sich doch einmal herablassen sollte, in den Grafikmodus zu schalten – dermaßen kläglich, daß sich beinahe die Frage stellt, wer für den Quellcode verantwortlich zeichnet!

»pcx_view.c« unterstützt uneingeschränkt sämtliche Auflösungen und Farben der Standard-VGA-Karte. Spezielle Videomodi, wie sie nur Super-VGA-Karten zur Verfügung stellen (640 × 480, 800 × 600, 1024 × 768, bei 16 oder 256 Farben), kann »pcx_view.c« wegen seiner Beschränkung auf 1024 Byte jedoch nicht unterstützen. Da jeder Hersteller (Tseng, Trident, Paradise und so weiter) sein eigenes Süppchen bei der Ansteuerung seiner Karte in »extended modes« kocht, würde ein Programm, das diese Modi nutzen wollte, sehr aufwendig (und lang) werden.

Die Anwendung von »pcx_view.c« ist denkbar einfach: Sie geben den Dateinamen der Grafik, eventuell mit Laufwerk- und Pfadangabe, als Kommandozeilenparameter an (die Erweiterung »pcx« darf fehlen). Das Programm schaltet hierauf automatisch in den optimalen Videomodus

und zeigt das Bild an. Sollte die nötige Auflösung die maximal verfügbare übersteigen, sehen Sie entsprechend nur den linken oberen Teil des Bildes, das dann auch nur bis zum unteren Bildschirmrand geladen wird. Geben Sie mehrere Dateinamen an, erhalten Sie ein Bild nach dem anderen am Bildschirm. Nach jedem Bild wartet das Programm auf einen Tastendruck; mit [Esc] kommen Sie sofort in den Textmodus zurück, ansonsten nach dem letzten Bild. Falls eine Datei nicht existiert oder nicht die PCX-Kennung hat, kehrt »pcx_view.c« mit dem Wert 1 im Rückkehrcode zur MS-DOS-Ebene zurück. Bei fehlerfreier Ausführung übergibt »pcx_view.c« den Wert 0 an den Rückkehrcode. Sie kompilieren den Quellcode mit Turbo C 2.0 im Speichermodell »small«, das die Standardeinstellung darstellt. Besondere Schalter brauchen Sie nicht zu berücksichtigen. Die schnellste Version erhalten Sie allerdings, wenn Sie im Menü »[O]ptions [C]ompiler [C]ode Generation« den Schalter »Alignment« auf »Word« setzen und in »[O] [C] [O]ptimization« alle Optimierungen aktivieren.

Der Compiler wird, sofern alle »warnings« eingeschaltet sind, zwar mit zehn Warnungen aufwarten, diese können Sie jedoch getrost ignorieren: es wird absolut korrekter Code erzeugt.

Der Quelltext, den Sie noch um einige Bytes kürzen könnten, wird unformatiert in seiner knappen Länge von 1023 Byte abgedruckt, um zu zeigen, was Sie bei wirklich extremer Kodierung mit der Programmiersprache C vollbringen können!

(Philipp Amtmann/et)

```
[444] 1: /* Programm: pcx view.c */
[043] 2: #include<stdlib.h>
[8W2] 3: #include<stdio.h>
[7M2] 4: #include<dos.h>
[KC3] 5: #define g_int_(16)
[L02] 6: #define f for
[XD6] 7: struct{char h,v,c,n;int l,t,r,b,x,y;char p[
48];d,a;int e;h;FILE*q;int e,i,j,k,t,w;cha
r*m,far*u,far*v,c,p[768];l() {f(m=p,j=0;j<h.
e);} {c=fgetc(q);if(h.c&&(c&192)>128)f(t=c&63
,c=fgetc(q);t;--)(s:=m=c;m+=j++<320;)}else{
t=1;goto s;}}main(){
[R43] 8: while(*(char*)strcpy(p,++ argv)){if(!strchr
r(p,'.')strcat(p,".PCX");i=1;if(!fopen(
p,"rb"))goto z;fread(&h,68,1,q);if(h.h!=10)
goto z;t=h.b-h.t+1;i=h.r-h.l+1;j=w=0;v=10L
<<28;_AX=c-h.n&8?19:t>350?18:t>200?16:13+(i
>320);g;k=1<h.n*h.a;
[KE3] 9: if(h.v!=3&&k>2){if(c>18){j=768;fseek(q,-j,2
);fread(p,1,j,q);e=m=p;}else{f(j<17;j++)p[
j]=j%16;DX=p;AX=4098;g;j=48;e=m-h.p;}f(j;
j--)*m++>=2;DX=e;CX=k;BX=0;_AX=4114;g;
)fseek(q,128,0);j=320;k=200;if(c<19){i/8;j
/c>13?4:8;k=480;}}
[KG2] 10: if((e=min(i,j))<j)w=j-i;f(i=min(t,k);i;--
,v=w)f(u=v,k=0;k<h.a;k++){l();if(h.a>1){out
portb(964,2);outportb(965,1<<k);}f(v=u,m=p,
j=0;j<e;j++)*v++=*m++;}fclose(q);_AX=0;_in
t_(22);if(_AH==1)break;}z:_AX=3;g;return i
;}
```

Listing 3. Mit »pcx_view.c« betrachten Sie PCX-Bilder in fast beliebiger Auflösung und Farbanzahl.

Diashow im Speicher

Mit dem speicherresidenten »vision.pas« (Listing 4) speichern Sie Schnappschüsse von VGA-Bildern auf Ihrem Datenträger, die Sie später wie in einer Diashow betrachten können. Der Benutzer benötigt hierbei keine Vorkenntnisse, um speziell kodierte Bilder wie GIF-Bilder in seine eigenen Programme einzubinden. Dank des residenten Teils von Vision muß der Benutzer nur noch die F10-Taste drücken, und »vision.exe« legt eine Datei namens »vision« auf Ihrem Datenträger an, die die Bildinformationen enthält. Besteht diese Datei bereits, werden weitere Bilder an diese angehängt. Jedes gespeicherte Bild verbraucht 64 768 Byte auf Diskette oder Festplatte. Die ersten 768 Byte sind die Paletteninformationen (256 Palettenregister mit je einer

Rot-, Grün- und Blau-Intensität), die restlichen 64 000 Bytes enthalten ein identisches Duplikat des VGA-Bildspeichers, das an der Adresse A000hex beginnt.

Der andere Teil von Vision realisiert die Diashow, das heißt, er liest die vorher generierte Datei »vision« und gibt die gespeicherten Bilder (Bild 2) aus. Nach jedem Bild wartet das Programm auf einen Tastendruck des Benutzers, um dann fortzufahren.

Die Anzahl der speicherbaren Bilder ist nur durch die Kapazität der Diskette oder der Festplatte beschränkt. Auf einer 720-KByte-Diskette lassen sich maximal elf Bilder, auf einer 1,2-MByte-Diskette 18 Bilder speichern. Teilen Sie die Kapazität des Speichermediums in der Maßeinheit Byte durch 64 768 Byte für jedes einzelne Bild, so erhalten Sie die Anzahl der speicherbaren Bilder.

Wegen der begrenzten Länge des Quellcodes erkennt das Programm nicht, ob es sich im VGA-13hex-Grafikmodus befindet. Es kopiert nur den Bildspeicherbereich ab Segment A000hex. Vision erkennt auch nicht, ob ein Datenträger in seiner Kapazität erschöpft ist und erzeugt daher keine Fehlermeldung und keinen Systemabsturz. Daher sollte sich der Benutzer zuvor vergewissern, wieviele Schnappschüsse das Speichermedium aufnehmen kann. Bei der Installation des residenten Teils von Vision sollte der Benutzer darauf achten, daß er es nicht mehrmals installiert, da das Programm nicht erkennen kann, ob es sich schon resident im Speicher befindet.

Den residenten Teil von Vision aktivieren Sie, indem Sie »vision« und einen beliebigen Parameter eingeben, zum Beispiel:

vision on

Die Diashow aktivieren Sie durch einfache Angabe von »vision« ohne Parameter. Die Bilderdatei »vision« muß sich dabei im aktuellen Verzeichnis befinden, da Sie sonst eine Fehlermeldung erhalten, egal, wo sich das Programm »vision.exe« momentan befindet.

Sie kompilieren »vision.pas« unter der Entwicklungsumgebung von Turbo Pascal Version 6.0, wobei Sie den integrierten Assembler nutzen (Zeile 21 bis 34). Der Quellcode umfaßt – als Einsendung zum Kilomark-Wettbewerb – nur 1022 Byte, das ablauffähige Kompilat umfaßt 4608 Byte. Der residente Teil reserviert einen 2 048 Byte großen Stack-Speicherbereich, wovon Sie nur 1 608 Byte benötigen. Sie aktivieren ihn durch Drücken der F10-Taste. Nach der Aktivierung erhält die Tastaturinterrupt-Routine die Tastaturinformation. »vision.exe« beinhaltet zwar keine Prozedur zur Fehlerbehandlung, stellt aber bei falscher Anwendung durch den Benutzer auch keinen Schaden an.

(Patrick Bennour/et)

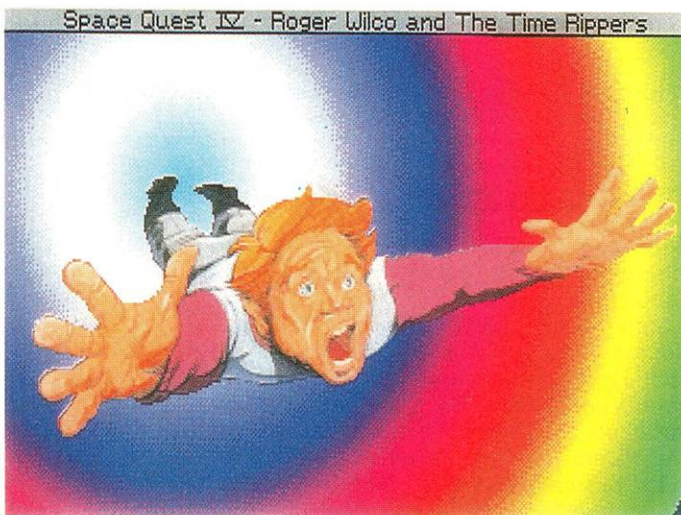


Bild 2. Mit »vision.pas« erscheint ein Bild aus der Datei »vision«.

```
[VN3] 1: {Programm: vision.pas
[QZ2] 2: Funktion: Diashow
[HV5] 3: Sprache: Turbo Pascal ab Version 6.0
[NS3] 4: Autor: Patrick Bennour
[TS4] 5: (c)1991 DMV Widuch GmbH & Co. KG)
[SU1] 6: {$M $8000,0,0}
[S62] 7: uses Crt,Dos;
[TN4] 8: var VX: Word absolute $A000;
[VA4] 9: 0:PX: array[1..768]of Byte;
[V13] 10: D: file;R: Integer;
[S04] 11: Ch: Char;Old: procedure;
[M62] 12: procedure Ds;
[MU0] 13: begin
[U43] 14: Assign(D,'Vision');
[DS3] 15: ($I-) Reset(D,1);($I+)
[L95] 16: if IOResult<>0 then ReWrite(D,1);
[ZN0] 17: end;
[HV3] 18: procedure Gs(Chc: Byte);
[VU1] 19: label CC,CD;
[MU0] 20: begin
[FN0] 21: Asm
[4R1] 22: XOR bx,bx;
[612] 23: mov cx,0FFh;
[6Y2] 24: mov dx,Seg(PX[1]);
[6H1] 25: mov es,dx
[HM3] 26: mov dx,Offset(PX[1]);
[PL1] 27: cmp chc,1;
[XU0] 28: je CC;
[RZ1] 29: mov ax,1017h;
[D11] 30: jmp CD
[RH0] 31: CC:
[3Z1] 32: mov ax,1012h;
[SU1] 33: CD: Int 10h
[ZN0] 34: end;
[ZN0] 35: end;
[ZW0] 36: ($F+)
[894] 37: procedure Vs;interrupt;
[MU0] 38: begin
[153] 39: if Port[$60]=$44 then
[MU0] 40: begin
[UD4] 41: Ds;Seek(D,FileSize(D));Gs(0);
[V93] 42: BlockWrite(D,PX,768,R);
[VD5] 43: BlockWrite(D,VX,64000,R);Close(D);
[ZN0] 44: end;
[EM1] 45: Asm pushf
[CC1] 46: end;Old;
[ZN0] 47: end;
[1X0] 48: ($F-)
[MU0] 49: begin
[G84] 50: if ParamCount=0 then begin
[DL0] 51: Ds;
[3L2] 52: Asm mov ax,13h;
[Y71] 53: Int 10h
[ZN0] 54: end;
[4W2] 55: while NOT Eof(D)do
[MU0] 56: begin
[X64] 57: BlockRead(D,PX,768,R);Gs(1);
[LN3] 58: BlockRead(D,VX,64000,R);
[Z22] 59: Ch:=ReadKey;
[ZN0] 60: end;
[XU3] 61: Close(D);TextMode(Co80);
[3D1] 62: end else
[MU0] 63: begin
[143] 64: GetIntVec($9,@Old);
[CY4] 65: SetIntVec($9,Addr(Vs));Keep(0);
[ZN0] 66: end;
[KN0] 67: end.
```

Listing 4. »vision.pas« lädt die Bilder aus der Datei »vision«.

Drehende Spirale

»spirale.c« (Listing 5) berechnet 15 Spiralen in verschiedenen Farben. Die anschließende Animation erreichen Sie durch Änderung der Farbpalette. Für die Spirale brauchen Sie eine EGA- oder VGA-Grafikkarte.

Sie kompilieren »spirale.c« in Turbo C++ oder in Turbo C 2.0 im Speichermodell »small«. Nach Programmaufruf lesen Sie die Meldung:

Defaultwerte? (j/n)

Mit »j« oder allein mit der Eingabe-Taste bestätigen Sie die voreingestellten Parameter (Bild 3). Der Programmablauf sieht dann so aus:

1. Die Variablen für die graphische Darstellung werden initialisiert; bei einem eventuell auftretenden Fehler bricht Spirale mit der entsprechenden Fehlermeldung ab.
2. Die Palette erhält eine Vorbelegung, so daß fließendere Farb-Übergänge entstehen.
3. Das Zeichnen der Spirale: Die Mathematik beschreibt

DOS-Software gesucht

Wir suchen für das Betriebssystem MS-DOS ab Version 3.0

- vermarktungsfähige Programme aller Kategorien (von Anwendungs- und Hilfsprogrammen bis hin zu Spielen oder Programmbibliotheken)
- Programme für den geschäftlichen Alltag
- Software für spezielle Anwendungsgebiete
- Branchenlösungen

Ob noch in der Entwicklung, gerade fertiggestellt oder bereits in Kleinserien vertrieben: Wir sind der richtige Vertriebspartner für Ihre Software. Möglicherweise wird gerade Ihr Programm zum "Bestseller". Scheuen Sie sich also nicht, den ersten Schritt zu unternehmen. Senden Sie Ihr Programm völlig unverbindlich mit Anleitung beziehungsweise Handbuch auf Diskette an

Redaktion DOS International
»DOS-Software«
Gruberstraße 46c
8011 Poing bei München
Tel.: (0 81 21) 7 69-0

Wir behandeln Ihre Einsendung diskret und freuen uns auf Ihre Reaktion.

DMV-PROFITOOLS FÜR CLIPPER

Statt 100 simpler Funktionen ein professionelles Entwicklungssystem

Die vorliegenden Clipper-Tools entstanden aus der Erfahrung und der täglichen Arbeit mit professionellen Projekten. Sie stellen nun eine standardisierte Lösung der immer wiederkehrenden Aufgaben dar. Im Laufe der Zeit konnten die Routinen in vielen Bereichen optimiert werden, so daß mit den DMV-ProfiTools für Clipper eine ausgereifte Programmsammlung für "Clipper Sommer '87" und "Clipper 5" zur Verfügung steht. Wie die grafische Darstellung zeigt, decken die ProfiTools alle Problembereiche der Clipper-Programmentwicklung ab. Sie enthalten sämtliche Mechanismen komplexer Applikationen und bilden als Gesamtes ein umfassendes Entwicklungssystem.

Um alle Funktionen anzusprechen, reicht der Platz hier bei weitem nicht. Um Sie aber neugierig zu machen, seien einige der Leistungsmerkmale herausgestellt. Neben einem **Window-System**, integrierter **Netzwerkfähigkeit**, einem **DOS-Manager**, automatischer **Farbanpassung** und vielem, vielem anderen bieten die ProfiTools:

Toggles: Sie markieren und verwalten komfortabel Anfang und Ende von Datenbankausschnitten und ersetzen die umständlicheren, zeitraubenden Filter.

Standard-Verwaltung: Ein zentrales Modul, dem alle notwendigen Funktionen angegliedert sind, verkürzt den Entwicklungsaufwand der

Datenbankentwicklung auf ein Minimum.

Übersicht: Toggling, Marking, Seitwärts-Scrolling, freie Programmierung und viele weitere Details gewähren präzise und komplette Überschaubarkeit der Datenbank mit diesen Tools.

Hilfesystem: Der intelligente Hilfgenerator erzeugt eine hierarchische Textgliederung, die interaktiv und in verschiedenen Applikationen aufgerufen werden kann.

Marking: Ein neues Verfahren der Datensatzmarkierung erhöht bei komplexen Selektionen enorm die Arbeitsgeschwindigkeit. Bis zu 500.000 Datensätze können pro Datei im RAM gehalten werden.

Druckmanager: Auch mehrfach geschachtelte Auswertungen kann der Drucklisten-Manager einfach und schnell bearbeiten.

Maskengenerator: derzeit das Optimum an Arbeitserleichterung.



DMV-ProfiTools für Clipper

199,- DM*

außerdem die Entwicklungsumgebung mit Editor:

DMV-Integrator für Clipper

199,- DM*

* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,- bzw. für das Ausland DM 6,- Porto und Verpackung.

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV
Daten- und
Medienverlag

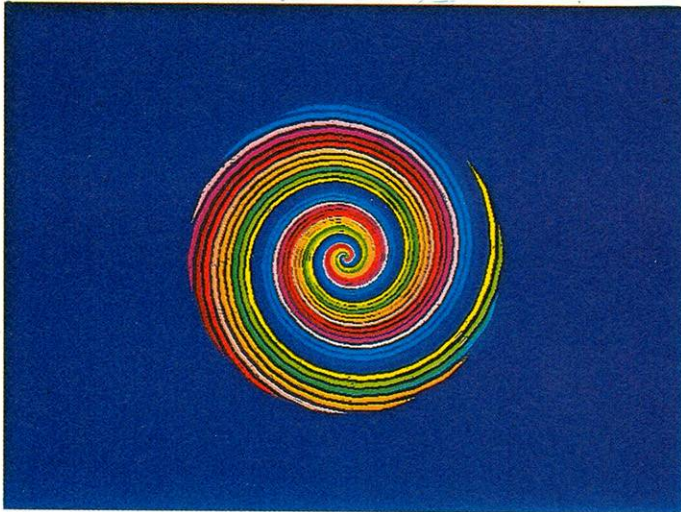


Bild 3. »spirale.c« zeichnet drehende Spiralen.

eine Spirale durch die Formel $r = \phi$, wobei Radius »r« und Winkel »phi« in Polarkoordinaten eingegeben werden. Um die einzelnen Punkte der Spirale auf dem Bildschirm darzustellen, rechnen Sie diese zuerst in die gewohnten kartesischen Koordinaten mit folgenden Formeln um:

```
x=r*sin(phi)*c
y=r*cos(phi)
```

Dabei bestimmen Sie noch den Korrekturfaktor »c«, der sich aus dem Seiten-Längen-Verhältnis (»aspectratio«) ableitet, das Sie mit der Funktion

```
getaspectratio(&x,&y)
```

für den jeweiligen Grafikadapter bestimmen. Der Korrekturfaktor c berechnet sich dann als:

```
c=y/x
```

Durch diese Korrektur erreichen Sie, daß die Spirale eine runde statt einer ovalen Form bekommt. Hier bleibt noch anzumerken, daß statt der Formel $r = \phi$, die Formel $r = \phi^3$ benutzt wurde, damit die Spirale schneller wächst.

4. Das scheinbare Drehen der Spirale erreichen Sie durch Verändern der Palette. Die Figur bleibt zwar unverändert stehen, aber die Farben ändern sich zyklisch. Die einzelnen Farbeinträge der Palette werden jeweils durch ihre Nachfolger ersetzt, also die Farbe 1 durch die Farbe 2, diese durch die Farbe 3 und so weiter. Zum Schluß ersetzen Sie im Programm Farbe 15 durch Farbe 1, und der Zyklus beginnt von vorn. Die Farbe 0 als Hintergrundfarbe setzen Sie auf Dunkelblau und lassen sie unverändert, weil sonst der ganze Bildschirm flackert. Mit einem Tastendruck beenden Sie Spirale. Wenn Sie nach Programmaufruf die Abfrage

```
»Defaultwerte? (j/n)«
```

mit »n« beantworten, liest Spirale vier Variablen ein; geben Sie die entsprechenden Parameter, durch Komma getrennt, an. Mit den Werten beeinflussen Sie die Form der Spirale. Beim Zeichnen der Spirale wird der Winkel phi stufenweise erhöht, und der zugehörige Radius »r« errechnet. Danach werden die Punkte »P(r,phi)« in »P(x,y)« umgerechnet und durch Geraden miteinander verbunden. Die Form der Spirale verändern also die vier Parameter:

1. »n« gibt an, wieviele ganze Umdrehungen Sie berechnen wollen. Sinnvolle Werte liegen zwischen 1 und 10. Größere Werte von »n« haben den Effekt, daß die Spirale weit über die Bildschirmgrenze hinauswächst.

2. »s« gibt die Schrittweite an, um die Sie den Winkel »phi« jeweils vergrößern wollen. Dieser Parameter geht direkt in die Rechenzeit ein. Sinnvolle Werte liegen zwischen 1 und 30. Je größer Sie »s« wählen, desto exakter fällt die Zeichnung aus.

3. »f« staucht die Spirale. Der Parameter geht als Faktor $1/f$ in die Berechnung des Radius mit ein:

```
x=1/f*r*sin(phi)
y=1/f*cos(phi)
```

Für große Werte des Parameters »n« sollten Sie auch »f« größer wählen, da sonst die Spirale über die Bildschirmgrenzen hinauswächst. Sinnvolle Werte für »f« liegen zwischen 1 und 500.

4. »d« enthält die Zeitspanne in Millisekunden, um die die Farbänderung jeweils verzögert wird. Mit »d« beeinflussen Sie also das »Drehen« der Spirale. Kleine Werte von »d« lassen die Spirale schneller, große Werte langsamer drehen. Sinnvolle Werte liegen zwischen 1 und 400.

Falls ein 80x87-Co-Prozessor vorhanden ist, lohnt es sich, den entsprechenden Compiler-Schalter zu setzen, da relativ viele Fließkomma-Rechnungen (»floating-point-Operationen«) durchgeführt werden. Die Konstanten, die im Programm auftauchen, haben folgende Bedeutung:

```
0.017453292 = Pi/180
```

Die Punkte der Spirale berechnen Sie also in Schritten nach der Formel $s * \pi / 180$. Die Konstanten »0,45« und »0.0195« (Zeile 38) wurden empirisch gefunden. Sie legen den Abstand fest, den zwei benachbarte Spiralen verschiedener Farben einnehmen. Jede Spirale besteht aus 12 Teilschrauben derselben Farbe. Die Übersichtlichkeit des Quelltextes ist zwar bescheiden, aber anders wäre es nicht machbar, das Programm auf 1024 Byte zu begrenzen. Daher benutzen Sie verschiedene Define-Anweisungen für Funktionen, die Sie öfter aufrufen. Das ist auch der Grund dafür, daß Sie noch ein »häßliches« »goto« einbauen, das als Ausstieg dient, falls der Benutzer eine Taste drückt.

Als Erweiterung von Spirale könnten Sie die Verzögerungszeit »d« variieren, so daß der Eindruck entsteht, die Spirale drehe sich mal schnell, dann wieder langsam. Auch könnten Sie die Abfrage der Parameter komfortabler gestalten. Weiterhin könnten Sie das ganze Programm als Funktion zusammenfassen, um es auch in anderen Programmen weiter zu verwenden. In diesem Fall sollten Sie die notwendigen Parameter als Funktionsargumente übergeben.

(Andreas Banholzer/et)

```
[ØE3] 1: #include <graphics.h>
[DR2] 2: #include <math.h>
[L13] 3: #include <stdio.h>
[2Ø3] 4: #include <conio.h>
[U83] 5: #include <stdlib.h>
[9L2] 6: #include <dos.h>
[X64] 7: #define T if(kbhit()) goto E
[7Ø4] 8: #define M clrscr();printf
[513] 9: #define P s.colors
[Al4] 10: #define S setallpalette
[E73] 11: struct palettetype
[MF4] 12: {
[W32] 13:     s={16,8,1,9,11,27,47,5,4,36,39,38,
[KH2] 14:     2,19,22,54,63};
[GP2] 15: void main(void)
[PT3] 16: {
[MD3] 17:     double r,p,a,c;
[8S3] 18:     int x,y,xa=4,ya=12,f=11Ø,
[8X2] 19:     d=99,i,k,l,X,Y,gd=Ø,gm;
[D45] 20:     M("Defaultwerte? (j/n)");
[NN1] 21:     if((x=getch())=='n')
[3H2] 22:     {
[BB2] 23:         M("n[%d],s[%d],r[%d],d[%d]\n",
[933] 24:         xa,ya,f,d);
[TN3] 25:         scanf("%d,%d,%d,%d",
[5K2] 26:         &xa,&ya,&f,&d);
[223] 27:         initgraph(&gd,&gm,"");
[6E1] 28:         if((i=graphresult())!=Ø)
[362] 29:         {
[762] 30:             M("Error:%s\n",
[YH3] 31:             grapherrormsg(i));
[NF4] 32:             exit(1);
[4X4] 33:             X=getmaxx()/2;
[W75] 34:             Y=getmaxy()/2;
[PM3] 35:             getaspectratio(&x,&y);
[E22] 36:             c=(double)y/(double)x;S(&s);
[812] 37:             for(i=15;i>1;i--)(setcolor(i);
[BP2] 38:             for(l=12;l>Ø;l--)(moveto(X,Y);
[C63] 39:             for(k=Ø,p=Ø;k<36Ø*xa;
[122] 40:             k+=ya,p=Ø.Ø17453292*k,
[DX1] 41:             r=1.Ø/f*p*p*p,
a=p+Ø.45*i+Ø.Ø195*1,
x=(int)(r*sin(a)*c)+X,
y=(int)(r*cos(a))+Y,
lineto(x,y))
```



```
[AY0] 42: T;}}
[SH3] 43: while(1){P[15]=P[1];
[SP4] 44: for(i=1;i<15;P[i]=P[i+1],i++);
[EX2] 45: delay(d);S(&s);T;}}
[SH2] 46: E:closegraph();
[U70] 47: }
```

Listing 5. »spirale.c« wirbelt eine Spirale über den Bildschirm.

Extrem schnelle Fraktaldarstellung

»speedfra.c« (Listing 6) schreiben Sie in Turbo-C Version 2.0. Speedfra baut Ihnen das Fraktal (Bild 4) an einem mit 25 Mhz getakteten 386-PC in weniger als zehn Sekunden auf. Der schnelle Bildaufbau erstaunt um so mehr, als Sie Speedfra in einer Hochsprache schreiben. Das verwendete Fraktal ist dem Apfelmännchen sehr ähnlich, jedoch mit niedrigerem numerischen Aufwand berechenbar. Zudem erfolgte hier eine Realisierung unter Nutzung der Integer-Arithmetik. Die eigentliche Darstellung führt die C-Funktion »fraktal« (Zeile 11) aus. Die übergebenen Koordinaten (»xu«, »xo«, »yu«, »yo«) und der Wert für den Abbruch (»rm«) sind als ganze Zahlen und als das etwa Tausendfache des jeweiligen Wertes der komplexen Zahlenebene (siehe Zeilen 11 bis 14) einzugeben. Die maximale Anzahl von Iterationen legen Sie durch »nm« fest, beispielsweise:

fraktal (-3000,3000,-100,1300,10 000,20);

Die Bedeutung dieser Parameter entnehmen Sie Tabelle 1. Vor dem Aufruf der Funktion »fraktal« schalten Sie die Grafik ein. Die Funktion ermittelt selbständig das aktuelle Format und paßt sich daran an.

(Klaus Röbenack/et)

Bedeutung	Bereich
Realteil	-3 bis +3
Imaginärteil	-0.1 bis +1.3
Abbruchschranke	bei Wert 10
Maximale Anzahl der Iterationen	20

Tabelle 1. Diese Parameter nutzt »speedfra.c« in Zeile 20.

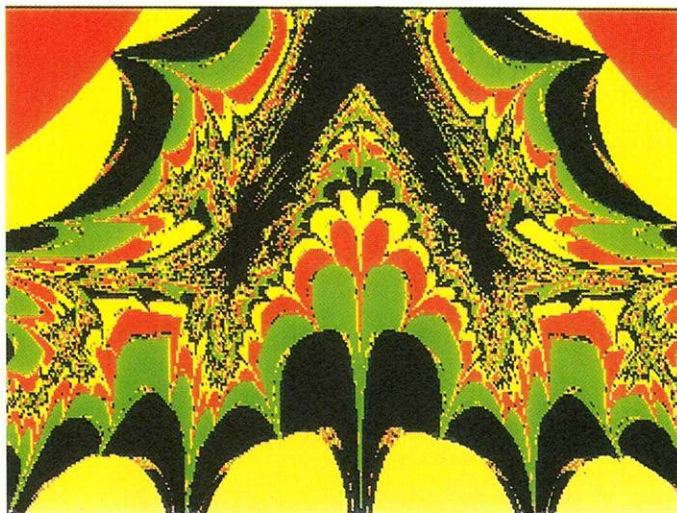


Bild 4. »speedfra.c« baut das Fraktal erstaunlich schnell auf.

```
[SS3] 1: /* Programm: speedfra.c
[RW4] 2: Funktion: schnelles Fraktal
[IR2] 3: Sprache: Turbo C
[AE4] 4: Autor: von Klaus Röbenack
[EZ4] 5: (c)1992 DMV Widuch GmbH & Co.KG */
```

```
[000] 6:
[DR2] 7: #include <math.h>
[L13] 8: #include <stdio.h>
[0E3] 9: #include <graphics.h>
[000] 10:
[FC4] 11: fraktal(xu,xo,yu,yo,rm,nm)
[6Z3] 12: int xu,xo; /* Real-Teil */
[YX3] 13: int yu,yo; /* Imag-Teil */
[P25] 14: int rm,nm; /* max. Radius/Iter. */
[P70] 15: {
[2U6] 16: int xd,yd,x,y,xz,yz,xq,yq,xm,ym,cm,i,j,n;
[RV2] 17: cm=getmaxcolor();
[462] 18: xm=getmaxx();
[962] 19: ym=getmaxy();
[562] 20: xd=(xo-xu)/xm;
[B62] 21: yd=(yo-yu)/ym;
[7K4] 22: for (y=yu,j=0;j<ym;j++,y+=yd)
[592] 23: for (x=xu,i=0;
[4C2] 24: i<xm;i++,x+=xd)
[P70] 25: {
[7D3] 26: for (n=0,xz=x,yz=y;
[YA1] 27: n<nm;n++)
[P70] 28: {
[8V1] 29: xq=abs(xz);
[BV1] 30: yq=abs(yz);
[533] 31: if (xq+yq>rm) break;
[GU2] 32: xz=(int)(long)xz
[132] 33: *(long)yz>>10;
[ZE1] 34: xz+=xz*x;
[R12] 35: yz+=yz*y;
[U70] 36: }
[U32] 37: if (n==nm) n=0;
[CK3] 38: putpixel(i,ym-j,n,cm);
[U70] 39: }
[U70] 40: }
[000] 41:
[UW0] 42: main()
[P70] 43: {
[8Y1] 44: int gd,gm;
[4Z3] 45: gd=CGA; /* z.B. VGA */
[P74] 46: gm=CGAC0; /* z.B. VGALO */
[AU4] 47: initgraph(&gd,&gm,"\\tc\\bgi");
[CB1] 48: fraktal
[AP3] 49: (-3000,3000,-1300,1000,20);
[R61] 50: getch();
[S42] 51: closegraph();
[U70] 52: }
```

Listing 6. »speedfra.c« baut Ihnen ein Fraktal in Bestzeit auf.

Fliegender Teppich

»teppich.c« (Listing 7) zeigt einen »fliegenden Teppich«, der immer wieder von den Bildschirmbegrenzungen »abprallt«. Sie kompilieren den Quellcode mit Borland C++ oder Turbo C. Vor der Zeile 28

graphresult

müssen Sie Ihren Pfad zu den BGI-Dateien eingeben, wobei Sie den Rückstrich in doppelter Ausführung eingeben, wie Zeile 27 beispielhaft demonstriert. Teppich arbeitet in allen von Borland C unterstützten Grafikmodi. Die Bedeutung der Variablen entnehmen Sie bitte Tabelle 2. Die Verwaltung des Zwischenspeichers ähnelt der eines Spoolers, in dem das zuerst Eingegangene zuletzt rausgeht. Experimente lohnen sich mit den Parametern »nl« und »sx0« bis »sy1«.

(Torsten Bolten/et)

Variable	Bedeutung
nl	Anzahl der Linien
sx0,sx1,sy0,sy1	Schrittweite der Eckpunkte
xx, yx	x-, y-Position der unteren rechten Bildschirm-ecke
xn, yn	x-, y-Position der oberen linken Bildschirm-ecke
gd, gm	Parameter für Grafikmodus
x0, x1, y0, y1	Position der aktuellen Linie
i	Laufvariable
p[i].x0 bis p[i].y1	Zwischenspeicher für gezeichnete Linien
h	Kopf (»Head«) des Zwischenspeichers.
t	Ende (»Tail«) des Zwischenspeichers.
obs()	Funktion zur Überwachung der Bildschirm-grenzen

Tabelle 2. In »teppich.c« arbeiten Sie mit diesen Variablen.


```
[PG3] 1: /* Programm: teppich.c
[1U5] 2:   Funktion: testet Grafikfähigkeit
[1R2] 3:   Sprache: Turbo C
[3L3] 4:   Autor: Torsten Bolten
[E24] 5:   (c)1992 DMV Widuch GmbH & Co.KG */
[BE3] 6: #include<graphics.h>
[043] 7: #include<stdlib.h>
[8W2] 8: #include<stdio.h>
[EU2] 9: #include<conio.h>
[000] 10:
[T63] 11: typedef struct p_a
[F93] 12: {int x0,y0,x1,y1;};
[000] 13:
[9W2] 14: struct p_a p[200];
[7L4] 15: int xx,yy,xn,yn,x0,y0,x1,y1,
[5M2] 16:   sx0,sy0,sx1,sy1;
[000] 17:
[3B2] 18: void obs(void)
[FV4] 19: {if(x0>xx;||x0<xn) sx0=-sx0;
[GM4] 20: if(x1>xx;||x1<xn) sx1=-sx1;
[VM4] 21: if(y0>yy;||y0<yn) sy0=-sy0;
[N45] 22: if(y1>yy;||y1<yn) sy1=-sy1;};
[000] 23:
[KH2] 24: void main(void)
[4Y4] 25: {int gd=DETECT,gm,i,t,h,nl=50;
[PS3] 26: sx0=5;sx1=3;sy0=2;sy1=4;
[F55] 27: initgraph(&gd,&gm,"C:\\tc\\BGI");
[NT3] 28: if(graphresult()!=grOk)
[V26] 29: {puts("Grafikfehler (Pfad falsch ?)");
[6E1] 30: exit(1);};
[Z24] 31: setcolor(getmaxcolor());
[5P1] 32: xn=0;yn=0;
[0N4] 33: xx=getmaxx();yy=getmaxy();
[B24] 34: setwritemode(XOR_PUT);
[E64] 35: x0=xn;y0=yn;x1=xn;y1=yn;
[LS2] 36: for(i=0; i<nl; i++)
[Z23] 37: {line(x0,y0,x1,y1);
[2R3] 38: p[i].x0=x0;p[i].y0=y0;
[ER3] 39: p[i].x1=x1;p[i].y1=y1;
[BD2] 40: x0+=sx0;x1+=sx1;
[ND2] 41: y0+=sy0;y1+=sy1;
[FE1] 42: obs();};
[UK0] 43: t=0;
[A82] 44: while(!kbhit())
[FL6] 45: {line(p[t].x0,p[t].y0,p[t].x1,p[t].y1);
[RY0] 46: h=t-1;
[FS1] 47: if(h<0)h=nl;
[WR2] 48: line(x0,y0,x1,y1);
[VP3] 49: p[h].x0=x0;p[h].y0=y0;
[9B4] 50: p[h].x1=x1;p[h].y1=y1;t++;
[HV1] 51: if(t>nl)t=0;
[BD2] 52: x0+=sx0;x1+=sx1;
[ND2] 53: y0+=sy0;y1+=sy1;
[FE1] 54: obs();};
[R61] 55: getch();
[FG2] 56: closegraph();}
```

Listing 7. »teppich.c« stellt die Grafikfähigkeiten Ihres PC dar.

Farben mit ihren Attributen

Die erste Bildschirmseite von »farb_doc.pas« (Listing 8) stellt alle auf einer EGA/VGA-Farbgrafikkarte darstellbaren Farbkombinationen sowie den entsprechenden Wert des dazugehörigen Farbattributbytes dar. Die zweite Bildschirmseite enthält eine Erweiterung der ersten Seite. Auf dieser zweiten Seite werden alle »hellen Hintergrundfarben« der ersten Seite wirklich hell und nicht blinkend dargestellt (Bild 5).

Das Programm kann dazu verwendet werden, einen angeschlossenen Farbbildschirm optimal einzustellen, da alle potentiellen Farbkombinationen gleichzeitig auf eine Bildschirmseite gebracht werden. Zusätzlich kann der Programmierer mit ihm ausprobieren, welche Farbkombinationen für den Programmanwender lesbar sind, und welche Farben in einem Programm vermieden werden sollten.

Weiterhin zeigt es die Anwendung der Unterfunktion 3 der Funktion 10hex des Video-BIOS-Interrupts 10hex. Mit diesem Interrupt läßt sich das Blinkbit des Farbattributbytes in ein Intensitätsbit umwandeln. Enthält das BL-Register vor dem Interruptaufruf den Wert 1, wird Blinken ausgeschaltet, enthält es 0, wird Blinken wieder eingeschaltet.

Das Programm wird von der Kommandozeile aus durch Eingabe von »farben« gestartet. Durch Drücken einer beliebigen Taste gelangt man zur zweiten Bildschirmseite, durch erneutes Drücken einer Taste zurück zum Betriebssystem.

Farb_doc ist bereits ab Turbo Pascal Version 4.0 ohne Änderung zu kompilieren. Sie erhalten eine dokumentierte Langfassung aus dem stark frequentierten Rennen um den 1024-Wettbewerb. Zwei Befehle in einer Zeile sind vermieden, da dieser Trick, welcher zwei Byte für Zeilenumsbruch und Wagenrücklauf (»cr« und »lf«) spart, nicht der strukturierten Programmierung entspricht.

(Robert Hoffmann /et)

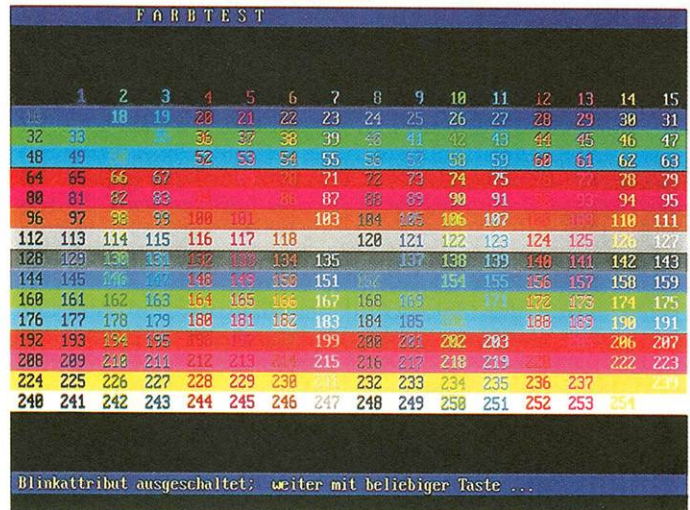


Bild 5. »farb_doc.pas« gibt die Farbwerte mit ihren Attributen aus.

```
[X44] 1: {Programm: farb_doc.pas
[FF5] 2:   Funktion: alle Farbkombinationen
[BV5] 3:   Sprache: Turbo Pascal ab Version 4.0
[YP3] 4:   Autor: Robert Hoffmann
[XX4] 5:   (c)1992 DMV Widuch GmbH & Co. KG
[U13] 6:   program FarbTest;
[S62] 7:   uses Crt,Dos;
[LE3] 8:   procedure TestBild;
[Z44] 9:   { Farbtabelle darstellen }
[UN5] 10: var h:Byte;{ Hintergrundfarbe }
[H06] 11: v:Byte;{ Zeichenfarbe "Vordergrund"}
[MU0] 12: begin
[KD5] 13:   ClrScr;{ Bildschirm löschen }
[KB2] 14:   TextAttr:=§1E;
[U45] 15:   { Farbe gelb auf blau setzen }
[GV1] 16:   GotoXY(1,1);
[3S5] 17:   { Cursor auf Anfang erste Zeile }
[5B1] 18:   ClrEol;
[FM5] 19:   { Bildschirmzeile blau füllen }
[T21] 20:   GotoXY(16,1);
[ZU4] 21:   { Programmtitel ausgeben }
[RB3] 22:   Write('F A R B T E S T ');
[YH2] 23:   for h:=0 to 15 do
[8Y5] 24:   { alle Hintergrundfarben von 0-15 }
[MU0] 25:   begin
[U52] 26:     GotoXY(1,h+5);
[UK2] 27:     for v:=0 to 15 do
[S35] 28:     { alle Zeichenfarben von 0-15 }
[MU0] 29:     begin
[804] 30:       TextAttr:=h SHL 4 OR v;
[S56] 31:       { Textattribut aus h und v setzen }
[PU3] 32:       Write((h SHL 4 OR v):4,' ');
[186] 33:       {Nr der Farbkombination ausgeben }
[4U4] 34:       end;{ nächste Zeichenfarbe }
[GK5] 35:     end;{ nächste Hintergrundfarbe }
[ZN0] 36:   end;
[ED5] 37: procedure BModeTxt(Txt:string);
[ZC4] 38: { Hinweiszeile ausgeben }
[MU0] 39: begin
[KB2] 40:   TextAttr:=§1E;
[U45] 41:   { Farbe gelb auf blau setzen }
[302] 42:   GotoXY(1,24);
[SH5] 43:   { Cursor auf Anfang der Zeile }
[5B1] 44:   ClrEol;
[FM5] 45:   { Bildschirmzeile blau füllen }
[402] 46:   GotoXY(2,24);
[1P5] 47:   { Hinweistext in Zeile ausgeben }
[K56] 48:   Write('Blinkattribut '+Txt+'; weiter'+
[F74] 49:   ' mit beliebiger Taste ...');
[ZN0] 50: end;
[000] 51:
```



```
[1F5] 52: procedure BlinkMode(Modus:Byte);
[554] 53: { Blinkbit bearbeiten }
[R63] 54: var reg:Registers;
[MU0] 55: begin
[KS1] 56:   reg.AH:=10;
[P95] 57:   { Funktion 10 des Interrupts }
[7T1] 58:   reg.AL:=3;
[WK6] 59:   { Unterfunktion 03 der Funktion 10 }
[DC2] 60:   reg.BL:=Modus;
[UB5] 61:   { Blinkbit setzen bzw. löschen }
[U32] 62:   Intr($10,reg);
[XC5] 63:   { BIOS-Interrupt 10 aufrufen }
[ZN0] 64: end;
[000] 65:
[MU0] 66: begin
[0L1] 67:   TestBild;
[R36] 68:   { Testbild mit blinkenden Zeichen }
[YD4] 69:   BModeTxt('eingeschaltet');
[E34] 70:   { Hinweistext ausgeben }
[P31] 71:   repeat
[7K5] 72:     { auf beliebige Taste warten ... }
[4N5] 73:   until KeyPressed AND(ReadKey<>'');
[PN5] 74:   { und Tastaturpuffer löschen }
[S12] 75:   BlinkMode(0);
[3D7] 76:   { Blinkbit in Intensitätsbit umschalten }
[4E4] 77:   BModeTxt('ausgeschaltet');
[E34] 78:   { Hinweistext ausgeben }
[P31] 79:   repeat
[7K5] 80:     { auf beliebige Taste warten ... }
[4N5] 81:   until KeyPressed AND(ReadKey<>'');
[PN5] 82:   { und Tastaturpuffer löschen }
[W12] 83:   BlinkMode(1);
[Y87] 84:   { Intensitätsbit in Blinkbit umschalten }
[NA2] 85:   TextAttr:=$07;
[GS5] 86:   { DOS-Standardfarben einstellen }
[KD5] 87:   ClrScr; { Bildschirm löschen }
[KN0] 88: end.
```

Listing 8. »farb.doc.pas« gibt Farbwerte mit den Attributen aus.

Atari-Bilder am PC

»getstad.pas« (Listing 9) dient dazu, Atari-St-Bilder für den PC zu konvertieren. Das im Atari-Bereich meistens verwendete Grafikformat ist das gepackte STAD-Bild. Die Endung dieses Formats lautet »pac«. Für den Atari gibt es sehr viele Grafiken, die somit auch dem PC-Anwender leicht zugänglich sind (Bild 6). Getstad zeigt ein PAC-Bild an und wartet auf einen Tastendruck, womit Sie die Arbeit abbrechen. Das gibt Ihnen ein Arbeitsmittel an die Hand, beispielsweise mit einem Capture-Programm von Word ein PC-Format des Bildes zu speichern.

Getstad erkennt alle Standard-Grafikkarten wie VGA, EGA und Hercules. Karten mit einer geringeren Auflösung als 640 x 400 Bildpunkte können nicht das ganze Bild darstellen. Achten Sie bitte darauf, den BGI-Grafiktreiber im glei-

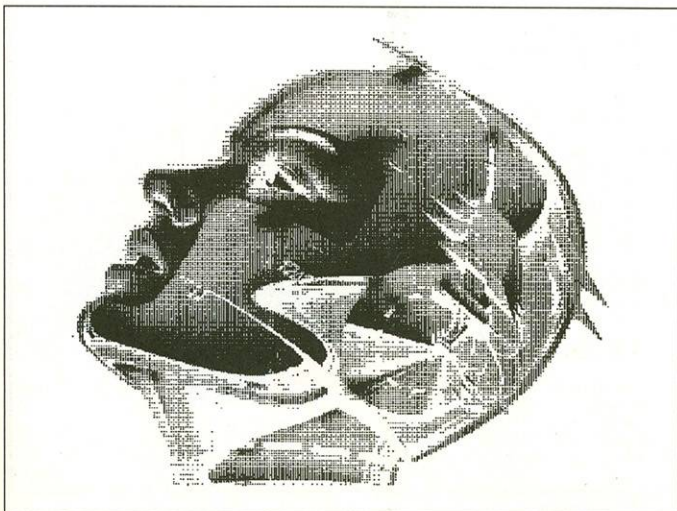


Bild 6. »getstad.pas« hat ein Atari- in ein PC-Bild umgewandelt.

chen Verzeichnis zu verwenden, in dem Sie auch Getstad starten. Die Syntax des Aufrufs belegt Listingzeile 101.

(Armin Daubmann/et)

```
[SU3] 1: {Programm: getstad.pas
[K56] 2: Funktion: Atari- in PC-Bilder wandeln
[BV5] 3: Sprache: Turbo Pascal ab Version 4.0
[EF3] 4: Autor: Armin Daubmann
[XX4] 5: (c)1992 DMV Widuch GmbH & Co. KG
[9P2] 6: uses Crt,Graph;
[WW1] 7: var F: file;
[SP5] 8:   FileName: string;C: Byte;Kennung:
[S05] 9:   string;NumRead: Integer;Buf:
[A35] 10:   array[1..32767]of Char;S: array
[T44] 11:   [0..7]of Byte;a3,d3: Word;
[V33] 12: procedure Graphic;
[8L4] 13: var GraphDriver,GraphMode,
[4B3] 14:   ErrorCode: Integer;
[MU0] 15: begin
[EK3] 16:   GraphDriver:=Detect;
[NZ5] 17:   InitGraph(GraphDriver,GraphMode,'');
[U74] 18:   ErrorCode:=GraphResult;
[3X3] 19:   if ErrorCode<>grOK then
[MU0] 20:     begin
[TK4] 21:       WriteLn('Graphics error: ',
[YG4] 22:         GraphErrorMsg(ErrorCode));
[AU3] 23:       ReadLn;Halt(1);end;end;
[U43] 24: procedure Male(B: Byte);
[WY1] 25: var L: Byte;
[MU0] 26: begin
[Z11] 27:   if B=255
[4F3] 28:   then Line(a3,d3,a3+7,d3)
[W64] 29:   else for L:=0 to 7 do begin
[EA3] 30:     if B=S[L]then begin
[933] 31:       PutPixel(a3+L,d3,0);
[FN3] 32:       B:=B-S[L];end;end;end;
[000] 33:
[TF3] 34: procedure Vertikal;
[M42] 35: var a2: Word;
[Z93] 36: d0,d1,d2,d4,d5,d6: Word;
[612] 37: d7: Integer;
[WU5] 38: label dcvr10,dcvr0,dcvr1,dcvr2,dcvr3,
[LJ3] 39:   dcvrloop,dcvrall,ende;
[MU0] 40: begin
[GB5] 41:   a2:=5;{ab 5. element von Buffer}
[EY3] 42:   a3:=0;{spalte}d2:=32000;
[604] 43:   d4:=Ord(Buf[a2]);Inc(a2);
[C04] 44:   d6:=Ord(Buf[a2]);Inc(a2);
[904] 45:   d5:=Ord(Buf[a2]);Inc(a2);
[CW2] 46:   d3:=0;dcvr10:=0;
[UZ3] 47:   d0:=Ord(Buf[a2]);Inc(a2);
[R04] 48:   if d4<>d0 then goto dcvr1;
[NV4] 49:   d1:=d6;d7:=Ord(Buf[a2]);Inc(a2);
[ZE3] 50:   goto dcvrloop;dcvr1:
[2V3] 51:   if d5=d0 then goto dcvr2;
[063] 52:   if d0=0 then Male(d0);
[U24] 53:   d7:=0;goto dcvrall;dcvr2:
[E54] 54:   d1:=Ord(Buf[a2]);Inc(a2);
[L84] 55:   if (d1=0)AND(Ord(Buf[a2])=0)
[AH2] 56:   then goto ende;
[UX4] 57:   dcvr3:=d7:=Ord(Buf[a2]);Inc(a2);
[DW4] 58:   dcvrloop: if d1>0 then Male(d1);
[HN2] 59:   dcvrall: Dec(d2);
[DK3] 60:   if d2=0 then goto ende;
[LA5] 61:   Inc(d3);if d3<>400 then goto dcvr0;
[FC4] 62:   d3:=0;a3:=a3+8;dcvr0: Dec(d7);
[SD4] 63:   if d7>-1 then goto dcvrloop;
[UX1] 64:   goto dcvr10;
[XT0] 65:   ende;
[ZN0] 66: end;
[YR3] 67: procedure Horizontal;
[L14] 68: var d0,d1,d4,d5,d6,d7: Word;
[ME3] 69: d2: Integer;a2: Word;
[DU4] 70: label dchr10,dchr1,dchr2,dchr3,
[KD2] 71:   dchrloop,ende;
[MU0] 72: begin
[GB5] 73:   a2:=5;{ab 5. element von Buffer}
[5T4] 74:   a3:=0;{spalte}d2:=32000;d3:=0;
[604] 75:   d4:=Ord(Buf[a2]);Inc(a2);
[C04] 76:   d6:=Ord(Buf[a2]);Inc(a2);
[904] 77:   d5:=Ord(Buf[a2]);Inc(a2);
[R55] 78:   dchr10:=0; d0:=Ord(Buf[a2]);Inc(a2);
[705] 79:   if d4<>d0 then goto dchr1;d1:=d6;
[F04] 80:   d7:=Ord(Buf[a2]);Inc(a2);
[ND3] 81:   goto dchrloop;dchr1:
[ST3] 82:   if d5=d0 then goto dchr2;
[063] 83:   if d0>0 then Male(d0);
[474] 84:   a3:=a3+8;if a3=640 then begin
[2N2] 85:     a3:=0;Inc(d3);end;
[KP4] 86:   Dec(d2);if d2<0 then goto ende;
[1W2] 87:   goto dchr10;dchr2:
[XZ3] 88:   d1:=Ord(Buf[a2]);Inc(a2);
[L84] 89:   if (d1=0)AND(Ord(Buf[a2])=0)
[AH2] 90:   then goto ende;
```

Listing 9. »getstad.pas« wandelt Atari- in PC-Bilder um.


```
[PV4] 91: dchr3: d7:=Ord(Buf[a2]);Inc(a2);
[NU4] 92: dchrloop: if d1>0 then Male(d1);
[474] 93: a3:=a3+8;if a3=640 then begin
[2N2] 94: a3:=0;Inc(d3);end;
[KP4] 95: Dec(d2);if d2<0 then goto ende;
[FX4] 96: if d7=0 then goto dchr10;Dec(d7);
[ZB3] 97: goto dchrloop;ende:
[ZN0] 98: end;
[RM3] 99: begin if ParamCount<>1
[ZN1] 100: then begin
[NY5] 101: WriteLn('Aufruf: GETSTAD bildname'+
[GN1] 102: '[.pac]');
[SN1] 103: Halt(1);end;
[0P3] 104: FileName:=ParamStr(1);
[173] 105: if Pos('.',FileName)=0
[2V5] 106: then FileName:=FileName+'.PAC';
[7L6] 107: ($I-)Assign(F,FileName);Reset(F,1);($I+)
[D14] 108: if IOResult<>0 then begin
[RV6] 109: WriteLn('Fehler beim Öffnen der Datei'+
[7P2] 110: '>',FileName,<'>');
[UW1] 111: Halt(1); end;
[5F4] 112: while NOT EOF(F)do BlockRead
[7A4] 113: (F,Buf,SizeOf(Buf),NumRead);
[AC1] 114: Graphic;
[W25] 115: SetColor(White);Bar(0,0,639,399);
[5R2] 116: SetColor(Black);
[RS3] 117: S[0]:=128;for C:=1 to 7 do
[G03] 118: S[C]:=S[C-1]DIV 2;
[000] 119:
[9F6] 120: Kennung:=Buf[1]+Buf[2]+Buf[3]+Buf[4];
[AX5] 121: if Kennung='pM86' then Vertikal else
[XC5] 122: if Kennung='pM85' then Horizontal
[EP3] 123: else begin CloseGraph;
[6T5] 124: WriteLn('Kein Atari-ST STAD Bild. ');
[T41] 125: Halt(1);
[ZN0] 126: end;
[P31] 127: repeat
[4X4] 128: until ReadKey<>'';CloseGraph;
[KN0] 129: end.
```

Listing 9. »getstad.pas« wandelt Atari- in PC-Bilder um.

Abrollende Kreise

»zykloide.pas« (Listing 10) zeichnet vier Kreise, die aufeinander abrollen. Auf dem inneren Kreis mit dem Radius »r1« rollt ein Kreis mit dem Radius »r2« ab. Auf dem umlaufenden Kreis läuft wiederum ein Kreis mit Radius »r3« ab, auf dem nun der Kreis mit dem Radius »r4« abläuft. In Listingzeile 25 finden Sie den Pfad-Verweis auf den BGI-Treiber, den Sie bitte Ihren Verhältnissen anpassen. Da die Einsendung dem 1024-Byte-Wettbewerb entstammt, bleiben Erweiterungen von Zykloide denkbar wie:

- Erhöhung der Anzahl der Kreise,
- Zeichnen der Mittelpunktbahnen,
- Ermittlung maximaler oder minimaler Punktbahngeschwindigkeiten.

Auch mit farblichen Experimenten können Sie dies kleine Grafikprogramm weiter ausgestalten.

(Carsten Kausch/et)

```
[N64] 1: (Programm: zykloide.pas
[EH5] 2: Funktion: abrollende vier Kreise
[YD4] 3: Sprache: Turbo Pascal ab 6.0
[CH3] 4: Autor: Carsten Kausch
[XX4] 5: (c)1992 DMV Widuch GmbH & Co. KG)
[F23] 6: program Zykloide;
[ZM2] 7: uses Graph,Crt;
[000] 8:
[706] 9: var x4,y4,x3,y3,x2,y2,x1,y1,r1,r2,r3,r4,
[SA3] 10: a,b,dt,t1,t2,t3: real;
[5Y2] 11: k,j,f,grafikmodus;
[6W3] 12: grafiktreiber:Integer;
[HE3] 13: procedure zeichnen;
[MU0] 14: begin
[GN3] 15: Circle(k,j,Trunc(r1*f));
[PH5] 16: Circle(Trunc(x2*f)+k,Trunc(y2*f*(-1))
[MB2] 17: +j,Trunc(r2*f));
[WH5] 18: Circle(Trunc(x3*f)+k,Trunc(y3*f*(-1))
[SB2] 19: +j,Trunc(r3*f));
[2K5] 20: Circle(Trunc(x4*f)+k,Trunc(y4*f*(-1))
[WB2] 21: +j,Trunc(r4*f));
[ZN0] 22: end;
[1U4] 23: begin grafiktreiber:=Detect;
```

```
[WK1] 24: InitGraph
[MX6] 25: (grafiktreiber,grafikmodus,'c:\TP6\bgi');
[B21] 26: k:=320;
[C21] 27: j:=240;
[ZV0] 28: r1:=9;
[0W0] 29: r2:=8;
[VV0] 30: r3:=4;
[UV0] 31: r4:=2;
[UP0] 32: f:=6;
[AP0] 33: a:=0;
[V71] 34: b:=2*Pi;
[XS1] 35: dt:=(b-a)/90;
[LW0] 36: t1:=a;
[P31] 37: repeat
[4F1] 38: t1:=t1+dt;
[Y62] 39: x1:=r1*cos(t1);
[N62] 40: y1:=r1*sin(t1);
[3X2] 41: x2:=(r1+r2)*sin(t1);
[BX2] 42: y2:=(r1+r2)*cos(t1);
[1W1] 43: t2:=r1/r2*t1;
[273] 44: x3:=x2+(r2+r3)*cos(t2);
[V63] 45: y3:=y2+(r2+r3)*sin(t2);
[AW1] 46: t3:=r2/r3*t2;
[473] 47: x4:=x3+(r3+r4)*sin(t3);
[L73] 48: y4:=y3+(r3+r4)*cos(t3);
[CT2] 49: SetColor(white);
[WH1] 50: zeichnen;
[5R2] 51: SetColor(black);
[WH1] 52: zeichnen;
[K13] 53: until KeyPressed;
[522] 54: CloseGraph;
[KN0] 55: end.
```

Listing 10. »zykloide.pas« läßt vier Kreise aufeinander abrollen.

Zweiter Mischer aller Farben

Das entsprechende Programm in der Februar-Ausgabe der DOS International, Seite 274, erfuhr mit »vga-mix.bas« (Listing 11) eine Überarbeitung. Sie arbeiten in dieser verbesserten Version ohne »gosub« in verschachtelten Schleifen, denn »vga-mix.bas« bedient sich effektiver einer einzigen Select-case-Anweisung. Die Palettenzuweisungen im ursprünglichen Programm macht vga-mix überflüssig, wobei Sie in diesem Programm zweckmäßiger mit Data-Zeilen arbeiten (Zeile 145). Aus diesen Zeilen lesen und zeichnen Sie besser in einer Schleife, statt dauernd mit Gosub-Befehlen springen zu müssen.

Doch »vga-mix.bas« wartet noch mit weiteren »Gimmicks« auf. So arbeitet »schweinchenrosa« diesmal mit einem weiteren Sechseckrahmen um die Sechsecke (Bild 7). In diesem Rahmenfenster sehen Sie die geänderte Farbe wie im Mittelfeld. Die Numerierung der Farbpalettenwerte und der Sechsecke dient hierbei der besseren Übersicht. Ebenfalls entfällt eine Notierung der gemischten Farbwerte, weil Sie Ihr Ergebnis einfach über den Drucker ausgeben lassen. Mit

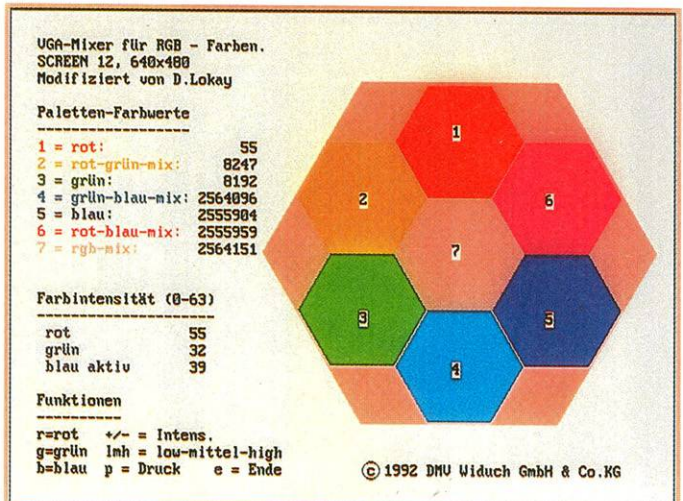


Bild 7. »vga-mix.bas« mischt die Farbwerte für »schweinchenrosa« noch komfortabler.

dem Tastendruck »p« beantworten Sie die Frage nach dem Namen Ihrer Mischfarbe, wobei Sie bitte zuvor den Drucker einschalten, weil »vga-mix.bas« hierfür keine Abfrage vorieht. Trotz dieser Zusätze kommt das verbesserte Programm ungefähr mit den gleichen Listingzeilen aus wie das ursprüngliche Programm.

(Detlef Lokay/et)

```
[9H3] 1: 'Programm: vga-mix.bas
[1B5] 2: 'Funktion: Farbmixer nach DOS 2'92
[2F3] 3: 'Sprache: Quick Basic
[CA3] 4: 'Autor: Detlef Lokay
[9R4] 5: '(c)1992 DMV Widuch GmbH & Co.KG
[B73] 6: SCREEN 12, 0, 0, 0: CLS
[ND5] 7: cc$ = " c 1992 DMV Widuch GmbH & Co.KG "
[T85] 8: ccl$ = "Modifiziert von D.Lokay"
[FU2] 9: PALETTE 0, 3289650:
[PU1] 10: PALETTE 9, 0
[F64] 11: blau = 32: gruen = 32: rot = 32
[5G3] 12: LINE (5, 5)-(634, 474), 11, B
[SB3] 13: LINE (6, 6)-(633, 473), 9, B
[U61] 14: COLOR 9
[8M1] 15: LOCATE 3, 5
[0N5] 16: PRINT "VGA-Mixer für RGB - Farben."
[HM5] 17: LOCATE 4, 5: PRINT "SCREEN 12, 640x480"
[9C3] 18: LOCATE 5, 5: PRINT ccl$
[F86] 19: LOCATE 7, 5: PRINT "Paletten-Farbwerte"
[L15] 20: LOCATE 8, 5: PRINT "-----"
[UL5] 21: COLOR 1: LOCATE 9, 5: PRINT "1 = rot:"
[T33] 22: COLOR 2: LOCATE 10, 5
[K24] 23: PRINT "2 = rot-grün-mix:"
[B06] 24: COLOR 3: LOCATE 11, 5: PRINT "3 = grün:"
[743] 25: COLOR 4: LOCATE 12, 5
[T54] 26: PRINT "4 = grün-blau-mix:"
[AV5] 27: COLOR 5: LOCATE 13, 5: PRINT "5 = blau:"
[N43] 28: COLOR 6: LOCATE 14, 5
[PU3] 29: PRINT "6 = rot-blau-mix:"
[W43] 30: COLOR 7: LOCATE 15, 5
[HW2] 31: PRINT "7 = rgb-mix:"
[U61] 32: COLOR 9
[AU6] 33: LOCATE 18, 5: PRINT "Farbintensität (0-63)"
[MG5] 34: LOCATE 19, 5: PRINT "-----"
[G35] 35: LOCATE 24, 5: PRINT "Funktionen"
[N84] 36: LOCATE 25, 5: PRINT "-----"
[866] 37: LOCATE 26, 5: PRINT "r=rot +/- = Intens."
[2N1] 38: LOCATE 27, 5
[YU5] 39: PRINT "g=grün lmh = low-mittel-high"
[4N1] 40: LOCATE 28, 5
[535] 41: PRINT "b=blau p = Druck e = Ende"
[BE3] 42: LOCATE 28, 43: PRINT cc$
[ML2] 43: CIRCLE (347, 439), 8
[0F1] 44: GOTO rot
[HR0] 45: scr:
[H33] 46: GOSUB screenchange
[N71] 47: change:
[3X2] 48: change$ = INPUT$(1)
[V33] 49: SELECT CASE change$
[P51] 50: CASE "e"
[PN0] 51: CLS
[091] 52: SCREEN 0
[4M0] 53: END
[A71] 54: CASE "r"
[VS0] 55: rot:
[UZ0] 56: fb$ = "r"
[U51] 57: COLOR 1
[MM1] 58: LOCATE 20, 5
[VT5] 59: PRINT " rot aktiv "; USING "#####"; rot
[PM1] 60: LOCATE 21, 5
[C16] 61: PRINT " grün "; USING "#####"; gruen
[SM1] 62: LOCATE 22, 5
[9B5] 63: PRINT " blau "; USING "#####"; blau
[HD1] 64: GOTO scr
[Y51] 65: CASE "g"
[FZ0] 66: fb$ = "g"
[261] 67: COLOR 3
[FD4] 68: LOCATE 20, 5: PRINT " rot "
[B45] 69: LOCATE 21, 5: PRINT " grün aktiv "
[NP4] 70: LOCATE 22, 5: PRINT " blau ";
[902] 71: GOTO change
[A51] 72: CASE "b"
[AZ0] 73: fb$ = "b"
[A61] 74: COLOR 5
[FD4] 75: LOCATE 20, 5: PRINT " rot "
[YM4] 76: LOCATE 21, 5: PRINT " grün "
[PY4] 77: LOCATE 22, 5: PRINT " blau aktiv "
[902] 78: GOTO change
[E21] 79: CASE "+"
[906] 80: IF fb$ = "r" AND rot < 63 THEN rot = rot +
1
[L27] 81: IF fb$ = "g" AND gruen < 63 THEN gruen = gr
uen + 1
[AG6] 82: IF fb$ = "b" AND blau < 63 THEN blau = blau
+ 1
```

```
[HD1] 83: GOTO scr
[P21] 84: CASE "-"
[556] 85: IF fb$ = "r" AND rot > 0 THEN rot = rot - 1
[NZ6] 86: IF fb$ = "g" AND gruen > 0 THEN gruen = gr
uen - 1
[BA6] 87: IF fb$ = "b" AND blau > 0 THEN blau = blau
- 1
[HD1] 88: GOTO scr
[271] 89: CASE "p"
[L83] 90: pl = pl + 1: zl = zl + 1
[5S1] 91: LOCATE 26, 43
[N37] 92: LINE INPUT "Ind. Farbname :", name$: LOCATE
28
[F76] 93: LPRINT " Paletten-Farbwerte Nr.:";
[NR4] 94: LPRINT pl; "Farbname : "; name$
[L35] 95: LPRINT "-----";
[C65] 96: LPRINT "-----"
[HS6] 97: LPRINT " 1 = rot :"; USING
"#####"; rot
[V38] 98: LPRINT " 2 = rot-grün-mix :"; USING
"#####"; colrg
[YT7] 99: LPRINT " 3 = grün :"; USING
"#####"; colgruen
[M28] 100: LPRINT " 4 = grün-blau-mix :"; USING
"#####"; colbg
[DE7] 101: LPRINT " 5 = blau :"; USING
"#####"; colblau
[L08] 102: LPRINT " 6 = rot-blau-mix :"; USING
"#####"; colrb
[PL7] 103: LPRINT " 7 = rgb-mix :"; USING
"#####"; colrgb
[518] 104: LPRINT " Grundfarbwert rot :"; USING
"#####"; rot
[AW8] 105: LPRINT " Grundfarbwert grün :"; USING
"#####"; gruen
[0A8] 106: LPRINT " Grundfarbwert blau :"; USING
"#####"; blau
[7W4] 107: LPRINT "-----"
[HS1] 108: LPRINT
[VF4] 109: LOCATE 26, 43: PRINT SPACE$(34)
[V95] 110: IF zl = 4 THEN LPRINT CHR$(12): zl = 0
[902] 111: GOTO change
[PF1] 112: CASE ELSE
[2F7] 113: IF change$ = "h" THEN rot = 63: gruen = 63:
blau = 63
[WF7] 114: IF change$ = "m" THEN rot = 32: gruen = 32:
blau = 32
[BX6] 115: IF change$ = "l" THEN rot = 0: gruen = 0: b
lau = 0
[HD1] 116: GOTO scr
[GS1] 117: END SELECT
[582] 118: screenchange:
[D34] 119: colrg = (rot + (gruen * 256))
[LL3] 120: colgruen = (gruen * 256)
[R35] 121: colbg = (gruen * 256) + (blau * 65536)
[7E3] 122: colblau = (65536 * blau)
[U34] 123: colrb = ((65536 * blau) + rot)
[L73] 124: colrgb = (65536 * blau)
[YL5] 125: colrgb = colrgb + ((256 * gruen) + rot)
[P35] 126: PALETTE 1, rot: PALETTE 2, colrg
[KZ5] 127: PALETTE 3, colgruen: PALETTE 4, colbg
[PT5] 128: PALETTE 5, colblau: PALETTE 6, colrb
[UR2] 129: PALETTE 7, colrgb
[U61] 130: COLOR 9
[CK5] 131: LOCATE 9, 24: PRINT USING "#####"; rot
[YZ5] 132: LOCATE 10, 24: PRINT USING "#####"; colrg
[UR1] 133: LOCATE 11, 24
[2M4] 134: PRINT USING "#####"; colgruen
[KY5] 135: LOCATE 12, 24: PRINT USING "#####"; colbg
[YR1] 136: LOCATE 13, 24
[S94] 137: PRINT USING "#####"; colblau
[KZ5] 138: LOCATE 14, 24: PRINT USING "#####"; colrb
[2S1] 139: LOCATE 15, 24
[P34] 140: PRINT USING "#####"; colrgb
[0K5] 141: LOCATE 20, 18: PRINT USING "#####"; rot
[216] 142: LOCATE 21, 18: PRINT USING "#####"; gruen
[UP5] 143: LOCATE 22, 18: PRINT USING "#####"; blau
[HM4] 144: sechsecke: RESTORE sechsecke
[5Y4] 145: DATA 337,394,7,400,180,1,308,233,2,308
[KS4] 146: DATA 339,3,400,392,4,492,339,5,492,233
[D32] 147: DATA 6,400,286,7
[0C5] 148: READ sx, sy, colnr: PSET (sx, sy)
[5X3] 149: DRAW "c=" + VARPTR$(colnr)
[VX4] 150: DRAW "ta0 r186 ta60 r186 ta120 r186"
[ZD5] 151: DRAW "ta180 r186 ta240 r186 ta300 r186"
[0A3] 152: PAINT (270, 230), colnr
[L44] 153: LINE (4, 4)-(635, 475), colnr, B
[GT2] 154: PAINT (1, 1), colnr
[X85] 155: FOR eck = 2 TO 8: READ sx, sy, colnr
[C02] 156: PSET (sx, sy)
[5X3] 157: DRAW "c=" + VARPTR$(colnr)
[CE4] 158: DRAW "ta0 r60 ta60 r60 ta120 r60"
[M05] 159: DRAW "ta180 r60 ta240 r60 ta300 r60"
[KB3] 160: LOCATE 8, 54: PRINT "1"
[N83] 161: LOCATE 12, 43: PRINT "2"
```

Listing 11. »vga-mix.bas« mischt komfortabel rote, grüne und blaue Farbwerte für VGA-Grafikkarten.


```
[793] 162: LOCATE 19, 43: PRINT "3"
[193] 163: LOCATE 22, 54: PRINT "4"
[N93] 164: LOCATE 19, 65: PRINT "5"
[B93] 165: LOCATE 12, 65: PRINT "6"
[K93] 166: LOCATE 15, 54: PRINT "7"
[HN3] 167: PAINT (sx, sy - 15), colnr
[3T2] 168: NEXT eck: RETURN
[000] 169:
```

Listing 11. »vga-mix.bas« mischt komfortabel rote, grüne und blaue Farbwerte für VGA-Grafikkarten.

Kurven in drei Dimensionen

»3d-funkt.bas« (Listing 12) kompilieren Sie mit Power Basic Version 2.10 oder mit Quick Basic. Wie der Name schon sagt, stellt es mathematische Funktionen dreidimensional dar (Bild 8). Zeile 7 gibt hierbei die Funktion vor:

$\cos(x) + \cos(2 \cdot x) + \cos(5 \cdot x)$.

Zum Experimentieren und Ausprobieren erhalten Sie in Tabelle 3 noch einige weitere Funktionsterme, bei denen teilweise eine Änderung der Variablen »l« und »z« erforderlich ist. »l« bestimmt dabei die Anzahl der zu zeichnenden Linien, »z« die Stärke der Linien. Setzen Sie »z« auf den Wert 1, so wird die Linie gepunktet, bei einem Wert von 0.5 wird sie durchgezogen gezeichnet. Die Änderung wird dann erforderlich, wenn eine »Verwischung« der Funktion auftritt, wenn also die eigentlich unsichtbaren, hinteren Linien teilweise noch zu erkennen sind.

Programmaufbau:

Nach dem Löschen des Bildschirms wird die Funktion über den Befehl »def fn« definiert, und es werden die Werte für »l« und »z« vorgegeben. Mit »SCREEN 12« schalten Sie sodann in den Grafikmodus der VGA-Karten mit einer Auflösung von 640 × 480 Bildpunkten um. Wollen Sie das Programm für andere Grafikformate verwenden, so sollten Sie bei der Berechnung von »x%« und »y%« die Werte 1,7 und 1,8 verändern; bei einer Auflösung von 320 × 200 Pixel, also der EGA-Auflösung mit »SCREEN 7«, setzen Sie beispielsweise die Werte 0.85 und 0.9 ein; zusätzlich sollten Sie auch die Werte von »xk« und »yk«, die für eine Korrektur in x- und y-Richtung zuständig sind, ändern, bei EGA: »xk = 20« und »yk = 12,5«.

Nun beginnt das eigentliche Programm:

Die For-Schleife »t« wird so oft durchlaufen, wie es die Variable »l« vorgibt. Mit der folgenden If-then-Konstruktion vermeiden Sie, daß – wie es bei Funktionen mit senkrechten Asymptoten (zum Beispiel: $1/x$) vorkommt – das Programm mit der Fehlermeldung

division by zero error

aussteigt.

Das Programm errechnet dann in der Variablen »r«, wie lang die aktuelle Linie ist, wozu die For-Schleife »i« benötigt wird. »P« bestimmt den aktuellen Funktionswert des zu zeichnenden Punktes, durch »x%« und »y%« wird seine Lage auf dem Grafikbildschirm errechnet. Mit »PSET« wird

er anschließend gezeichnet, und mit dem »Line-Befehl« wird erreicht, daß eigentlich unsichtbare hintere Linien »hidden lines« gelöscht werden. Sie können »3d-funkt.bas« jederzeit mit einer Taste und der ESC-Taste unterbrechen. Am Ende wird die Hintergrundfarbe des Bildschirms geändert und ein Tastendruck und die Esc-Taste erwartet. Der Grafikbildschirm wird schließlich wieder abgeschaltet.

(Martin Lehmann/et)

Funktionsterm	»l-Wert«	»z-Wert«
$\cos(1/x) + \cos(2/x) + \cos(5/x) - 3$	72	1
$\sin(x) + \sin(2 \cdot x) + \sin(5 \cdot x)$	72	1
$\cos(\sin(x))$	72	1
$\cos(\sin(2 \cdot x))$	72	1
$\cos(\sin(i/x))$	128	1
$\cos(\sin(2 \cdot x + i))$	72	1
$\sin(\cos(2 \cdot x))$	72	1
$\cos(2 \cdot x) + \cos((x + i)/16)$	72	1
$\text{sqr}(\text{abs}(0.5 \cdot (16 - x \cdot x))) + 1/(x \cdot 4)$	72	1
$\cos(4 \cdot x) + 20/(x \cdot x + 3)$	72	1
$-\text{abs}(1/x)$	72	1
$-\text{abs}(1/(x + i))$	72	1
$1/\text{sqr}(x) - 3$	72	0.5

Tabelle 3. In »3d-funkt.bas« können Sie auch mit diesen Funktionen experimentieren.

```
[FX3] 1: 'Programm: 3d-funkt.bas
[DR5] 2: 'Funktion: dreidimensionale Kurve
[ZF3] 3: 'Sprache: Quick Basic
[YR3] 4: 'Autor: Maritn Lehmann
[FP4] 5: '(c) 1992 DMV Widuch GmbH & Co.KG
[PN0] 6: CLS
[666] 7: DEF fna (x) = COS(x) + COS(2 * x) + COS(5 * x)
[N44] 8: l = 128: z = 1: xk = 40: yk = 30
[000] 9:
[5C1] 10: SCREEN 12
[HR4] 11: FOR t = -144 TO 144 STEP (288 / l)
[FN2] 12: IF t = 0 THEN t = .1
[284] 13: r = INT(.5 + SQR(20736 - t * t))
[5E3] 14: FOR i = -r TO r STEP z
[A05] 15: p = fna(SQR(i * i + t * t) * .0327) * 20
[KN4] 16: x% = INT(1.7 * (i + t / 2.25 + 160)) + x
      k
[TH5] 17: y% = INT(1.8 * (199 - (p - t / 2.25 + 90))) + yk
[GB2] 18: PSET (x%, y%), 15
[LP3] 19: LINE (x%, y% + 1) - (x%, 480), 0
[MW2] 20: IF instat THEN END
[U11] 21: NEXT i
[V21] 22: NEXT t
[000] 23:
[7M1] 24: COLOR 15, 8
[NB0] 25: DO
[922] 26: LOOP UNTIL instat
[091] 27: SCREEN 0
[4M0] 28: END
```

Listing 12. »3d-funkt.bas« zeichnet dreidimensionale Kurven.

VGA-Grafikkarte im Geschwindigkeitsrausch

Stellen Sie sich vor, Sie befinden sich im Grafik-Modus, und Sie merken es nicht! Unmöglich – werden Sie sagen, alleine der große Geschwindigkeitsverlust der Textausgabe gegenüber dem Textbildschirm zeigt sich immer sehr deutlich. Doch stellen Sie sich weiter vor, daß Sie ein Grafik-Fenster öffnen, den Hintergrund speichern und dabei nicht ein Byte Ihres kostbaren Arbeitsspeichers verbrauchen, und nebenbei geschieht dies viel schneller als mit »putimage/getimage«. Unmöglich – und doch bietet Ihnen »fastvga.pas« (Listing 13) all dies und einiges mehr.

Fastvga arbeitet nur mit einer VGA-Karte der Auflösung 640 × 480 Punkten bei 16 Farben und enthält Routinen für folgende Bereiche:

- Speichern und Wiederherstellen von Bildschirmabschnitten

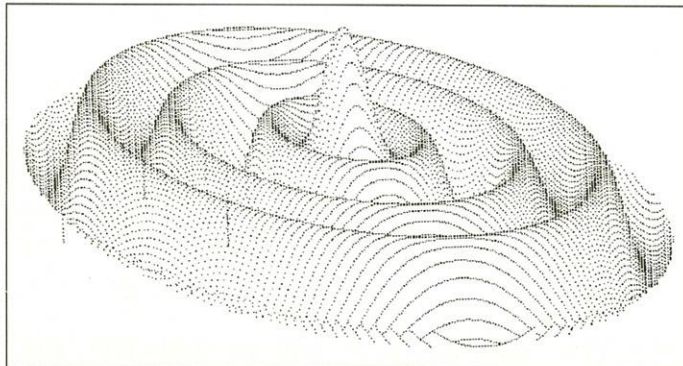


Bild 8. »3d-funkt.bas« hat die Kurve aus Zeile 7 gezeichnet.

- Textausgabe mit Zwei-Farben-Option für Menüeinträge
- Horizontale und vertikale Linien
- Icon-Definition und Ausgabe
- Punkte zeichnen
- Ausgefüllte Balken zeichnen

Sie erzielen einen Geschwindigkeitsgewinn um einen Faktor von etwa 1,5 bis fast 8 gegenüber entsprechenden Befehlen der Unit »graph«. Wo liegt das Geheimnis von Fastvga? Wesentlich ist die Spezialisierung auf den VGA-Standard und auf einzelne Routinen.

So können »showtext« und »fasttextxy« nur im »Default-Zeichensatz« ausgeben, den »outtext« gewöhnlich benutzt. Für die meisten Anwendungsfälle genügt dies völlig. Belohnt werden Sie dafür mit einer erheblichen Geschwindigkeitssteigerung. »fasttextxy« ist sogar so weit spezialisiert, daß der Text nur auf einem 80 Zeichen mal 30 Zeilen messenden Bildschirm ausgegeben wird. Auf diese Weise lassen sich auch die Geschwindigkeitsgewinne der anderen Routinen erklären.

Eine weitere Besonderheit ergibt sich aus folgender Überlegung: Die VGA-Karte besitzt einen Bildschirmspeicher von 256 KByte. Für die Auflösung $640 \times 480 = 80 \text{ Byte mal } 480$ Punkte ergibt sich ein Speicherbedarf von 38 400 Byte. Um 16 Farben darstellen zu können, benötigt jeder Punkt weitere 4 Bit. Es ergibt sich ein Speicherbedarf von 150 KByte. 106 KByte bleiben also ungenutzt. Entscheidend ist, daß einfach und schnell Grafikdaten innerhalb des Bildschirmspeichers kopiert werden können. Daher liegt nahe, diesen Bereich zum Speichern von Bildschirm-Ausschnitten zu nutzen. Da der Maustreiber noch einen kleinen Teil dieses Speichers benutzt, verkleinert sich der Bereich ein wenig. Doch jetzt konkrete Informationen zu den einzelnen Routinen. Die Argumente sind, soweit es sich nicht um Zeiger handelt, vom Typ integer. »st« kennzeichnet eine Zeichenkette.

Vor dem Aufruf von »initfast« oder »getcharacters« stellen Sie Speicherplatz für den Zeichensatz zur Verfügung. Einen Zeiger auf diesen Bereich übergeben Sie »setcharmem«.

```
setcharmem(pointer^ )
```

Da immer der ganze Zeichensatz geladen wird, müssen Sie 256×8 Byte zur Verfügung stellen, beispielsweise:

```
getmem(pchars,256*8)
(* 256 Zeichen mit 8 Bytes *)
setcharmem(pchars^ )
(*mit pchars: pointer*)
```

Sie können anstelle eines Zeigers auch ein entsprechendes »Array of Byte« verwenden. Ein Zeichen wird, beginnend mit der ersten Zeile, Byte für Byte abgelegt. Jedes Zeichen benötigt 8 Byte. Sie können also durch Veränderung des Bitmusters Ihren eigenen Zeichensatz definieren.

Die Prozedur »initfast« wird einmal vor der Benutzung der Grafik-Routinen (nach »setcharmem«) aufgerufen. Vor dem Aufruf muß Initgraph ausgeführt und der Modus $640 \times 480 \times 16$ gesetzt worden sein. Um nun den Zeichensatz nicht selbst definieren zu müssen, wird in der linken oberen Ecke des Bildschirms der Original-Default-Zeichensatz mit »outtext« ausgegeben und vom Bildschirm in den mit »setcharmem« definierten Speicherbereich gelesen. Sie sehen davon nichts, da Text und Hintergrund schwarz sind. Rufen Sie also »initfast« auf, bevor sie Ihren Bildschirm gestalten. Der Zeichensatz wird eingelesen und der

Speicherbeginn für »get-window« festgelegt. Den Zeichensatz 8×8 »getcharacters« liest Turbo Pascal ein. Diese Routine ruft »initfast« automatisch auf. »setvga« aktiviert den Modus $640 \times 480 \times 16$. »clearscreen« löscht den gesamten Bildschirm mit der aktuellen Hintergrundfarbe. Die Prozedur ist etwa dreimal schneller als ein »clear-viewport« für den gesamten Bildschirm. Die Hintergrundfarbe legt wie üblich »setbkcolor« fest. »resetvga« setzt die Register der VGA-Karte auf die Default-Werte zurück, was die Schreib- und Lesemodi wieder in den Standardzustand schaltet. »resetwinbuffer« setzt den Speicher für die Bildschirmdaten zurück, was den Zeiger auf den freien Bereich für den Startwert setzt. Dieser Speicherbereich befindet sich innerhalb des Teiles des Bildschirmspeichers, der nicht auf dem Monitor sichtbar ist. Den Befehl

```
plot(x, y, farbe)
```

nutzen Sie als Ersatz für »putpixel«, wobei Sie etwa um den Faktor 2,35 an Geschwindigkeit gewinnen. Ein eventuell vorhandener Viewport wird berücksichtigt. Der Befehl

```
solidbar(x1, y1, x2, y2)
```

füllt ein Rechteck in der aktuellen Füllfarbe. Diese bestimmen Sie mit:

```
setfillstyle(SolidFill, Farbe)
```

Der vorhandene Viewport wird berücksichtigt. Die Breite sollte mindestens 2 Byte betragen. Die Routine arbeitet etwa 1,6 mal schneller als »bar«. Mit der Routine

```
sethotcolor(farbe)
```

legen Sie die Farbe für die Buchstaben der Textausgabe fest, die in geschweiften Klammern eingeschlossen sind. Nehmen Sie beispielsweise an, Sie möchten einen Menüeintrag mit dem Inhalt »Save as...« ausgeben. Der Buchstabe »a« in »as« soll jedoch in einer anderen Farbe auffallen. Dies ist bei der Verwendung von »showtext« sehr einfach. Die zweite Farbe legen Sie mit »sethotcolor« fest. Die Textausgabe sieht nun so aus:

```
showtext ('Save {a}s...');
```

Den hervorzuhebenden Text schließen also einfach geschweifte Klammern ein. Die aktuelle Farbe »gethotcolor«, die Sie mit »sethotcolor« gesetzt haben, wird ermittelt.

»showtext(st)« gibt die Zeichenkette in der aktuellen Farbe an der Grafik-Cursor-Position aus. Startpunkt ist die linke obere Ecke. Buchstaben, die Sie mit »{« einschließen, werden in der mit »sethotcolor« festgelegten Farbe ausgegeben. Dies brauchen Sie, um in Menüs Tasten zu markieren. Showtext unterstützt nur den Default-Zeichensatz, den aber etwa viermal so schnell wie »outtext«. Genauso arbeitet auch

```
showtextxy(X,Y,st)
```

jedoch mit Koordinatenübergabe. Ein eigenes Koordinatensystem mit den X-Bereichen von 1 bis 80 und den Y-Bereichen von 1 bis 30 hingegen verwendet:

```
fasttextxy(X,Y,st)
```

»fasttext« verwendet die linke obere Ecke mit der Koordinate (1,1). Den Text in »st« erhalten Sie in der aktuellen Farbe. Text in geschweiften Klammern hebt die in »sethotcolor« definierte Farbe hervor. »fasttext« arbeitet etwa 7,7 mal schneller als »outtext«. Mit

```
getwindow(x1,y1,x2,y2)
```

kopieren Sie den angegebenen Bildausschnitt hinter das Ende des VGA-Speichers, wobei »x1,y1« die linke, obere Ecke und »x2,y2« die rechte untere Ecke darstellt. Rufen Sie »getwindow« mehrmals auf, so werden die Ausschnitte auf einem Stapel abgelegt. Wenn Sie also drei Ausschnitte gespeichert haben, liegen diese in folgender Reihenfolge am Ende des Bildschirmspeichers:

```
getwindow(...) 1. Ausschnitt holen
getwindow(...) 2. Ausschnitt holen
getwindow(...) 3. Ausschnitt holen
```

Der nicht sichtbare Bildschirmspeicher hält die Daten im dritten Ausschnitt an der größten Adresse »TOS = Top Of

Wert »nr«	Zuweisung für
1	80x240
2	80x120
3	80x60
4	80x30

Tabelle 4. Mit »setlineshift(nr)« verändern Sie das Koordinatensystem in Y-Richtung.

Stack«. Die Informationen für die Größe der Ausschnitte werden ebenfalls automatisch auf diesem Stapel abgelegt. Den zuletzt mit »getwindow« eingelesenen Bildschirmausschnitt geben Sie an der Position »x,y« (Ecke links oben) mit

putwindow(x,y)
aus. Für das vorherige Beispiel bedeutet dies, daß Sie mit »putwindow(..) den dritten Ausschnitt wieder herstellen, dann den zweiten und so weiter. Dieses Verfahren erlaubt es, daß sich die Fenster überlappen dürfen. Es wird also immer das Fenster geschlossen, welches sich an oberster Stelle befindet. Gegenüber »getimage/putimage« ergibt sich eine Geschwindigkeits-Steigerung um einen Faktor von etwa 5,6!

Mit

hline(x1,y1,x2)
zeichnen Sie eine horizontale Linie von »x1,y1« nach »x2,y1«. Die Zeichenfarbe ist die aktuelle Vordergrundfarbe. Die Routine arbeitet für diesen Spezialfall etwa neunmal schneller als »line«. Genauso zeichnen Sie mit

vline(x1,y1,y2)
eine vertikale Linie von »x1,y1« nach »x1,y2«. Mit

readicon(filename,pointer+)
lesen Sie ein Icon im Format 8×3 aus der Textdatei »icon.txt« in den durch den Zeiger adressierten Bereich ein, wobei diese Textdatei beispielsweise folgende Daten enthalten kann:

```
...xxx..  
...xxx..  
..xxxxxxx
```

Das Zeichen ».« bedeutet keinen Punkt, das »x« bedeutet einen Punkt. Der benötigte Speicherplatz ergibt sich aus X-Punkten dividiert durch acht mal Y-Punkte. Die Breite sollte ein Vielfaches von acht sein. Für dieses Beispiel erhalten Sie 83 = 24, beispielsweise:

```
getmem(picon,24);  
readicon('Icon.txt',picon^ )  
[mit picon: pointer]
```

Breite und Höhe werden automatisch aus der Text-Datei ermittelt. Die aktuelle Icongröße legen Sie mit

```
seticonsize(Breite, Höhe)  
manuell fest. Breite und Höhe sind vom Typ Byte. Im 80 mal 30 Koordinatensystem geben Sie mit
```

```
fasticonxy(x,y,pointer^ )  
das Icon aus, dessen Daten ab »pointer^ « zu finden sind. Sollen mehrere Farben verwendet werden, so legen Sie Icons übereinander. Das Icon wird in der aktuellen Grafikfarbe ausgegeben, die Sie mit »setcolor« festgelegt haben. Abschließend verändern Sie noch mit
```

```
setlineshift(nr)  
das Koordinatensystem von 80 mal 30 in Y-Richtung. Die Werte für »nr« entnehmen Sie hierbei Tabelle 4.
```

Die Anwendung der einzelnen Routinen lassen sich auch mit Hilfe von »fastdemo.pas« (Listing 14) verdeutlichen. Wenn Sie also beispielsweise ein Menüsystem programmieren wollen, könnte der Einsatz von Fastvga so aussehen:

1. Bildausschnitt sichern mit »getwindow«
2. Ausschnitt füllen mit »solidbar«
3. Rahmen zeichnen mit »hline« und »vline«
4. Menütext ausgeben mit »showtext« oder »fasttextxy«
5. Angewählten Menüpunkt mit »solidbar« hervorheben und erneute Textausgabe.

Um Fastvga platzsparend abdrucken zu können, finden Sie möglichst viele Assemblerbefehle in einer Listingzeile. Die Datei »icon.txt« legen Sie gesondert an, da Fastdemo darauf in Zeile 134 zugreifen muß.

(Klaus Eberle/et)

```
[SE3] 1: {Datei: fastvga.pas  
[0D5] 2: Funktion: Unit schneller als BGI  
[HV5] 3: Sprache: Turbo Pascal ab Version 6.0  
[R93] 4: Autor: Klaus Eberle  
[XX4] 5: (c)1992 DMV Widuch & Co. KG)  
[K72] 6: unit fastvga;  
[SG6] 7: interface uses Dos,Graph;(Daten init)  
[2F3] 8: procedure initfast;  
[HE6] 9: {Offset für Daten zurücksetzen}  
[RD4] 10: procedure resetwinbuffer;  
[737] 11: {8x8 Zeichensatz in chars einlesen, zuvor  
[A97] 12: nur initgraph und setgraphmode ausführen}  
[W74] 13: procedure getcharacters;  
[UX4] 14: procedure setcharm(var ch);  
[WU4] 15: {Grafikmodus 640x480 setzen}  
[903] 16: procedure setvga;  
[827] 17: {Bildschirm (gesamt) mit Farbe 0 löschen,  
[EW6] 18: etwa 3 mal schneller als clearviewport}  
[XU3] 19: procedure clearscreen;  
[FC5] 20: {Aktuellen Viewport löschen}  
[BK4] 21: procedure clearscreenpart;  
[H57] 22: {VGA-Register auf default Werte setzen}  
[0F3] 23: procedure resetvga;  
[AB6] 24: {Punkt mit Farbe an x,y ausgeben, etwa  
[S65] 25: 2.36 mal schneller als Putpixel}  
[U57] 26: procedure plot(x,y:Integer;farbe:Integer);  
[PB7] 27: {gefülltes Rechteck zeichnen, in akt.Farbe,  
[7R6] 28: Viewport-Verschiebung berücksichtigt:  
[AN6] 29: xoben,yoben,xunten,yunten, min. 2 Bytes,  
[P95] 30: etwa 1.58 mal schneller als bar}  
[5L6] 31: procedure solidbar(x1,y1,x2,y2:Integer);  
[1W6] 32: {Text an akt. Grafik-Position in akt. Vor-  
[ZU6] 33: dergrundfarbe. Bezugspunkt: Links Oben.  
[ET7] 34: Vor Anwendung 1 mal getcharacters aufrufen}  
[546] 35: { * Buchstaben in {} eingeschlossen*}  
[257] 36: {die folgenden Buchstaben in der Farbe, mit  
[S87] 37: SETHOTCOLOR festgelegt -> für Menüs, etwa  
[CB5] 38: 3.85 mal schneller als Outtext}  
[4K5] 39: procedure showtext(Str:string);  
[VB7] 40: {Wie schowtext, mit Koordinatenübergabe}  
[CT3] 41: procedure showtextxy  
[X34] 42: (x,y:Integer;st:string);  
[C85] 43: { *2. Textfarbe für Chars in {} *}  
[5U5] 44: procedure sethotcolor(c:Integer);  
[665] 45: function gethotcolor:Integer;  
[K67] 46: {Das letzte mit GETWINDOW geholte Fenster  
[FK6] 47: an x1,y1 ausgegeben, Speicher wird frei}  
[T26] 48: procedure putwindow(x1,y1:Integer);  
[YN7] 49: {Zwischenspeichern und Zurückschreiben des  
[K26] 50: Ausschnittes. x1,y1: linke obere Ecke  
[EL6] 51: Nur getwindow: x2,y2: linke untere Ecke  
[S86] 52: Daten ans Ende des Bildschirmspeichers  
[X47] 53: Immer ganze Bildschirmbytes übertragen.)  
[W77] 54: {Mehrere Aufruf von getwindow stapelt die  
[M66] 55: Fenster, und gibt sie mit put im LIFO-  
[H56] 56: Prinzip (putwindow) aus - etwa 5.6 mal  
[VM4] 57: schneller als Put/Getimage!}  
[K26] 58: procedure getwindow(x1,y1,x2,y2:Integer);  
[PS6] 59: {Vertikale Linie zeichnen von x1,y1 nach  
[DB6] 60: x1,y2 in akt. Farbe, Gewinn 1.2 bei line}  
[2P5] 61: procedure vline(x1,y1,y2:Integer);  
[4Y6] 62: {Horizontale Linie von x1,y1 nach x2,y1 in  
[LN6] 63: aktueller Farbe, Gewinn 9 mal als Line}  
[6M5] 64: procedure hline(x1,y1,x2:Integer);  
[8M6] 65: {Text im Koordinatensystem x:1..80 y:1..30  
[FN6] 66: 1,1 links Oben, Übergabe Koordinaten und  
[Z37] 67: String, Zeichnen in aktueller Farbe, View-  
[ZN7] 68: port berücksichtigt, Gewinn 7.7 ->Outtext!}  
[GR3] 69: procedure fasttextxy  
[SA4] 70: (x,y:Integer;Str:string);  
[HS6] 71: {ICON aus Textdatei lesen und in Bitfeld  
[PX6] 72: schreiben, Textformat: x=Punkt, .=Kein Pkt  
[9Y5] 73: IconSize automatisch ermittelt}  
[563] 74: procedure readicon  
[9K4] 75: (fname: string; var field);  
[DH7] 76: {Festlegen der Icongröße in Punkten, x-  
[N88] 77: Richtung umgerechnet in vollständige Bytes}  
[ZF7] 78: procedure SetIconSize(Ibreite,Ihoehe:Byte);  
[6W6] 79: {Icon-Ausgabe im 80x30 Koordinatensystem}  
[NA7] 80: (x,y: Koordinate,IconBuf: Bitfeld Zeiger)  
[YM3] 81: procedure FastIconXY  
[4A4] 82: (x,y:Integer;var iconbuf);  
[GZ5] 83: {Zeilenabstand für FAST-Routinen:  
[3D5] 84: 1=Faktor 1, 2=Faktor 4, 3=Faktor 8,  
[906] 85: 4=Faktor 16 (standard)-> ergibt ein 30  
[7N3] 86: Zeilen Koord. System}  
[DE5] 87: procedure setlineshift(l:Byte);  
[XN5] 88: implementation var keyfarbe:Byte;  
[B32] 89: {Hotcolor}  
[V07] 90: winbuffer:Word;(Offset Grafik - Speicher)  
[CW6] 91: Chars:pointer;(Zeiger auf Zeichenfeld)  
[846] 92: hoehe:Byte;(aktuelle Icongröße)  
[7V4] 93: breite:Byte;lineshift:Byte;  
[ED6] 94: {Schiebefaktor für Zeilenberechnung}  
[W74] 95: procedure getcharacters;  
[PD2] 96: var i:Integer;  
[MU0] 97: begin  
[UY1] 98: SetColor(1);  
[265] 99: SetFillStyle(solidfill,black);
```



```
[UY3] 100: SetPalette(1,black);I:=0;
[NB2] 101: while I<>2048 do
[MU0] 102: begin
[GU1] 103:   MoveTo(0,0);
[WL3] 104:   OutText(Chr(I DIV 8));
[FN0] 105:   asm
[HE5] 106:     push ds;lds si,Chars;mov ax,$A000
[Z94] 107:     mov es,ax;add si,I;mov di,0
[706] 108:     mov ax,es:[di];mov[si],ax;add di,80
[PG5] 109:     Inc si;mov ax,es:[di];mov[si],ax
[275] 110:     add di,80;Inc si;mov ax,es:[di]
[TL4] 111:     mov[si],ax;add di,80;Inc si
[706] 112:     mov ax,es:[di];mov[si],ax;add di,80
[PG5] 113:     Inc si;mov ax,es:[di];mov[si],ax
[275] 114:     add di,80;Inc si;mov ax,es:[di]
[TL4] 115:     mov[si],ax;add di,80;Inc si
[DG4] 116:     mov ax,es:[di];mov[si],ax
[275] 117:     add di,80;Inc si;mov ax,es:[di]
[2T5] 118:     mov[si],ax;add di,80;Inc si;pop ds
[ZN0] 119:   end;
[C03] 120:   Inc(I,8);Bar(0,0,8,8);
[ZN0] 121: end;
[NF2] 122: SetPalette(1,1);
[2A5] 123: SetFillStyle(solidfill,white);
[ZN0] 124: end;
[UX4] 125: procedure setcharmem(var ch);
[554] 126: begin Chars:=Addr(ch);end;
[2F3] 127: procedure initfast;
[MU0] 128: begin
[WV6] 129:   getcharacters;resetwinbuffer;resetvga;
[ZD6] 130:   sethotcolor(red);setlineshift(4);end;
[RD4] 131: procedure resetwinbuffer;
[WS3] 132: begin winbuffer:=40800;
[BE6] 133:   ((38401) Differenz für den Maustreiber )
[ZN0] 134: end;
[903] 135: procedure setvga;
[6Y2] 136: var r:Registers;
[MU0] 137: begin
[051] 138:   r.ah:=0;
[D14] 139:   r.al:=12;(mode 640 x 480)
[VR1] 140:   Intr($10,r);
[ZN0] 141: end;
[XU3] 142: procedure clearsreen;
[B52] 143: label weiter;
[MU0] 144: begin
[FN0] 145:   asm
[3T3] 146:     mov dx,$3CE;mov ax,$0005
[U75] 147:     {05: Modus, 00=Read/Write Mode 0}
[7C3] 148:     out dx,ax;mov ax,$0003
[1L5] 149:     {03: Function select, 0=ersetzen}
[4G3] 150:     out dx,ax;mov ax,$FF08
[4E4] 151:     {08: Bitmap, ff=alle Bits}
[YB3] 152:     out dx,ax;mov ax,$0000
[ES4] 153:     {00: Colorcode, farbe 0 in AH}
[RC3] 154:     out dx,ax;mov ax,$0F01
[U07] 155:     {01: Verknüpfung, 0f=Farbe aus Colorcode}
[ZF3] 156:     out dx,ax;mov bx,38399
[B45] 157:     {Anzahl der Bildschirmbytes}
[PA6] 158:     push ds;mov ax,$A000 {Video-Segment}
[1H1] 159:     mov ds,ax
[UH6] 160:     @weiter:Übertragen in VGA Speicher}
[CY1] 161:     mov[bx],al
[NP6] 162:     {alle überschreiben->kein Einlesen}
[7W4] 163:     Dec bx {der Latches nötig}
[752] 164:     jnz @weiter
[ED7] 165:     mov[bx],al {letztes Byte noch ergänzen}
[S96] 166:     pop ds;mov ax,$0001 {01: Verknüpfung}
[YH1] 167:     out dx,ax
[C86] 168:     XOR al,al {00: Colorcode, farbe in AH}
[633] 169:     mov ah,0;out dx,ax
[4M0] 170:   end
[ZN0] 171: end;
[U57] 172: procedure plot(x,y:Integer;farbe:Integer);
[WE4] 173: var limits:viewporttype;
[MU0] 174: begin
[7C4] 175:   GetViewSettings(limits);
[FN0] 176:   asm
[7D5] 177:     mov ax,$A000{Video Speicher Anf}
[X86] 178:     mov es,ax {Aktuelles Datensegment}
[H94] 179:     mov ax,y{Zeile nach ax}
[Vb7] 180:     add ax,limits.y1 {Viewportverschiebung}
[W17] 181:     cmp ax,480 {Test Bereichsüberschreitung}
[674] 182:     jae @ende {dann Abbruch}
[E84] 183:     mov bx,ax {und nach bx}
[WX3] 184:     mov cl,6{6 mal schieben}
[884] 185:     SHL bx,cl {also mal 64}
[MX3] 186:     mov cl,4{4 mal schieben}
[284] 187:     SHL ax,cl {also mal 16}
[G75] 188:     add bx,ax {Summe ist Zeile * 80}
[E64] 189:     mov ax,x{Spalte holen}
[RB7] 190:     add ax,limits.x1 {Viewportverschiebung}
[P17] 191:     cmp ax,640 {Test Bereichsüberschreitung}
[RR3] 192:     jae @ende {dann ende}
[ZE5] 193:     mov cl,al {Lo-Byte X-Pos merken}
[KG5] 194:     SHR ax,1{x-drei mal rechts=/ 8}
[DX6] 195:     SHR ax,1;SHR ax,1;add bx,ax {bx=Offset}
[6T2] 196:     AND cl,7{Mod 8}
[UA5] 197:     XOR cl,7{7- x Mod 8, Bitnummer}
```

```
[1S3] 198:   mov dx,$3CE {Portadr}
[S04] 199:   mov al,$08 {Bitmask-Reg}
[AS5] 200:   mov ah,1{ein Bit zum Verschieben}
[DF4] 201:   SHL ah,cl {Bit auswählen}
[C46] 202:   out dx,ax;mov ax,$0205{Writemode 2}
[DY5] 203:   out dx,ax;mov ax,$0003{Replace Mode}
[LV6] 204:   out dx,ax;mov al,es:[bx]{Latches laden}
[9P6] 205:   mov ax,farbe;mov es:[bx],al;mov ax,$0005
[LV6] 206:   out dx,ax;mov ax,$0001{01: Verknüpfung}
[YH1] 207:   out dx,ax
[C86] 208:   XOR al,al {00: Colorcode, farbe in AH}
[633] 209:   mov ah,0;out dx,ax
[301] 210: @ende:
[4M0] 211: end
[ZN0] 212: end;
[0F3] 213: procedure resetvga;
[MU0] 214: begin
[FN0] 215:   asm
[NA4] 216:     mov dx,$3CE{Portadresse}
[8K6] 217:     mov ax,$FF08 {Bitmaske: Alle Bits (FF)}
[YH1] 218:     out dx,ax
[ZG6] 219:     mov ax,$0005 {05: Modus: Write/Read: 0}
[YH1] 220:     out dx,ax
[YR6] 221:     mov ax,$0003 {03: Verknüpfung: Ersetzen}
[YH1] 222:     out dx,ax
[E05] 223:     mov ax,$0000 {00: Farbcode auf 0}
[UA6] 224:     out dx,ax;mov ax,$0001 {Freigabe auf 0}
[FR1] 225:     out dx,ax;
[4M0] 226:   end
[ZN0] 227: end;
[5L6] 228: procedure solidbar(x1,y1,x2,y2:Integer);
[XD3] 229: var startmask:Byte;
[CA2] 230: endmask:Byte;
[BD4] 231: filler:fillsettingstype;
[5S3] 232: limits:viewporttype;
[MU0] 233: begin
[7C4] 234:   GetViewSettings(limits);
[K94] 235:   GetFillSettings(filler);
[FN0] 236:   asm
[Z33] 237:     mov ax,x2;cmp ax,x1
[SC5] 238:     jae @xok {wenn x2>=x1 dann xok}
[PP4] 239:     mov dx,x1{sonst tauschen}
[T93] 240:     mov x1,ax;mov x2,dx
[3W0] 241:   @xok:
[343] 242:     mov ax,y2;cmp ax,y1
[V85] 243:     jae @yok {Wenn y2>=y1 dann yok}
[RP4] 244:     mov dx,y1{sonst tauschen}
[Z93] 245:     mov y1,ax;mov y2,dx
[6W0] 246:   @yok:
[FA4] 247:     mov ax,y1{Zeile nach ax}
[X47] 248:     add ax,limits.y1 {Viewport verschiebung}
[W17] 249:     cmp ax,480 {Test Bereichsüberschreitung}
[Z94] 250:     jae @ende{dann Abbruch}
[2X3] 251:     mov di,ax{und nach di}
[994] 252:     mov cl,6 {6 mal schieben}
[SP3] 253:     SHL di,cl{also mal 64}
[Z84] 254:     mov cl,4 {4 mal schieben}
[YR3] 255:     SHL ax,cl{also mal 16}
[XE5] 256:     add di,ax{Summe ist Zeile * 80}
[3G4] 257:     mov ax,x1{Spalte holen}
[RB7] 258:     add ax,limits.x1 {Viewportverschiebung}
[P17] 259:     cmp ax,640 {Test Bereichsüberschreitung}
[6F3] 260:     jae @ende{dann ende}
[EW5] 261:     mov cl,al{Lo-Byte X-Pos merken}
[YC5] 262:     SHR ax,1 {x-drei mal rechts=/ 8}
[3T2] 263:     SHR ax,1;SHR ax,1
[9N5] 264:     add di,ax{di=Offset für Video}
[934] 265:     AND cl,7 {x1 low Mod 8}
[154] 266:     mov al,$FF {laden mit 255}
[G97] 267:     SHR al,cl{cl mal nach rechts für start}
[WU2] 268:     mov startmask,al
[1X5] 269:     {Startmaske sichern erstes Byte}
[8E4] 270:     mov cx,x2{x-ende holen}
[H67] 271:     add cx,limits.x1 {Viewport-Verschiebung}
[6L2] 272:     AND cl,7 {mod 8}
[C63] 273:     XOR cl,7 {7- rest}
[154] 274:     mov al,$FF {Laden mit 255}
[6E5] 275:     SHL al,cl{cl mal links schieben}
[EU6] 276:     mov endmask,al {Bitmaske letztes Byte}
[UR6] 277:     mov dx,$3CE {Portadr für Video-Karte}
[ZU1] 278:     mov ax,$0005
[U75] 279:     {05: Modus, 00=Read/Write Mode 0}
[7C3] 280:     out dx,ax;mov ax,$0003
[1L5] 281:     {03: Function select, 0=ersetzen}
[YH1] 282:     out dx,ax
[406] 283:     XOR al,al{00: Colorcode, farbe in AH}
[N74] 284:     mov ah,Byte{filler.color}
[RC3] 285:     out dx,ax;mov ax,$0F01
[U07] 286:     {01: Verknüpfung, 0f=Farbe aus Colorcode}
[YH1] 287:     out dx,ax
[KB5] 288:     push ds{Sichern Datensegment}
[GW4] 289:     mov ax,$A000 {Video-Segment}
[3H1] 290:     mov es,ax
[177] 291:     {als aktuelles Extrasegment für stosb}
```

Listing 13. »fastvga.pas« gibt Ihnen eine Unit als Ersatz für Ihre BGI-Treiber, die mit Assembler-Tuning erheblichen Geschwindigkeitsgewinn bringt.


```
[9P6] 292: mov ds,ax(Video A000 ist akt. Datenseg.)
[PN5] 293: mov cx,x2;add cx,limits.x1;SHR cx,1
[4L6] 294: SHR cx,1;SHR cx,1 (x2 in Bytes, d.h /8)
[CN3] 295: mov ax,x1(x1 nach ax)
[SG5] 296: add ax,limits.x1;SHR ax,1;SHR ax,1
[224] 297: SHR ax,1 (x1 in Bytes, d.h /8)
[DN3] 298: sub cx,ax(Länge +1)
[EP2] 299: Dec cx (Länge)
[MC6] 300: mov bx,y2;sub bx,y1(Höhe-1 ermitteln)
[AN3] 301: Inc bx (Höhe nach bx)
[K65] 302: mov al,$08 (08: Bitmap-Register)
[NM7] 303: mov ah,startmask (Maske für erste Spalte)
[VF7] 304: out dx,ax(Ins Maskenregister übertragen)
[X65] 305: push di(Video-Offset sichern)
[334] 306: push bx(Breite sichern)
[R06] 307: mov si,$50 (Zeilenabstand 80 Bytes)
[NE5] 308: @first:(Erste Spalte ausgeben)
[KT4] 309: mov ah,[di](Latches laden)
[GZ7] 310: mov[di],ah(Neue Registerwerte übertragen)
[DC4] 311: add di,si (Nächste Zeile)
[Z24] 312: Dec bx (Höhe=Höhe - 1)
[GV1] 313: jnz @first
[MX6] 314: {Wenn noch nicht Spalte fertig, springe}
[WT3] 315: pop bx (Höhe holen)
[M15] 316: pop di (alter Start-Offset)
[XM5] 317: Inc di (für Last Berechnung +1)
[A06] 318: cmp cx,81(Breite >=80, dann 0 Breite)
[G12] 319: jae @breite0
[S05] 320: push di(und wieder sichern)
[R76] 321: push bx(Höhe zwischenspeichern)
[0V6] 322: add di,cx(Zeiger auf Blockende-1 in di)
[U77] 323: mov ah,endmask (Bitmaske für Blockende)
[VN5] 324: out dx,ax(Ins Bit-Mask-Register)
[S06] 325: @last:(letzte Spalte beschreiben)
[AG5] 326: mov ah,[di](Latches lesen und)
[CV5] 327: mov[di],ah(verändert zurück)
[DC4] 328: add di,si (nächste Zeile)
[KP2] 329: Dec bx (Höhe -1)
[9L1] 330: jnz @last
[U77] 331: {wenn Höhe<>0 dann nächstes Blockstück}
[ZP5] 332: pop bx (Höhe wiederherstellen)
[PN4] 333: pop di (Blockanfang holen)
[DX6] 334: mov ah,$FF (Bitmaske für Blockinneres)
[5W5] 335: out dx,ax(Ins Bitmask-Register)
[MV5] 336: mov ds,cx(Breite in ds speichern)
[HF1] 337: sub si,cx
[XZ5] 338: {80-Breite=Summand für neue Zeile}
[444] 339: cld(aufwärts zählen)
[U91] 340: @zeile:
[WU6] 341: rep stosb (Breite * Bytes übertragen)
[FA5] 342: add di,si (Neuer Zeilenanfang)
[WW4] 343: mov cx,ds (Anzahl der Bytes)
[WT2] 344: Dec bx (Höhe - 1)
[SU1] 345: jnz @zeile
[TG1] 346: @breite0:
[G77] 347: pop ds (Datensegment wieder herstellen)
[301] 348: @ende:
[165] 349: mov ax,$0001 (01: Verknüpfung)
[YH1] 350: out dx,ax
[406] 351: XOR al,al(00: Colorcode, farbe in AH)
[633] 352: mov ah,0;out dx,ax
[ZN0] 353: end;
[ZN0] 354: end;
[BK4] 355: procedure clearscreentpart;
[WE4] 356: var limits:viewporttype;
[MU0] 357: begin
[7C4] 358: GetViewSettings(limits);
[586] 359: SetFillStyle(solidfill,GetBkColor);
[ZF6] 360: with limits do solidbar(0,0,x2-x1,y2-y1);
[ZN0] 361: end;
[ZF7] 362: procedure SetIconSize(Ibreite,Ihoehe:Byte);
[MU0] 363: begin
[PS3] 364: breite:=Ibreite DIV 8;
[SP5] 365: if Ibreite MOD 8<>0 then Inc(breite);
[ZB2] 366: hoehe:=Ihoehe;
[ZN0] 367: end;
[DE5] 368: procedure setlineshift(1:Byte);
[MU0] 369: begin
[472] 370: lineshift:=1;
[137] 371: (y der FAST-Routinen nach links schieben)
[ZN0] 372: end;
[YM3] 373: procedure FastIconXY
[4A4] 374: (x,y:Integer;var iconbuf);
[HD6] 375: var limits:viewporttype;farbe:Byte;
[MU0] 376: begin
[W57] 377: GetViewSettings(limits);farbe:=GetColor;
[3E5] 378: asm;push ds;push bp;mov ax,$A000
[P56] 379: mov es,ax(Aktuelles Extrasegment)
[UR6] 380: mov dx,$3CE (Portadr für Video-Karte)
[ZU1] 381: mov ax,$0005
[285] 382: (05: Modus, 02=Read/Write Mode 2)
[H43] 383: out dx,ax;mov ax,$03
[1L5] 384: (03: Function select, 0=ersetzen)
[RC3] 385: out dx,ax;mov ax,$0F01
[U07] 386: (01: Verknüpfung, 0F=Farbe aus Colorcode)
[YH1] 387: out dx,ax
[X66] 388: XOR al,al(00: Colorcode, farbe in AH)
[ZL4] 389: mov ah,farbe (Farbe laden)
```

```
[YH1] 390: out dx,ax
[G55] 391: mov ax,y (Zeile (1..30) nach ax)
[H23] 392: Dec ax;mov cl,lineshift
[FA6] 393: (shift 4=mal 16 für Zeilenabstand 16)
[2C5] 394: SHL ax,cl(also 4 mal nach links)
[VB7] 395: add ax,limits.y1 (Viewportverschiebung)
[C71] 396: add ax,4
[4G6] 397: (Zentrierung 30 Zeilen- Bildmitte + 4)
[ZH1] 398: cmp ax,472
[T78] 399: (Test Bereichsüberschreitung noch 8 Punkte)
[294] 400: jae @ende(dann Abbruch)
[723] 401: mov bx,ax(und nach bx)
[YR3] 402: SHL ax,cl(also mal 16)
[994] 403: mov cl,6 (6 mal schieben)
[183] 404: SHL bx,cl(also mal 64)
[3D5] 405: add bx,ax(Summe ist Zeilen * 80)
[SF7] 406: mov ax,limits.x1 (Viewportverschiebung)
[W16] 407: SHR ax,1 (Byte Versch. ermitteln, /8)
[064] 408: SHR ax,1;SHR ax,1;add ax,x
[V53] 409: (Spalte addieren)
[4C3] 410: Dec ax(Beginn bei 0)
[S17] 411: cmp ax,80(Test Bereichsüberschreitung)
[6F3] 412: jae @ende(dann ende)
[3M3] 413: add bx,ax(bx=Offset)
[183] 414: mov dx,$3CE (Portadr)
[935] 415: mov cl,breite(Breite laden)
[6F1] 416: XOR ch,ch
[ZC4] 417: mov si,80(Zeilenabstand)
[SS5] 418: sub si,cx(Summand für neue Zeile)
[6B4] 419: mov ch,hoehe (Höhe Laden)
[FE6] 420: lds di,iconbuf (Zeiger auf Icon-Daten)
[ND1] 421: mov al,08
[1A1] 422: @lines:
[PG1] 423: mov bp,cx
[9B1] 424: @bytes:
[K35] 425: mov ah,[di](Bitmaske laden)
[G54] 426: out dx,ax;mov ah,es:[bx]
[PC6] 427: mov es:[bx],ah (in Zielladr.schreiben)
[CK3] 428: Inc bx (neues Ziel)
[ZF3] 429: Inc di (neue Quelle)
[WX2] 430: Dec cl (Breite -1)
[5D5] 431: jnz @bytes (Bis Zeile kopiert)
[MN3] 432: add bx,si(neue Zeile)
[ZD4] 433: mov cx,bp;Dec ch(Höhe -1)
[ZB6] 434: jnz @lines (bis alle Zeilen kopiert)
[301] 435: @ende:
[DG5] 436: pop bp;pop ds;mov ax,$0001;out dx,ax
[4M0] 437: end
[ZN0] 438: end;
[5W2] 439: procedure storeb
[G16] 440: (var field:number;Word;value:Byte);
[MU0] 441: begin
[1R5] 442: asm;push ds (Datenseg. sichern)
[8S6] 443: lds di,field (Adr. des Datenfeldes holen)
[L76] 444: add di,number (Byte-Index berechnen)
[BV6] 445: XOR ah,ah;mov al,value (Wert ins Array)
[VZ2] 446: mov[di],ax;pop ds
[ZN0] 447: end;
[ZN0] 448: end;
[VH6] 449: function getbyte(var st:string):Byte;
[UT3] 450: var bytest:string[8];
[YT4] 451: erg:Byte;maske:Byte;I:Byte;
[MU0] 452: begin
[1H3] 453: bytest:=Copy(st,1,8);
[992] 454: Delete(st,1,8);
[XT2] 455: maske:=128;erg:=0;
[AF3] 456: for I:=1 to 8 do begin
[4A5] 457: (Stringteil in Byte umwandeln)
[8H3] 458: if bytest[I]='x' then
[K33] 459: erg:=erg OR maske;
[153] 460: maske:=maske DIV 2;
[ZN0] 461: end;
[T92] 462: getbyte:=erg;
[ZN0] 463: end;
[KF7] 464: procedure ReadIcon(fname:string;var field);
[D26] 465: var f:Text;st:string;rest:Integer;
[9Y5] 466: I:Integer;aktuell:Word;hoehe:Byte;
[N32] 467: breite:Byte;
[9F3] 468: begin Assign(f,fname);
[9Y2] 469: Reset(f);hoehe:=0;
[PY1] 470: aktuell:=0;
[P31] 471: repeat
[332] 472: ReadLn(f,st);
[355] 473: rest:=7 AND(8-Length(st)MOD 8);
[K25] 474: for I:=1 to rest do st:=st+'.';
[UW1] 475: Inc(hoehe);
[V63] 476: breite:=Length(st);
[7N3] 477: while Length(st)<>0 do
[MU0] 478: begin
[9U5] 479: storeb(field,aktuell,getbyte(st));
[N72] 480: Inc(aktuell);
[ZN0] 481: end;
[322] 482: until Eof(f);
[3G4] 483: SetIconSize(breite,hoehe);
[ZN0] 484: end;
[5U5] 485: procedure sethotcolor(c:Integer);
[MU0] 486: begin
[622] 487: keyfarbe:=c;
```



```
[ZNØ] 488: end;
[665] 489: function gethotcolor:Integer;
[MUØ] 490: begin
[DY3] 491:   gethotcolor:=keyfarbe;
[ZNØ] 492: end;
[4K5] 493: procedure showtext(Str:string);
[9A2] 494: const hoehe=8;
[WE4] 495:   var limits:viewporttype;
[Ø14] 496:   farbe:Byte;x,y:Integer;
[MUØ] 497: begin
[7C4] 498:   GetViewSettings(limits);
[FN5] 499:   farbe:=GetColor;x:=GetX;y:=GetY;
[U94] 500:   MoveTo(x+Length(Str)*8,y);
[FNØ] 501:   asm
[UR6] 502:     mov dx,$3CE (Portadr für Video-Karte)
[2U1] 503:     mov ax,$ØØØ5
[U75] 504:     {Ø5: Modus, ØØ=Read/Write Mode Ø}
[543] 505:     out dx,ax;mov al,$Ø3
[1L5] 506:     {Ø3: Function select, Ø=ersetzen}
[RC3] 507:     out dx,ax;mov ax,$ØFØ1
[UØ7] 508:     {Ø1: Verknüpfung, Øf=Farbe aus Colorcode}
[X93] 509:     out dx,ax;XOR al,al
[NF4] 510:     {ØØ: Colorcode, farbe in AH}
[ZL4] 511:     mov ah,farbe (Farbe laden)
[YH1] 512:     out dx,ax
[7L6] 513:     lea di,Str (Laden Zeiger String-Start)
[WC6] 514:     mov dl,ss:[di](Länge String in dl)
[C85] 515:     mov ah,$AØ (Video Speicher Anf)
[P56] 516:     mov es,ax(Aktuelles Extrasegment)
[D94] 517:     mov ax,y (Zeile nach ax)
[VB7] 518:     add ax,limits.y1 (Viewportverschiebung)
[W17] 519:     cmp ax,48Ø (Test Bereichsüberschreitung)
[294] 520:     jae @ende(dann Abbruch)
[7Z3] 521:     mov bx,ax(und nach bx)
[994] 522:     mov cl,6 (6 mal schieben)
[1S3] 523:     SHL bx,cl(also mal 64)
[Z84] 524:     mov cl,4 (4 mal schieben)
[YR3] 525:     SHL ax,cl(also mal 16)
[6G5] 526:     add bx,ax(Summe ist Zeile * 8Ø)
[1F4] 527:     mov ax,x (Spalte holen)
[RB7] 528:     add ax,limits.x1 (Viewportverschiebung)
[P17] 529:     cmp ax,64Ø (Test Bereichsüberschreitung)
[6F3] 530:     jae @ende(dann ende)
[EW5] 531:     mov cl,al(Lo-Byte X-Pos merken)
[YC5] 532:     SHR ax,1 (x-drei mal rechts=/ 8)
[3T2] 533:     SHR ax,1;SHR ax,1
[3M3] 534:     add bx,ax(bx=Offset)
[BØ1] 535:     Inc di
[4K7] 536:     {Zeiger auf nächsten String-Buchstaben}
[UR1] 537:     @nextchar:
[CW5] 538:     mov al,ss:[di](Buchstabe nach al)
[VH5] 539:     cmp al,'(' (Testlauf Buchstaben)
[VU2] 540:     jne @nextzeichen
[MV2] 541:     Dec dl (Länge -1)
[ZC1] 542:     push dx
[1L6] 543:     mov dx,$3CE(Portadr für Video-Karte)
[X66] 544:     XOR al,al{ØØ: Colorcode, farbe in AH}
[VV4] 545:     mov ah,keyfarbe(Farbe laden)
[6R2] 546:     out dx,ax;pop dx
[BØ1] 547:     Inc di
[4K7] 548:     {Zeiger auf nächsten String-Buchstaben}
[CW5] 549:     mov al,ss:[di](Buchstabe nach al)
[X82] 550:     @nextzeichen:
[HR2] 551:     XOR ah,ah (ah=Ø)
[RE5] 552:     SHL ax,1 (Tabellenberechnung:)
[5S2] 553:     SHL ax,1;SHL ax,1
[M25] 554:     {Mal 8, da 8 Byte pro Buchstabe}
[8P4] 555:     push dx(sichern Länge dl)
[755] 556:     push bx(Pos Ausgabe sichern)
[HK6] 557:     push di(Zeiger auf Buchstaben sichern)
[SF3] 558:     push ds;lds si,Chars
[E64] 559:     {Zeichentabellenstart}
[ØD1] 560:     add si,ax
[9F6] 561:     {si: Zeiger auf auszugebende Grafik}
[YK4] 562:     AND cl,7 (Low byte x mod 8)
[Y84] 563:     mov dx,$3CE(Video Port)
[EØ5] 564:     mov ch,hoehe (8 Bytes Höhe)
[MT1] 565:     @nextbyte:
[5G1] 566:     mov di,cx
[X76] 567:     {16 bit Register AX mit Grafikbyte +
[6F3] 568:     ØØØØØØØØ(1) laden und cl mal nach rechts
[7Ø7] 569:     schieben. Man erhält beide Bitmasken für
[ØW6] 570:     beide Teile, bei Grafik zw. zwei Bytes}
[ØK5] 571:     mov ah,[si](Grafik-Byte nach ah)
[1X2] 572:     XOR al,al(al=Ø)
[VF1] 573:     SHR ax,cl
[Z25] 574:     {cl (low x mod 8) mal nach rechts
[BK4] 575:     Zustand: Bitmasken in ah,al
[E86] 576:     Zeiger auf Grafik in si, auf Video in bx
[BT6] 577:     Stringlänge in dl,Zeig. auf Buchst. in di
[4T3] 578:     Ch: Höhe Buchstabe}
[4S5] 579:     mov ch,al(Maske rechts sichern)
[1W4] 580:     mov al,Ø8(Bit Mask-Register)
[CX5] 581:     out dx,ax(Neue Bitmaske setzen)
[RG2] 582:     mov ah,es:[bx]
[5X4] 583:     {Latches Lesen und schreiben}
[BK2] 584:     mov es:[bx],ah
[ØU4] 585:     mov ah,ch(rechte Bitmaske)
```

```
[4V4] 586:     out dx,ax(Bitmaske setzen)
[MØ6] 587:     Inc bx (nächste Speicheradresse)
[RG2] 588:     mov ah,es:[bx]
[BZ4] 589:     {Latches benachbartes Byte}
[PA7] 590:     mov es:[bx],ah;add bx,79(nächste Zeile)
[3A5] 591:     Inc si (nächstes Grafik-Byte)
[PL6] 592:     mov cx,di(Grafik-Byte-Zähler holen)
[K54] 593:     Dec ch (Byte-Zähler - 1)
[CC2] 594:     jnz @nextbyte
[E93] 595:     pop ds;pop di;pop bx
[GF4] 596:     Inc bx (nächste Spalte)
[BØ1] 597:     Inc di
[4K7] 598:     {Zeiger auf nächsten String-Buchstaben}
[G76] 599:     segss mov al,[di](Buchstabe nach al)
[TL1] 600:     cmp al,'('
[KY6] 601:     {Test auf Ende hervorgehobene Buchstaben}
[L42] 602:     jne @nochhot
[X66] 603:     XOR al,al{ØØ: Colorcode, farbe in AH}
[ZL4] 604:     mov ah,farbe (Farbe laden)
[RE6] 605:     out dx,ax;Inc di;pop dx;Dec dl;push dx
[NK1] 606:     @nochhot:
[WP3] 607:     pop dx (Stringlänge)
[Z23] 608:     Dec dl (Länge - 1)
[KA2] 609:     jnz @nextchar
[3Ø1] 610:     @ende:
[6X1] 611:     mov dx,$3CE
[165] 612:     mov ax,$ØØØ1 {Ø1: Verknüpfung}
[YH1] 613:     out dx,ax
[C86] 614:     XOR al,al {ØØ: Colorcode, farbe in AH}
[YH1] 615:     out dx,ax
[ZNØ] 616: end;
[ZNØ] 617: end;
[CT3] 618: procedure showtextxy
[X34] 619:   (x,y:Integer;st:string);
[MUØ] 620: begin
[815] 621:   MoveTo(x,y);showtext(st);end;
[GR3] 622: procedure fasttextxy
[SA4] 623:   (x,y:Integer;Str:string);
[9A2] 624: const hoehe=8;
[WE4] 625:   var limits:viewporttype;
[NY1] 626:   farbe:Byte;
[MUØ] 627: begin
[W57] 628:   GetViewSettings(limits);farbe:=GetColor;
[FNØ] 629:   asm
[UR6] 630:     mov dx,$3CE (Portadr für Video-Karte)
[2U1] 631:     mov ax,$ØØØ5
[U75] 632:     {Ø5: Modus, ØØ=Read/Write Mode Ø}
[543] 633:     out dx,ax;mov al,$Ø3
[1L5] 634:     {Ø3: Function select, Ø=ersetzen}
[RC3] 635:     out dx,ax;mov ax,$ØFØ1
[UØ7] 636:     {Ø1: Verknüpfung, Øf=Farbe aus Colorcode}
[YH1] 637:     out dx,ax
[X66] 638:     XOR al,al{ØØ: Colorcode, farbe in AH}
[325] 639:     mov ah,farbe (Farbe setzen)
[YH1] 640:     out dx,ax
[7L6] 641:     lea di,Str (Laden Zeiger String-Start)
[WC6] 642:     mov dl,ss:[di](Länge String in dl)
[C85] 643:     mov ah,$AØ (Video Speicher Anf)
[P56] 644:     mov es,ax(Aktuelles Extrasegment)
[G55] 645:     mov ax,y (Zeile (1..3Ø) nach ax)
[NL4] 646:     Dec ax(Beginn bei Zeile Ø)
[ZS2] 647:     mov cl,lineshift
[S65] 648:     {=4=mal 16 für Zeilenabstand 16}
[ZC5] 649:     SHL ax,cl(also 4 mal nach links)
[VB7] 650:     add ax,limits.y1 (Viewportverschiebung)
[C71] 651:     add ax,4
[4G6] 652:     {Zentrierung 3Ø Zeilen- Bildmitte + 4}
[ZH1] 653:     cmp ax,472
[T78] 654:     {Test Bereichsüberschreitung noch 8 Punkte}
[294] 655:     jae @ende(dann Abbruch)
[7Z3] 656:     mov bx,ax(und nach bx)
[YR3] 657:     SHL ax,cl(also mal 16)
[994] 658:     mov cl,6 (6 mal schieben)
[1S3] 659:     SHL bx,cl(also mal 64)
[3D5] 660:     add bx,ax(Summe ist Zeilen * 8Ø)
[SF7] 661:     mov ax,limits.x1 (Viewportverschiebung)
[W16] 662:     SHR ax,1 (Byte Versch. ermitteln, /8)
[5P6] 663:     SHR ax,1;SHR ax,1;add ax,x (plus Spalte)
[4C3] 664:     Dec ax(Beginn bei Ø)
[S17] 665:     cmp ax,8Ø(Test Bereichsüberschreitung)
[6F3] 666:     jae @ende(dann ende)
[3M3] 667:     add bx,ax(bx=Offset)
[445] 668:     mov cl,dl(Länge String in cl)
[TØ7] 669:     Inc di(Zeiger auf 1. String-Buchstaben)
[UR6] 670:     mov dx,$3CE (Portadr für Video-Karte)
[UR1] 671:     @nextchar:
[NG2] 672:     mov al,ss:[di]
[Z56] 673:     {Buchstabe nach al, String auf stack}
[NL1] 674:     cmp al,'('
[PE6] 675:     {Test auf hervorgehobenen Buchstaben}
[VU2] 676:     jne @nextzeichen
[KV2] 677:     Dec cl (Länge -1)
[X66] 678:     XOR al,al{ØØ: Colorcode, farbe in AH}
[VV4] 679:     mov ah,keyfarbe(Farbe laden)
```

Listing 13. »fastvga.pas« gibt Ihnen eine Unit als Ersatz für Ihre BGI-Treiber, die mit Assembler-Tuning erheblichen Geschwindigkeitsgewinn bringt.


```
[YH1] 680: out dx,ax
[B01] 681: inc di
[4K7] 682: {Zeiger auf nächsten String-Buchstaben}
[CW5] 683: mov al,ss:[di]{Buchstabe nach al}
[X82] 684: @nextzeichen:
[HR2] 685: XOR ah,ah {ah=0}
[RE5] 686: SHL ax,1 {Tabellenberechnung:}
[S52] 687: SHL ax,1;SHL ax,1
[M25] 688: {Mal 8, da 8 Byte pro Buchstabe}
[BC1] 689: push ds
[U87] 690: {Datensegment für farbe,keyfarbe sichern}
[K12] 691: lds si,Chars
[276] 692: {Offset und DS von Chars-Daten laden}
[0D1] 693: add si,ax
[9F6] 694: {si: Zeiger auf auszugebende Grafik}
[VH5] 695: push bx{Bild-Offset speichern}
[LY4] 696: mov ch,hoehe {8 Byte Höhe}
[8F1] 697: mov al,ch
[MW6] 698: {Reg 8: Bit-Mask-Select nur if hoehe=8}
[MT1] 699: @nextbyte:
[0K5] 700: mov ah,[si]{Grafik-Byte nach ah}
[CX5] 701: out dx,ax{Neue Bitmaske setzen}
[AD7] 702: mov ah,es:[bx]{Latches lesen und zurück}
[HA7] 703: mov es:[bx],ah;add bx,80{nächste Zeile}
[3A5] 704: inc si {nächstes Grafik-Byte}
[K54] 705: dec ch {Byte-Zähler - 1}
[CC2] 706: jnz @nextbyte
[G55] 707: inc di {Nächster Buchstabe}
[CW5] 708: mov al,ss:[di]{Buchstabe nach al}
[8P5] 709: pop bx {alte Bildschirmadresse}
[6P5] 710: pop ds {Segment keyfarbe,farbe}
[TL1] 711: cmp al,' '
[PE6] 712: {Test auf hervorgehobenen Buchstaben}
[L42] 713: jne @nochoth
[9M6] 714: XOR al,al {00: Colorcode, farbe in AH}
[V74] 715: mov ah,farbe{Farbe laden}
[ZC6] 716: out dx,ax;inc di{nächstes Zeichen}
[Y22] 717: dec cl{Länge - 1}
[NK1] 718: @nochoth:
[GF4] 719: inc bx {nächste Spalte}
[T14] 720: dec cl {Länge String -1}
[KA2] 721: jnz @nextchar
[301] 722: @ende:
[165] 723: mov ax,$0001 {01: Verknüpfung}
[YH1] 724: out dx,ax
[C86] 725: XOR al,al {00: Colorcode, farbe in AH}
[YH1] 726: out dx,ax
[ZN0] 727: end;
[ZN0] 728: end;
[6M5] 729: procedure hline(x1,y1,x2:Integer);
[XD3] 730: var startmask:Byte;
[E74] 731: endmask:Byte;farbe:Byte;
[S53] 732: limits:viewporttype;
[MU0] 733: begin
[W57] 734: GetViewSettings(limits);farbe:=GetColor;
[FN0] 735: asm
[Z33] 736: mov ax,x2;cmp ax,x1
[SC5] 737: jae @xok {wenn x2=x1 dann xok}
[PP4] 738: mov dx,x1{sonst tauschen}
[T93] 739: mov x1,ax;mov x2,dx
[3W0] 740: @xok:
[FA4] 741: mov ax,y1{Zeile nach ax}
[X47] 742: add ax,limits.y1 {Viewport verschiebung}
[W17] 743: cmp ax,480 {Test Bereichsüberschreitung}
[294] 744: jae @ende{dann Abbruch}
[X23] 745: mov di,ax{und nach di}
[994] 746: mov cl,6 {6 mal schieben}
[SP3] 747: SHL di,cl{also mal 64}
[Z84] 748: mov cl,4 {4 mal schieben}
[YR3] 749: SHL ax,cl{also mal 16}
[XE5] 750: add di,ax{Summe ist Zeile * 80}
[3G4] 751: mov ax,x1{Spalte holen}
[RB7] 752: add ax,limits.x1 {Viewportverschiebung}
[P17] 753: cmp ax,640 {Test Bereichsüberschreitung}
[6F3] 754: jae @ende{dann ende}
[EW5] 755: mov cl,al{Lo-Byte X-Pos merken}
[YC5] 756: SHR ax,1 {x-drei mal rechts=/ 8}
[3T2] 757: SHR ax,1;SHR ax,1
[9N5] 758: add di,ax{di=Offset für Video}
[934] 759: AND cl,7 {x1 low Mod 8}
[154] 760: mov al,$FF {laden mit 255}
[G97] 761: SHR al,cl{cl mal nach rechts für start}
[7U2] 762: mov startmask,al
[1X5] 763: {Startmaske sichern erstes Byte}
[8E4] 764: mov cx,x2{x-ende holen}
[H67] 765: add cx,limits.x1 {Viewport-Verschiebung}
[6L2] 766: AND cl,7 {mod 8}
[C63] 767: XOR cl,7 {7- rest}
[154] 768: mov al,$FF {Laden mit 255}
[6E5] 769: SHL al,cl{cl mal links schieben}
[EUE] 770: mov endmask,al {Bitmaske letztes Byte}
[UR6] 771: mov dx,$3CE {Portadr für Video-Karte}
[ZU1] 772: mov ax,$0005
[U75] 773: {05: Modus, 00=Read/Write Mode 0}
[7C3] 774: out dx,ax;mov ax,$0003
[1L5] 775: {03: Function select, 0=ersetzen}
[X93] 776: out dx,ax;XOR al,al
[WF4] 777: {00: Colorcode, farbe in AH}
```

```
[FN5] 778: mov ah,farbe;out dx,ax;mov ax,$0F01
[U07] 779: {01: Verknüpfung, 0F=Farbe aus Colorcode}
[YH1] 780: out dx,ax
[GW4] 781: mov ax,$A000 {Video-Segment}
[3H1] 782: mov es,ax
[177] 783: {als aktuelles Extrasegment für stob}
[P94] 784: mov cx,x2;add cx,limits.x1
[S64] 785: SHR cx,1;SHR cx,1;SHR cx,1
[D83] 786: {x2 in Bytes, d.h /8}
[CN3] 787: mov ax,x1{x1 nach ax}
[PN2] 788: add ax,limits.x1
[L54] 789: SHR ax,1;SHR ax,1;SHR ax,1
[983] 790: {x1 in Bytes, d.h /8}
[DN3] 791: sub cx,ax{Länge +1}
[EP2] 792: dec cx {Länge}
[K65] 793: mov al,$08 {08: Bitmap-Register}
[NM7] 794: mov ah,startmask {Maske für erste Spalte}
[V77] 795: out dx,ax{ins Maskenregister übertragen}
[Y75] 796: mov ah,es:[di]{Latches laden}
[2H2] 797: mov es:[di],ah
[F36] 798: {Neue Registerwerte übertragen}
[XN5] 799: inc di {für Last Berechnung +1}
[DX6] 800: mov ah,$FF {Bitmaske für Blockinneres}
[SW5] 801: out dx,ax{ins Bitmask-Register}
[A06] 802: cmp cx,81{Breite >=80, dann 0 Breite}
[0F1] 803: jae @ende
[444] 804: cld{aufwärts zählen}
[S16] 805: rep stob{Breite * Bytes übertragen}
[U77] 806: mov ah,endmask {Bitmaske für Blockende}
[VN5] 807: out dx,ax{ins Bit-Mask-Register}
[E25] 808: mov ah,es:[di]{Latches lesen und}
[466] 809: mov es:[di],ah {verändert zurück}
[391] 810: @ende:
[165] 811: mov ax,$0001 {01: Verknüpfung}
[X93] 812: out dx,ax;XOR al,al
[WF4] 813: {00: Colorcode, farbe in AH}
[633] 814: mov ah,0;out dx,ax
[ZN0] 815: end;
[ZN0] 816: end;
[2P5] 817: procedure vline(x1,y1,y2:Integer);
[0D6] 818: var farbe:Byte;limits:viewporttype;
[MU0] 819: begin
[W57] 820: GetViewSettings(limits);farbe:=GetColor;
[FN0] 821: asm
[343] 822: mov ax,y2;cmp ax,y1
[V85] 823: jae @yok {Wenn y2>y1 dann yok}
[RP4] 824: mov dx,y1{sonst tauschen}
[Z93] 825: mov y1,ax;mov y2,dx
[6W0] 826: @yok:
[FA4] 827: mov ax,y1{Zeile nach ax}
[X47] 828: add ax,limits.y1 {Viewport verschiebung}
[W17] 829: cmp ax,480 {Test Bereichsüberschreitung}
[294] 830: jae @ende{dann Abbruch}
[BY3] 831: mov bx,ax{und nach di}
[994] 832: mov cl,6 {6 mal schieben}
[1S3] 833: SHL bx,cl{also mal 64}
[Z84] 834: mov cl,4 {4 mal schieben}
[YR3] 835: SHL ax,cl{also mal 16}
[6G5] 836: add bx,ax{Summe ist Zeile * 80}
[3G4] 837: mov ax,x1{Spalte holen}
[RB7] 838: add ax,limits.x1 {Viewportverschiebung}
[P17] 839: cmp ax,640 {Test Bereichsüberschreitung}
[6F3] 840: jae @ende{dann ende}
[EW5] 841: mov cl,al{Lo-Byte X-Pos merken}
[YC5] 842: SHR ax,1 {x-drei mal rechts=/ 8}
[ZE4] 843: SHR ax,1;SHR ax,1;add bx,ax
[S53] 844: {di=Offset für Video}
[6L2] 845: AND cl,7 {Mod 8}
[H55] 846: XOR cl,7 {7- x Mod 8, Bitnummer}
[1S3] 847: mov dx,$3CE {Portadr}
[S04] 848: mov al,$08 {Bitmask-Reg}
[3X5] 849: mov ah,1 {ein Bit zum Verschieben}
[XG4] 850: SHL ah,cl{Bit auswählen}
[R96] 851: out dx,ax{Portadr für Video-Karte}
[ZU1] 852: mov ax,$0005
[U75] 853: {05: Modus, 00=Read/Write Mode 0}
[YH1] 854: out dx,ax
[0U1] 855: mov ax,$0003
[1L5] 856: {03: Function select, 0=ersetzen}
[RC3] 857: out dx,ax;mov ax,$0F01
[U07] 858: {01: Verknüpfung, 0F=Farbe aus Colorcode}
[X93] 859: out dx,ax;XOR al,al
[WF4] 860: {00: Colorcode, farbe in AH}
[995] 861: mov ah,farbe;out dx,ax;mov ah,$A0
[Z23] 862: {Video-Segment}
[CS6] 863: mov es,ax{als aktuelles Extrasegment}
[UC6] 864: mov cx,y2;sub cx,y1{Höhe-1 ermitteln}
[GN3] 865: inc cx {Höhe nach cx}
[ED4] 866: mov dx,80{Zeilenabstand}
[KF1] 867: @punkte:
[695] 868: mov al,es:[bx]{Latches laden}
[927] 869: mov es:[bx],al;add bx,dx {Nächste Zeile}
[ZM6] 870: loop @punkte {Höhe aus cx-1 wenn 0: End}
[301] 871: @ende:
[6X1] 872: mov dx,$3CE
[165] 873: mov ax,$0001 {01: Verknüpfung}
[YH1] 874: out dx,ax
[406] 875: XOR al,al{00: Colorcode, farbe in AH}
```



```
[633] 876: mov ah,0;out dx,ax
[4M0] 877: end
[ZN0] 878: end;
[KZ6] 879: procedure getwindow(x1,y1,x2,y2:Integer);
[WE4] 880: var limits:viewporttype;
[MU0] 881: begin
[7C4] 882:   GetViewSettings(limits);
[FN0] 883:   asm
[MK4] 884:     mov ax,x2;cmp ax,x1;jae @xok
[5Y3] 885:   (wenn x2>=x1 dann xok)
[PP4] 886:     mov dx,x1(sonst tauschen)
[T93] 887:     mov x1,ax;mov x2,dx
[3W0] 888:   @xok:
[343] 889:     mov ax,y2;cmp ax,y1
[V85] 890:     jae @yok (Wenn y2>y1 dann yok)
[RP4] 891:     mov dx,y1(sonst tauschen)
[Z93] 892:     mov y1,ax;mov y2,dx
[6W0] 893:   @yok:
[ZU1] 894:     mov ax,$0000
[U35] 895:   (Video Speicher Anf (Quelle))
[W36] 896:     mov es,ax(Aktuelles Datenssegment)
[FA4] 897:     mov ax,y1(Zeile nach ax)
[VB7] 898:     add ax,limits.y1 (Viewportverschiebung)
[W17] 899:     cmp ax,480 (Test Bereichsüberschreitung)
[294] 900:     jae @ende(dann Abbruch)
[7Z3] 901:     mov bx,ax(und nach bx)
[994] 902:     mov cl,6 (6 mal schieben)
[1S3] 903:     SHL bx,cl(also mal 64)
[Z84] 904:     mov cl,4 (4 mal schieben)
[YR3] 905:     SHL ax,cl(also mal 16)
[6G5] 906:     add bx,ax(Summe ist Zeile * 80)
[3G4] 907:     mov ax,x1(Spalte holen)
[RB7] 908:     add ax,limits.x1 (Viewportverschiebung)
[P17] 909:     cmp ax,640 (Test Bereichsüberschreitung)
[6F3] 910:     jae @ende(dann ende)
[YC5] 911:     SHR ax,1 (x-drei mal rechts=/ 8)
[RH6] 912:     SHR ax,1;SHR ax,1;add bx,ax(bx=Offset)
[P94] 913:     mov cx,x2;add cx,limits.x1
[564] 914:     SHR cx,1;SHR cx,1;SHR cx,1
[D83] 915:   (x2 in Bytes, d.h /8)
[CN3] 916:     mov ax,x1(x1 nach ax)
[PN2] 917:     add ax,limits.x1
[L54] 918:     SHR ax,1;SHR ax,1;SHR ax,1
[983] 919:   (x1 in Bytes, d.h /8)
[HN3] 920:     sub cx,ax(Länge -1)
[FP2] 921:     Inc cl (Länge)
[ZC4] 922:     mov si,80(Zeilendistanz)
[SS5] 923:     sub si,cx(Summand für neue Zeile)
[NG4] 924:     mov dx,$3CE (Portadresse)
[ZU1] 925:     mov ax,$0005
[U75] 926:   (05: Modus, 00=Read/Write Mode 0)
[YH1] 927:     out dx,ax
[0U1] 928:     mov ax,$0003
[1L5] 929:   (03: Function select, 0=ersetzen)
[RC3] 930:     out dx,ax;mov ax,$0F01
[U07] 931:   (01: Verknüpfung, 0F=Farbe aus Colorcode)
[X93] 932:     out dx,ax;XOR al,al
[WF4] 933:   (00: Colorcode, farbe in AH)
[UW4] 934:     mov ah,$0F;out dx,ax;mov al,$08
[UZ2] 935:   (Bit Mask Select)
[X35] 936:     push bx(Ziel Offset sichern)
[LP3] 937:     mov bx,y2(Y2 Laden)
[L63] 938:     sub bx,y1(Höhe -1)
[2R3] 939:     Inc bx (Höhe in bx)
[7F5] 940:     mov di,winbuffer (Ziel-Offset)
[AB4] 941:     mov ax,cl(Breite in ah)
[LS5] 942:     out dx,ax(Als Bitmaske schreiben)
[RR2] 943:     mov ah,es:[38400]
[0M5] 944:     mov es:[di],ah (Breite sichern)
[1B5] 945:     Inc di (Nächstes Puffer-Byte)
[3A4] 946:     mov ah,bl(Low-Byte Höhe)
[YH4] 947:     out dx,ax;mov ah,es:[38400]
[KA6] 948:     mov es:[di],ah (Höhe (low) sichern)
[MC6] 949:     Inc di;mov ah,bh(Höhe (high) sichern)
[YH4] 950:     out dx,ax;mov ah,es:[38400]
[2H2] 951:     mov es:[di],ah
[Y16] 952:   (Höhe (hihg) übertragen sichern)
[6E5] 953:     Inc di (Zeiger auf Datenpuffer)
[AG4] 954:     mov ax,$0105 (Writemode 1)
[PW5] 955:     out dx,ax;mov dx,bx(Höhe in dx)
[EH3] 956:     pop bx (Bild-Offset)
[UP6] 957:     mov ch,cl(zwischenspeichern Breite)
[9B1] 958:   @bytes:
[695] 959:     mov al,es:[bx](Latches laden)
[DK6] 960:     mov es:[di],al (in Zieladr. schreiben)
[VF4] 961:     Inc bx (nächstes Ziel)
[F34] 962:     Inc di (nächstes Ziel)
[A53] 963:     Dec cl (Breite - 1)
[RT5] 964:     jnz @bytes (bis Zeile eingelesen)
[KY3] 965:     mov cl,al(alte Breite)
[MN3] 966:     add bx,si(neue Zeile)
[0U2] 967:     Dec dx (Höhe - 1)
[L57] 968:     jnz @bytes (bis alle Zeilen eingelesen)
[301] 969:   @ende:
[3T3] 970:     mov dx,$3CE;mov ax,$0005
[U75] 971:   (05: Modus, 00=Read/Write Mode 0)
[A43] 972:     out dx,ax;mov al,$08
[UZ2] 973:   (Bit Mask Select)
[FT2] 974:     mov bx,winbuffer
```

```
[A25] 975: (Startadr ans Ende anhängen)
[L84] 976:     mov ah,bl(Low-Byte adr)
[YH4] 977:     out dx,ax;mov ah,es:[38400]
[SP6] 978:     mov es:[di],ah (Low Byte übertragen)
[YX3] 979:     Inc di (Next Speicher)
[Z85] 980:     mov ah,bh(High-Byte wählen)
[YH4] 981:     out dx,ax;mov ah,es:[38400]
[DT6] 982:     mov es:[di],ah (High-Byte übertragen)
[B01] 983:     Inc di
[DN6] 984:   (Zeiger auf nächsten freien Bereich)
[3M6] 985:     mov winbuffer,di (Neue Pufferadresse)
[VK4] 986:     mov ah,$FF (Default-Werte)
[1C3] 987:     out dx,ax;mov ax,$0001
[U07] 988:   (01: Verknüpfung, 0F=Farbe aus Colorcode)
[TB3] 989:     out dx,ax;XOR ax,ax
[WF4] 990:   (00: Colorcode, farbe in AH)
[YH1] 991:     out dx,ax
[4M0] 992:   end
[ZN0] 993:   end;
[T26] 994: procedure putwindow(x1,y1:Integer);
[WE4] 995: var limits:viewporttype;
[MU0] 996: begin
[7C4] 997:   GetViewSettings(limits);
[FN0] 998:   asm
[SG6] 999:     mov ax,$0000 (Video Speicher (Quelle))
[W36] 1000:     mov es,ax(Aktuelles Datenssegment)
[FA4] 1001:     mov ax,y1(Zeile nach ax)
[VB7] 1002:     add ax,limits.y1 (Viewportverschiebung)
[W17] 1003:     cmp ax,480 (Test Bereichsüberschreitung)
[294] 1004:     jae @ende(dann Abbruch)
[7Z3] 1005:     mov bx,ax(und nach bx)
[994] 1006:     mov cl,6 (6 mal schieben)
[1S3] 1007:     SHL bx,cl(also mal 64)
[Z84] 1008:     mov cl,4 (4 mal schieben)
[YR3] 1009:     SHL ax,cl(also mal 16)
[6G5] 1010:     add bx,ax(Summe ist Zeile * 80)
[3G4] 1011:     mov ax,x1(Spalte holen)
[RB7] 1012:     add ax,limits.x1 (Viewportverschiebung)
[P17] 1013:     cmp ax,640 (Test Bereichsüberschreitung)
[6F3] 1014:     jae @ende(dann ende)
[YC5] 1015:     SHR ax,1 (x-drei mal rechts=/ 8)
[RH6] 1016:     SHR ax,1;SHR ax,1;add bx,ax(bx=Offset)
[1S3] 1017:     mov dx,$3CE (Portadr)
[K24] 1018:     mov ax,$0005 (Readmode 0)
[YH1] 1019:     out dx,ax
[BZ6] 1020:   (Seite 3 hier stehen die Informationen)
[9D3] 1021:     mov ax,$004 (Plane 0)
[7G4] 1022:     out dx,ax;mov di,winbuffer
[315] 1023:   (Zeiger auf freien Bereich)
[FS1] 1024:     cmp di,38400
[DB5] 1025:   (kleiner gleich 38400 (Bildende))
[3F1] 1026:     jbe @ende
[7Y6] 1027:   (dann Abbruch, kein Fenster gespeichert)
[WK5] 1028:     Dec di (Zeiger auf Startadresse)
[LD3] 1029:     Dec di;mov ax,es:[di]
[FR3] 1030:   (Startadresse holen)
[B46] 1031:     mov di,ax(Startoffset Daten in di)
[NH6] 1032:     mov winbuffer,ax (Speicher freigeben)
[B45] 1033:     mov cl,es:[di](Breite laden)
[ZC4] 1034:     mov si,80(Zeilendistanz)
[SS5] 1035:     sub si,cx(Summand für neue Zeile)
[UP6] 1036:     mov ch,cl(zwischenspeichern Breite)
[M94] 1037:     Inc di (Breite überlesen)
[SH4] 1038:     push bx(xy start sichern)
[Z35] 1039:     mov bx,es:[di](Höhe Laden)
[1S3] 1040:     mov dx,$3CE (Portadr)
[5U1] 1041:     mov ax,$0105
[S85] 1042:   (writemode 1 d.h nur kopieren)
[H95] 1043:     out dx,ax;mov dx,bx(dx: Höhe)
[7M3] 1044:     pop bx (Start Ziel)
[SL4] 1045:     Inc di (Höhe überlesen)
[B01] 1046:     Inc di
[9B1] 1047:   @bytes:
[A85] 1048:     mov al,es:[di](Latches laden)
[RK2] 1049:     mov es:[bx],al
[A55] 1050:   (in Zieladresse schreiben)
[CK3] 1051:     Inc bx (neues Ziel)
[3F3] 1052:     Inc di (neue Quelle)
[TT1] 1053:     cmp bx,38401
[U06] 1054:   (Sicherheit: 38400 nicht überschreiten!)
[EY5] 1055:     jae @abbruch (dann sofort Abbruch)
[WX2] 1056:     Dec cl (Breite -1)
[5D5] 1057:     jnz @bytes (Bis Zeile kopiert)
[SF1] 1058:   @abbruch:
[KY3] 1059:     mov cl,ah(alte Breite)
[MN3] 1060:     add bx,si(neue Zeile)
[PP2] 1061:     Dec dx (Höhe -1)
[BF6] 1062:     jnz @bytes (bis alle Zeilen kopiert)
[301] 1063:   @ende:
[1S3] 1064:     mov dx,$3CE (Portadr)
[5G4] 1065:     mov ax,$0005 (Writemode 0)
[ZX4] 1066:     out dx,ax;mov ax,$0004;out dx,ax
[4M0] 1067:   end
[ZN0] 1068:   end;
[KN0] 1069:   end.
```

Listing 13. »fastvga.pas« gibt Ihnen eine Unit als Ersatz für Ihre BGI-Treiber, die mit Assembler-Tuning erheblichen Geschwindigkeitsgewinn bringt.


```
[T74] 1: {Programm: fastdemo.pas
[S55] 2: Funktion: für Unit fastvga.tpu
[HV5] 3: Sprache: Turbo Pascal ab Version 6.0
[R93] 4: Autor: Klaus Eberle
[XX4] 5: (c)1992 DMV Widuch GmbH & Co. KG
[UP4] 6: uses Dos, fastvga, Graph, Crt;
[YS2] 7: procedure winr;
[MU0] 8: begin
[WY4] 9:   SetViewport(5,18,316,475,false);
[ZN0] 10: end;
[CS2] 11: procedure winl;
[MU0] 12: begin
[685] 13:   SetViewport(325,18,635,475,false);
[ZN0] 14: end;
[U73] 15: procedure waitkey;
[3Y1] 16: var c:Char;
[Y32] 17: begin repeat;
[ZY4] 18:   until KeyPressed;c:=ReadKey;
[ZN0] 19: end;
[N73] 20: procedure runtext;
[6H3] 21: var I:Integer;c:Char;
[AW1] 22: begin winr;
[LN2] 23:   repeat Sound(60);
[XE3] 24:     directvideo:=false;
[RM3] 25:     for I:=1 to 27 do begin
[DB2] 26:       MoveTo(5,I*16);
[A44] 27:       SetColor(lightmagenta);
[ST2] 28:       OutText('***** ');
[TD2] 29:       SetColor(red);
[XR2] 30:       OutText('Text');
[A44] 31:       SetColor(lightmagenta);
[3H1] 32:       OutText
[2H4] 33:         ('ausgabe mit Outtext *****');
[NS4] 34:       end;NoSound;ClearViewport;
[ZY4] 35:       until KeyPressed;c:=ReadKey;
[3W0] 36:       winl;
[1W5] 37:       SetFillStyle(solidfill,darkgray);
[LN2] 38:       repeat Sound(60);
[B15] 39:       for I:=2 to 28 do begin Delay(7);
[AM2] 40:         fasttextxy(2,I,
[SE6] 41:         '***** (Text)ausgabe mit FASTTEXT *****');
[ZN0] 42:       end;
[US4] 43:       NoSound;solidbar(2,10,308,450);
[ZY4] 44:       until KeyPressed;c:=ReadKey;
[R25] 45:       SetColor(white);MoveTo(100,200);
[WN4] 46:       OutText('Nicht schlecht ?');
[NS3] 47:       Delay(500);MoveTo(10,260);
[3H1] 48:       OutText
[DA6] 49:       ('Gut, dann jetzt ohne Warteschleife !');
[A44] 50:       SetColor(lightmagenta);
[DF1] 51:       waitkey;
[P31] 52:       repeat
[OK1] 53:         Sound(60);
[T65] 54:         for I:=2 to 28 do fasttextxy(2,I,
[SE6] 55:         '***** (Text)ausgabe mit FASTTEXT *****');
[OT4] 56:         NoSound;solidbar(2,13,308,450);
[ZY4] 57:         until KeyPressed;c:=ReadKey;
[ZN0] 58:       end;
[W73] 59: procedure runplot;
[UW2] 60: var x,y:Integer;
[C71] 61: c:Char;
[MU0] 62: begin
[5Y2] 63:   winr;MoveTo(10,4);
[3H5] 64:   OutText('PutPixel (x,y,farbe) :');
[PU3] 65:   for y:=20 to 420 do begin
[8S1] 66:     Sound(5000);
[TF4] 67:     for x:=5 to 300 do PutPixel
[5N4] 68:       (x,y,lightmagenta);NoSound;
[ZN0] 69:   end;
[H25] 70:   waitkey;ClearViewport;winl;
[XY1] 71:   MoveTo(10,4);
[AR4] 72:   OutText('Plot (x,y,farbe) :');
[PU3] 73:   for y:=20 to 420 do begin
[8S1] 74:     Sound(5000);
[GR3] 75:     for x:=5 to 300 do plot
[V63] 76:       (x,y,lightmagenta);
[262] 77:     NoSound;end;
[Q24] 78:   waitkey;ClearViewport;
[ZN0] 79: end;
[563] 80: procedure runline;
[644] 81: var x,y:Integer;c:Char;
[MU0] 82: begin
[5Y2] 83:   winr;MoveTo(10,4);
[OH5] 84:   OutText('Horizontale Linien :');
[253] 85:   SetColor(magenta);
[PN2] 86:   repeat Sound(70);
[622] 87:     for y:=20 to 430 do
[WL4] 88:       Line(5,y,300,y);Bar(5,20,300,430);
[3G1] 89:     NoSound;
[GF4] 90:   until KeyPressed;waitkey;
[PC5] 91:   ClearViewport;winl;MoveTo(10,4);
[7M4] 92:   OutText('HLINE (x1,y2,x2) :');
[PN2] 93:   repeat Sound(70);
[775] 94:     for y:=20 to 430 do hline(5,y,300);
[TP4] 95:     solidbar(5,20,300,430);NoSound;
[GF4] 96:   until KeyPressed;waitkey;
```

```
[LK2] 97:   ClearViewport;
[ZN0] 98: end;
[113] 99: procedure runwin;
[Z55] 100: var feld:array[1..30000]of Byte;
[WR3] 101: I:Integer;add:Integer;
[MU0] 102: begin
[F76] 103:   SetFillStyle(solidfill,lightgreen);
[6K4] 104:   winr;SetColor(lightgreen);
[XY1] 105:   MoveTo(10,4);
[A85] 106:   OutText('Getimage/Putimage :');
[A44] 107:   SetColor(lightmagenta);
[RM3] 108:   for I:=1 to 27 do begin
[DB2] 109:     MoveTo(5,I*16);
[3H1] 110:     OutText
[636] 111:     ('***** Textausgabe mit OUTTEXT *****');
[8L5] 112:     end;I:=10;add:=8;repeat Sound(400);
[SE4] 113:       GetImage(I,100,140+I,300,feld);
[B94] 114:       Bar(I,100,140+I,300);NoSound;
[U55] 115:       PutImage(I,100,feld,normalput);
[OB1] 116:       I:=I+add;
[X43] 117:       if I>150 then add:=8;
[RW2] 118:       if I<10 then add:=8;
[GF4] 119:       until KeyPressed;waitkey;
[PC5] 120:       ClearViewport;winl;MoveTo(10,4);
[1R3] 121:       SetColor(lightgreen);
[TR5] 122:       OutText('Getwindow/Putwindow :');
[A44] 123:       SetColor(lightmagenta);
[T65] 124:       for I:=2 to 28 do fasttextxy(2,I,
[U36] 125:       '***** Textausgabe mit FASTTEXT *****');
[9X4] 126:       I:=10;add:=8;repeat Sound(400);
[NZ3] 127:         getwindow(I,100,140+I,300);
[YK3] 128:         solidbar(I,100,140+I,300);
[B74] 129:         NoSound;putwindow(I,100);
[OB1] 130:         I:=I+add;
[X43] 131:         if I>150 then add:=8;
[RW2] 132:         if I<10 then add:=8;
[H77] 133:         until KeyPressed;waitkey;ClearViewport;
[ZN0] 134:       end;
[R53] 135: procedure runicon;
[BY4] 136: var x,y:Byte;picon:pointer;
[MU0] 137: begin
[YS2] 138:   GetMem(picon,100);
[ZH4] 139:   readicon('icon.txt',picon);
[564] 140:   SetColor(magenta);winr;
[EM2] 141:   fasttextxy(2,14,
[OP5] 142:   ' --- Schnelle Ausgabe von Icons -->');
[AS3] 143:   winl; repeat Sound(200);
[YK2] 144:     for y:=1 to 14 do
[6G3] 145:       for x:=1 to 7 do begin
[F22] 146:         SetColor(x);
[NB4] 147:         fasticonxy(x*5,y*2,picon);
[ZN0] 148:       end;
[RR4] 149:       solidbar(4,15,308,453);NoSound;
[GF4] 150:       until KeyPressed;waitkey;
[ZN0] 151:     end;
[453] 152: procedure daswars;
[MU0] 153: begin
[TZ4] 154:   SetViewport(0,0,639,479,false);
[765] 155:   Bar(190,210,440,250);MoveTo(195,217);
[3H1] 156:   OutText
[NC5] 157:   ('FASTVGA 2.0 - von Klaus Eberle');
[GA2] 158:   MoveTo(235,233);
[1F5] 159:   OutText('Ende der FASTVGA-Demo');
[DF1] 160:   waitkey;
[ZN0] 161: end;
[166] 162: var mode,treiber:Integer;(Grafik)
[XV6] 163: pchars:pointer;(Zeiger auf Zeichensatz)
[MU0] 164: begin
[7C5] 165:   TextMode(co80);treiber:=detect;
[PD4] 166:   InitGraph(treiber,mode,'');
[AH4] 167:   SetGraphMode(mode);setvga;
[X43] 168:   GetMem(pchars,255*8);
[4D3] 169:   setcharmem(pchars);
[H95] 170:   initfast;SetBkColor(darkgray);
[D16] 171:   SetPalette(2,black);SetColor(white);
[955] 172:   sethotcolor(red);clearscreen;
[CT2] 173:   SetColor(white);
[Y35] 174:   SetFillStyle(blue,solidfill);
[ZR4] 175:   Line(3,3,318,3);Line(318,3,318,476);
[325] 176:   Line(323,3,636,3);Line(636,3,636,476);
[X24] 177:   SetColor(2);Line(3,3,476);
[585] 178:   Line(3,476,318,476);Line(323,3,323,476);
[N25] 179:   Line(323,476,636,476);MoveTo(88,7);
[CT2] 180:   SetColor(white);
[HL4] 181:   OutText('Speedtest - BGI');
[Z32] 182:   MoveTo(408,7);
[RM6] 183:   showtext('Speedtest - (FASTVGA 2.0)');
[RC4] 184:   SetColor(13);MoveTo(88,17);
[FB4] 185:   LineTo(207,17);MoveTo(408,17);
[C75] 186:   LineTo(592,17);runtext;runicon;
[CX5] 187:   runplot;runline;runwin;daswars;
[522] 188:   closeGraph;
[KN0] 189: end.
```

Listing 14. »fastdemo.pas« führt Ihnen die beeindruckende Arbeit von »fastvga.tpu« vor.

Einfache Hintergrundgrafik

»grafiken.pas« (Listing 15) kompilieren Sie in Turbo Pascal ab Version 5.0. Grafiken zeichnet auf sehr einfache Weise verblüffende Hintergrundbilder.

Um in ein Programm eine Hintergrundgrafik einzubinden, benötigen Sie normalerweise ein Zeichen- oder ein Fraktalprogramm mit sehr langen Rechenzeiten. Grafiken zeichnet jedoch fünf verblüffende Hintergrundgrafiken innerhalb kurzer Zeit. Das System ist einfach: Mit je einer X- und Y-Schleife wird jeder Pixel genau einmal gezeichnet. Die Farbe wird in allen Grafiken ähnlich errechnet, durch einen Term, dessen Wert von »X« und »Y« abhängt. Der Wert ist jedoch fast immer höher als die Anzahl der Farben, also größer als 15. Er wird deshalb von Turbo Pascal abgeschnitten, so daß immer eine Farbe von 0 (schwarz) bis 15 (weiß) entsteht. Daraus ergeben sich dann bunte Pixel nebeneinander, die sich zu einem interessanten Muster ergänzen – Kachelmuster, Teppich oder einfach Bilder in der Art eines Regenbogens (Bild 9).

Der Benutzer kann zu den vorhandenen Bildern auch noch selbst einige hinzufügen. Die Terme sollten jedoch keine komplizierten mathematischen Funktionen enthalten, wichtig ist nur, daß die Zahlenwerte groß und deshalb von Turbo Pascal abgeschnitten werden, so daß wieder eine Zahl zwischen 0 und 15 herauskommt. Deshalb bringen Modulo-Anweisungen (wie »x mod 60«) auch schöne Ergebnisse. Sogar der einfachste Term, »x*y« zeichnet ein Muster über den Bildschirm. Im Quelltext sollten Sie bei der InitGraph-Anweisung (Zeile 10) eventuell das Verzeichnis, in dem sich die BGI-Dateien befinden, angeben. Grafiken eignet sich nur für einen Farbbildschirm, da auf einem Schwarz/Weiß-Bildschirm die Pixelmuster nur als graue Fläche erscheinen.

(Jan Leicher/et)

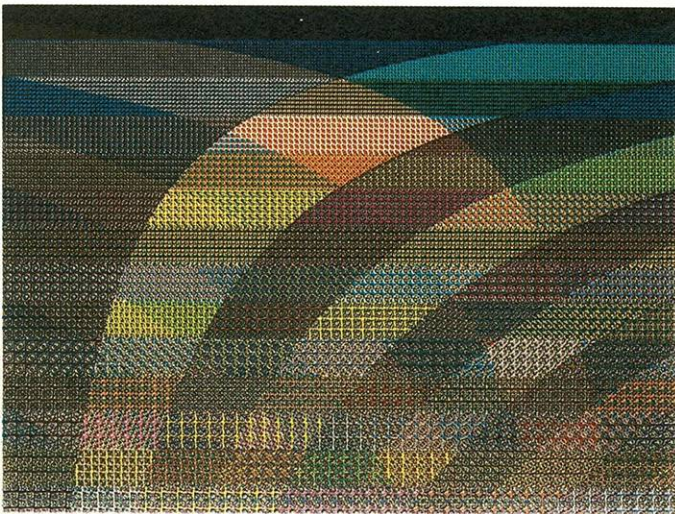


Bild 9. »grafiken.pas« zeigt mit jeder Eingabe-Taste ein neues Bild.

```
[354] 1: {Programm: grafiken.pas
[WK6] 2: Funktion: einfache Hintergrundgrafiken
[EV5] 3: Sprache: Turbo Pascal ab Version 5.0
[DX2] 4: Autor: Jan Leicher
[XX4] 5: (c)1992 DMV Widuch GmbH & Co. KG
[C02] 6: uses Graph;
[YZ4] 7: var d,m,i,j,l: Integer;k: Byte;
[MU0] 8: begin
[UF1] 9:   d:=9;m:=2;
[MR2] 10:  InitGraph(d,m,'');
[UY4] 11: {evtl. Verzeichnis angeben}
[VU2] 12:  for I:=0 to 639 do
[SV2] 13:  for j:=0 to 479 do
[CW4] 14:    PutPixel(I,j,Sqr(I)*1 DIV(j+1));
[991] 15:  ReadLn;
```

```
[VU2] 16:  for I:=0 to 639 do
[SV2] 17:  for j:=0 to 479 do
[TH1] 18:    PutPixel
[844] 19:    (I,j,3+(I MOD 60*j MOD 60*I*j));
[991] 20:  ReadLn;
[VU2] 21:  for I:=0 to 639 do
[1U2] 22:  for j:=0 to 514 do
[TH1] 23:    PutPixel
[204] 24:    (I,j-35,(I*j)MOD(j DIV 35+1));
[991] 25:  ReadLn;
[FW0] 26:  l:=48;
[MU2] 27:  for I:=140 to 779 do
[SV2] 28:  for j:=0 to 479 do
[MU0] 29:  begin
[0T4] 30:    if(j=0)AND(I MOD 3=0)then Inc(l);
[VU2] 31:    if j>=249 DIV 2 then
[MU0] 32:    begin
[162] 33:      k:=(I*j)MOD l;
[L23] 34:      PutPixel(I-140,j,k);
[CK3] 35:      PutPixel(I-140,j-249,k);
[ZN0] 36:    end;
[ZN0] 37:  end;
[991] 38:  ReadLn;
[CZ2] 39:  for I:=50 to 689 do
[LY2] 40:  for j:=50 to 529 do
[RD5] 41:    PutPixel(I-50,j-50,(I*j)DIV(I*I+1));
[991] 42:  ReadLn;
[522] 43:  CloseGraph;
[KN0] 44:  end.
```

Listing 15. »grafiken.pas« baut mit jeder Eingabe eine neue, einfache Hintergrundgrafik auf.

Fraktal zum Zoomen

»fraktal.cpp« (Listing 16) kompilieren Sie mit dem C++ Compiler 1.0 von Borland oder einem kompatiblen Produkt. Das Programm verwendet Gleitkommazahlen vom Typ »double«. Das bedeutet, Sie müssen den Compiler entweder Code für den 8087/80287 oder Emulationscode erzeugen lassen.

Der Originalbeitrag ist eine Einsendung zum 1024-Wettbewerb. Dessen Größenbegrenzung ist der Grund dafür, daß alle Variablen maximal 1 Zeichen lang sind und daß die Übersichtlichkeit verlorengeht, das gilt insbesondere für die Einrückungen und Leerzeichen.

Fraktal zeichnet zuerst die Mandelbrotmenge in ihrem bekanntesten Ausschnitt:

(A = (-2.0,1.25), B = (0.5,-1.25))

Danach sehen Sie ein Quadrat am Bildschirm, das Sie mit Hilfe der Cursor-Tasten bewegen und mit »+« oder »-« vergrößern oder verkleinern können. Sobald Sie die Eingabe betätigen, wird der so gewählte Ausschnitt in den gesamten verfügbaren Bereich vergrößert. So können Sie immer weiterzoomen, um weitere Bilder (Bild 10) zu »malen«. Die Genauigkeit der Farben an den Rändern der Mandel-

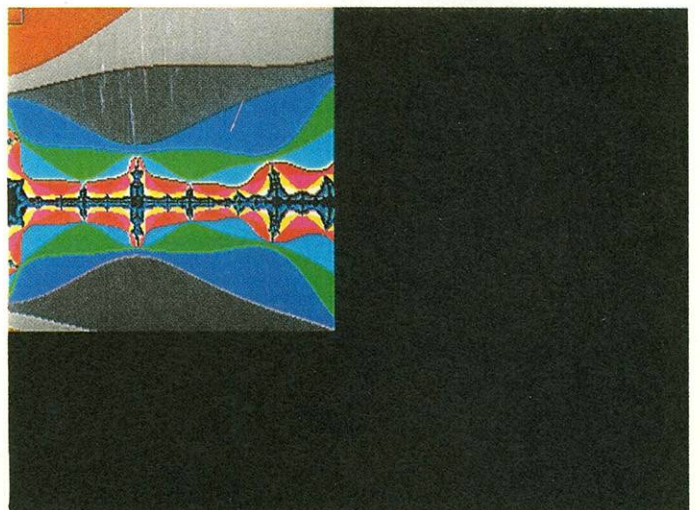


Bild 10. »fraktal.cpp« bietet ein Fraktal zum Zoomen.

brotmenge nimmt mit steigender maximaler Iterationstiefe »0« zu. Wählen Sie »0« zu klein, erfolgt die Berechnung zwar erheblich schneller, aber Sie sind schon nach ein paar »Zooms« am Ende der Genauigkeit und erhalten so nur noch große, gleichfarbige Flächen, was natürlich recht uninteressant ist.

Durch Veränderung der Farbzuoordnung bringen Sie Ihre Bilder noch besser zur Geltung; je nach Tiefe »t« können Sie andere Farbe setzen. Ein reines Graustufenbild wirkt beispielsweise sogar manchmal besser als ein buntes.

Der Bildschirmausschnitt wird durch »x« und »y« festgelegt. Sie können ihn verkleinern, was die Berechnung beschleunigt, oder vergrößern, was die Auflösung verbessert. Sie sollten jedoch beachten, daß der Ausschnitt immer quadratisch ist, damit sich das Bild beim Zoomen nicht verzerrt. Das Zoom-Fenster ist nämlich ein Quadrat.

Fraktal können Sie selbst sehr nützlich erweitern, indem Sie zum Beispiel eine Speicherroutine schreiben, damit Sie das Ergebnis von langen Rechenzeiten rasch wieder laden und anschauen können. Oder Sie können schnellere Berechnungsroutinen in Assembler schreiben, wobei C selbst aber auch schon flott ist. Ihrer Fantasie zur Ausgestaltung sind also kaum Grenzen gesetzt.

(Holger Schwedes/et)

```
[MX3] 1: /* Programm: fraktal.cpp
[AR4] 2: Funktion: Fraktal:  $z = z^2 + c$ 
[HP3] 3: Sprache: Borland C++ 1.0
[OT3] 4: Autor: Holger Schwedes
[EZ4] 5: (c)1992 DMV Widuch GmbH & Co.KG */
[EB3] 6: #include <complex.h>
[OE3] 7: #include <graphics.h>
[Z03] 8: #include <conio.h>
[HK4] 9: complex p1=complex(-2.0,1.25),
[404] 10: /* Punkt links oben und */
[R33] 11: p2=complex(0.5,-1.25),
[WK6] 12: /* Punkt rechts unten (Standardansicht) */
[490] 13: p,
[PC5] 14: /* Punkt der zu berechnen ist (c) */
[7B4] 15: z; /* für Formel  $z=z^2+c$  */
[WR3] 16: double rspalt,ispalt;
[F96] 17: /* Pixelabstand in x (real) und y (imag)*/
[TH2] 18: int gd=9,gm=2,
[UN6] 19: /* Variablen für Graphdriver und Mode */
[NR2] 20: maxx=200,maxy=200,
[N46] 21: /* Größe des Fensters in Pixeln */
[BZ0] 22: tiefe,
[MD5] 23: /* Farbe für berechneten Punkt */
[BY1] 24: maxtiefe=20,
[P55] 25: /* Maximale Iterationstiefe */
[AT1] 26: x1,y1,x2,y2;
[DG5] 27: /* Koordinaten des Zoomfensters */
[NV1] 28: char tast;
[LS5] 29: /* enthält Zeichen bei Tastendruck */
[Z00] 30:
[TG2] 31: void main (void)
[P70] 32: {
[LD3] 33:     initgraph (&gd,&gm,'');
[GL0] 34:     do {
[375] 35:         rspalt=(real(p2)-real(p1))/maxx;
[H65] 36:         ispalt=(imag(p2)-imag(p1))/maxy;
[684] 37:         p=complex(real(p),imag(p1));
[GL0] 38:         do {
[D94] 39:             p=complex(real(p1),imag(p));
[GL0] 40:             do {
[8N3] 41:                 z=complex(0,0);tiefe=0;
[GL0] 42:                 do {
[DD1] 43:                     z=z*z+p;
[F61] 44:                     tiefe++;
[8C1] 45:                 } while
[UV4] 46:                 ((abs(z)<2.0)&&(tiefe<maxtiefe));
[1P4] 47:                 if (tiefe==maxtiefe) tiefe=0;
[TH1] 48:                 putpixel
[VH3] 49:                 ((int)((real(p)-real(p1))
[7H1] 50:                 /rspalt),
[4E3] 51:                 (int)((imag(p)-imag(p1))
[BN2] 52:                 /ispalt),tiefe);
[BH5] 53:                 p=complex(real(p)+rspalt,imag(p));
[UD4] 54:             } while (real(p)<real(p2));
[YB5] 55:             p=complex(real(p),imag(p)+ispalt);
[MD4] 56:         } while (imag(p)>imag(p2));
[K83] 57:         x1=y1=0;x2=10;y2=10;
[7T2] 58:         setwritemode (1);
[GL0] 59:     } do {
```

```
[ZL3] 60:         rectangle (x1,y1,x2,y2);
[Z22] 61:         tast=getch();
[024] 62:         if (tast==0) tast=getch();
[ZL3] 63:         rectangle (x1,y1,x2,y2);
[RB2] 64:         if (tast==77) if
[VH3] 65:             (x2<maxx) {x1++;x2++;}
[MB2] 66:         if (tast==80) if
[ZK3] 67:             (y2<maxy) {y1++;y2++;}
[KB2] 68:         if (tast==72) if
[EY2] 69:             (y1>0) {y1--;y2--;}
[NB2] 70:         if (tast==75) if
[6Y2] 71:             (x1>0) {x1--;x2--;}
[7B2] 72:         if (tast==43) if
[7E5] 73:             ((x2<maxx)&&(y2<maxy)) {x2++;y2++;}
[9B2] 74:         if (tast==45) if
[Z25] 75:             ((x2>x1+1)&&(y2>y1+1)) {x2--;y2--;}
[DT4] 76:         } while ((tast!=13)&&(tast!=27));
[VT0] 77:         p=p1;
[DS1] 78:         pl=complex
[D06] 79:         (real(p)+x1*rspalt,imag(p)+y1*ispalt);
[GS1] 80:         p2=complex
[L06] 81:         (real(p)+x2*rspalt,imag(p)+y2*ispalt);
[H23] 82:         } while (tast!=27);
[U70] 83: }
```

Listing 16. »fraktal.cpp« können Sie sogar im Zoom-Fenster neu berechnen lassen.

Die Dateizerteil-Maschine

»cut.pas« (Listing 17) kompilieren Sie mit Turbo Pascal ab Version 6.0. »cut.pas« zerlegt eine bestehende Datei in mehrere Seiten (Einzeldateien) mit der als Parameter angegebenen Seitengröße. Speichern Sie beispielsweise während einer Btx-Sitzung mehrere Btx-Seiten in einer Datei, so läßt sich diese Datei zu einem späteren Zeitpunkt mit Cut wieder in die einzelnen Bildschirmseiten zerlegen. Diese können Sie somit nachträglich bearbeiten, womit es Ihnen leichter fallen dürfte, beispielsweise die neuesten Börsen-Online-Kurse in Ihre Kalkulationsprogramme zu übernehmen. Sie rufen Cut nach der Syntax auf:

cut [Laufwerk:][Pfad]Dateiname Seitengröße [Anzahl]

Die Bedeutung der einzelnen Parameter in diesem Aufruf entnehmen Sie Tabelle 5. Fehlt der Parameter »Anzahl«, verursachen Sie ein Zerlegen der gesamten Datei. Die Seiten der zerlegten Datei erhalten den Namen der Ausgangsdatei, jedoch mit hochzählender Erweiterung von »001« bis maximal »999«. Vor der Dateizerkleinerung bleibt empfehlenswert, den Faktor zu bestimmen, durch den Sie teilen können, damit den Datenträger nicht sinnlos viele, zu kleine Dateien bevölkern.

(Dipl.Ing. Volker Redlof/et)

Bezeichnung	Erklärung
Dateiname	Vollständiger Name der zu zerlegenden Datei
Seitengröße	Größe der einzelnen Seiten in Byte
Anzahl	Anzahl der Seiten vom Dateianfang

Tabelle 5. »cut.pas« rufen Sie mit diesen Parametern auf.

```
[643] 1: {Programm: cut.pas
[LU4] 2: Funktion: Datei in 1000 Teilen
[EV5] 3: Sprache: Turbo Pascal ab Version 5.0
[LV5] 4: Autor: Dipl.-Ing.(FH) Volker Redlof
[XX4] 5: (c)1992 DMV Widuch GmbH & Co. KG
[S62] 6: uses Crt,Dos;
[Y43] 7: const FileMax=51200;
[WY4] 8: type AdressParameter=record
[7Z4] 9:     Segment,Offset: Integer;end;
[RN0] 10: var
[0S3] 11: UserFile: file of Byte;
[C95] 12: Datei: array[1..FileMax]of Byte;
[N77] 13: S,DateiCheck: string;Par2,Par3,Par4:string;
[ZC6] 14: I,Code: Integer;SeitGroesse: Integer;
[5N6] 15: DatGroesse: Longint;SeitZahl: Integer;
```



```
[DR5] 16: Anzahl: Integer; RealTeiler: Real;
[W16] 17: Fehler, Handle: Byte; Cpu: Registers;
[T94] 18: PfadAdr: AdressParameter;
[ZP3] 19: Dateiname: string[59];
[R3] 20: Pfad_Name: string[60];
[824] 21: procedure Beschreibung;
[MU0] 22: begin
[6B6] 23: Write('Zerlegt eine Datei in Seiten ');
[045] 24: WriteLn('gleicher Größe. ');
[SU6] 25: Write('CUT [Laufwerk:] [Pfad] Dateiname ');
[ZV7] 26: WriteLn('Seitengröße [Anzahl.]; WriteLn;
[NC6] 27: Write('Dateiname : Vollständiger ');
[CU5] 28: Write('Name der Datei die zerlegt ');
[H04] 29: WriteLn('werden soll. ');
[FL7] 30: Write('Seitengröße: Größe der ein ');
[CD4] 31: WriteLn('zeilen Seiten. ');
[835] 32: Write('Anzahl : Anzahl der ');
[C36] 33: WriteLn('Seiten von Dateianfang. ');
[905] 34: Write(' (Fehlt hier die ');
[7M5] 35: Write('Angabe, so wird die gesamte ');
[ZH6] 36: WriteLn('Datei zerlegt. '); WriteLn; end;
[A87] 37: procedure Fehleingabe(FehlerCode: Integer);
[MU0] 38: begin
[ZL1] 39: Fehler:=1;
[823] 40: case FehlerCode of
[SD6] 41: 1: WriteLn('Falsche Parameterangaben!');
[551] 42: 2: begin
[HB6] 43: Write('Die Größe der angegebenen ');
[M04] 44: Write('Datei beträgt: ');
[L15] 45: WriteLn(DatGroesse, ' Bytes. ');
[L25] 46: Write('Datei lässt sich nicht mit ');
[LU5] 47: Write('angegebener Seitengröße ');
[3B4] 48: WriteLn('zerlegen !!'); end;
[751] 49: 3: begin
[YW4] 50: Write('Angegebene Anzahl der ');
[PT5] 51: Write('Seiten größer als Datei ');
[LL6] 52: WriteLn('größe/Seitengröße !!'); end;
[K06] 53: 4: begin Write('File-Sharing nicht ');
[EV4] 54: WriteLn('installiert !!'); end;
[KU5] 55: 5: WriteLn('Datei nicht gefunden !!');
[XR5] 56: 6: begin Write('Pfad oder Teil des ');
[AT5] 57: WriteLn('Pfades nicht gefunden !!');
[ZN0] 58: end;
[0Z5] 59: 7: begin Write('Zu viele geöffnete ');
[T04] 60: WriteLn('Dateien !!'); end;
[X16] 61: 8: begin Write('Zugriff verweigert ');
[ZW5] 62: Write('wie Diskette ist schreib ');
[X34] 63: WriteLn('geschützt !!'); end;
[WM5] 64: 9: WriteLn('Ungültiger Zugriff !!');
[KK6] 65: 10: WriteLn('Ungültiger Handle !!'); end;
[SG5] 66: Sound(1000); Delay(500); NoSound; end;
[PB7] 67: procedure DatSchreiben(Seitzahl: Integer);
[V86] 68: var I, K, Position: Integer; S: string;
[V03] 69: begin Position:=0;
[895] 70: WriteLn('Zerlege die Datei .... ');
[WA6] 71: for I:=1 to Seitzahl do begin Str(I, S);
[C33] 72: if I<10 then S:='0'+S;
[S34] 73: else if I<100 then S:='0'+S;
[3F5] 74: Assign(UserFile, DateiCheck+'.'+S);
[PN4] 75: ({I-}) Reset(UserFile); ({I+})
[CD6] 76: if IOResult<>0 then Rewrite(UserFile);
[PD5] 77: for K:=1 to Seitzahl do begin
[XP3] 78: Position:=Position+1;
[4N5] 79: Write(UserFile, Datei[Position]);
[6S4] 80: end; Close(UserFile); end; end;
[ZC3] 81: procedure DatLesen;
[ZH2] 82: var I: Integer;
[386] 83: begin I:=1; Assign(UserFile, Dateiname);
[PN4] 84: ({I-}) Reset(UserFile); ({I+})
[ZX3] 85: if IOResult=0 then begin
[4Z5] 86: WriteLn('Lese die Datei ', Dateiname);
[T45] 87: while NOT EOF(UserFile) do begin
[Y24] 88: Read(UserFile, Datei[I]);
[T45] 89: I:=I+1; end; Close(UserFile); end
[BV3] 90: else Fehleingabe(3); end;
[LA7] 91: procedure OeffneDat(pfad: AdressParameter;
[LU1] 92: Code: Byte);
[8U4] 93: begin Cpu.ah:=$3D; Cpu.al:=Code;
[T07] 94: Cpu.ds:=pfad.Segment; Cpu.dx:=pfad.Offset;
[N74] 95: MsDos(Cpu); Handle:=Cpu.ax;
[0E5] 96: if (Cpu.flags AND 1)=0 then Fehler:=0
[R34] 97: else Fehler:=Cpu.al; end;
[LN6] 98: (Zeiger auf das Ende der Datei setzen)
[PH6] 99: procedure DatZeiger(Nummer, Hi_Offset,
[6D5] 100: Lo_Offset: Integer; Modus: Byte);
[XZ4] 101: begin Cpu.ah:=$42; Cpu.al:=Modus;
[M46] 102: Cpu.bx:=Nummer; Cpu.cx:=Lo_Offset;
[A55] 103: Cpu.dx:=Hi_Offset; MsDos(Cpu);
[0E5] 104: if (Cpu.flags AND 1)=0 then Fehler:=0
[R34] 105: else Fehler:=Cpu.al; end;
[KD3] 106: (Datei schließen)
[MS6] 107: procedure SchliessDat(Nummer: Integer);
[M85] 108: begin Cpu.ah:=$3E; Cpu.bx:=Nummer;
[XV1] 109: MsDos(Cpu);
[0E5] 110: if (Cpu.flags AND 1)=0 then Fehler:=0
[R34] 111: else Fehler:=Cpu.al; end;
[663] 112: (Hauptprogramm)
[XE6] 113: begin Fehler:=0; Dateiname:=ParamStr(1);
```

```
[GW3] 114: Pfad_Name:=Dateiname+#0;
[PX5] 115: Par2:=ParamStr(2); Par3:=ParamStr(3);
[LN5] 116: if Dateiname='?' then Beschreibung
[KE4] 117: else if ((Dateiname<>'') AND
[106] 118: (Par2='')) then Fehleingabe(1) else begin
[686] 119: PfadAdr.Segment:=Seg(Pfad_Name[1]);
[UU5] 120: PfadAdr.Offset:=Ofs(Pfad_Name[1]);
[AA3] 121: OeffneDat(PfadAdr, 0);
[VF3] 122: if Fehler=0 then begin
[AN3] 123: DatZeiger(Handle, 0, 0, 2);
[VF3] 124: if Fehler=0 then begin
[LA5] 125: DatGroesse:=Cpu.dx*65536+Cpu.ax;
[LD3] 126: SchliessDat(Handle);
[YN2] 127: if Fehler<>0 then
[324] 128: Fehleingabe(Fehler+4); end
[SR3] 129: else DatGroesse:=0; end
[9G4] 130: else Fehleingabe(Fehler+3);
[VF3] 131: if Fehler<>0 then begin
[CS4] 132: Val(Par2, Seitzahl, Code);
[LP3] 133: Val(Par3, Anzahl, Code);
[286] 134: RealTeiler:=DatGroesse/Seitzahl;
[EH5] 135: if Round(Frac(RealTeiler)*1000000)<>0
[DU4] 136: then Fehleingabe(2) else begin
[7V4] 137: Seitzahl:=Round(RealTeiler);
[M04] 138: if Anzahl>Seitzahl then
[4Z3] 139: Fehleingabe(4) else begin
[SE4] 140: DatLesen; if Anzahl<>0 then
[285] 141: Seitzahl:=Anzahl; DateiCheck:='';
[EV0] 142: S:='';
[495] 143: for I:=1 to Length(Pfad_Name) do
[4X3] 144: begin S:=Pfad_Name[I];
[DL1] 145: if S<>'.' then
[X14] 146: DateiCheck:=DateiCheck+S
[Y44] 147: else I:=Length(Pfad_Name); end;
[YG2] 148: if Fehler=0 then
[LP4] 149: DatSchreiben(Seitzahl); end;
[X12] 150: end; end; end;
[KN0] 151: end.
```

Listing 17. »cut.pas« zerlegt eine bestehende Datei in mehrere Einzeldateien, mit der als Parameter angegebenen Seitengröße.

Die DOS Extra ist eine Zeitschrift zum Mitmachen. Vor allem die Tips-&-Tricks-Rubriken leben von Ihrer Mitarbeit. Wenn Sie sich bereits besser mit Ihrem PC auskennen, überlegen Sie: Was hat Ihnen als Einsteiger damals besondere Schwierigkeiten bereitet? An welchem Problem sind Sie wieder und wieder gescheitert? Schreiben Sie uns! Durch Ihre Einsendung erhalten auch andere Leser die Gelegenheit, von Ihrer Erfahrung zu profitieren. Ein Beitrag für »Tips & Tricks für Einsteiger« muß weder journalistisch ausgefeilt noch eine überragende Neuentdeckung auf dem PC-Sektor sein. Und ein angemessenes Honorar gibt es dafür natürlich auch (bei Veröffentlichung). Die Höhe des Honorars richtet sich nach der Qualität der Idee und eventueller Listings. Schreiben Sie an:

Redaktion DOS International, Stichwort Tips & Tricks, Gruberstraße 46c, 8011 Poing bei München

Gutschein

für PC-Programme "nur für Erwachsene"

Bilder & Movies auf: ☐ 3 x 3 1/2" bzw. ☐ 6 x 5 1/4" (bitte ankreuzen) Disketten zum Schnupperpreis von

NUR DM 30.-

Gutschein ausschneiden, Bargeld oder Scheck bitte beifügen und diskret im Briefumschlag einsenden an:

EUROS TOP MARKET, Postfach 30, 7701 Büsingen

Name

Strasse

PLZ / Ort

Ihr Geburtsdatum, damit wir Sie überraschen können:

660.18

Scrolling

Trickkiste für VGA-Karten

Die VGA-Karte bietet mit einer Palette von 256 aus 262 144 Farben eine farbenfrohe Spielwiese für Programmierer. Dabei wird im Standard-Modus nur ein Bruchteil der Leistungsfähigkeit der VGA-Karte ausgenutzt. Wie wäre es jedoch mit vier Grafikseiten mit 256 Farben, Softscrolling oder geteiltem Bildschirm?

Als Nachfolger der EGA-Grafikkarte bietet die VGA-Karte selbstverständlich eine höhere Auflösung, aber das eigentliche Leistungsmerkmal ist die 256-Farben-Darstellung. Damit können Sie erstmals Bilder auf den Schirm bringen, die zumindest aus einiger Entfernung wie Fotografien oder Fernsehbilder aussehen. Damit erschließen Sie neue Anwendungen, an die vorher mit den 16 Farben der EGA-Grafikkarte nicht zu denken war – Grund genug, sich mit den 256 Farben der VGA-Karte näher zu beschäftigen.

Wenn Sie sich schon einmal mit der Architektur eines Computers auseinandergesetzt haben, kommt Ihnen die Anzahl 256 sicherlich nicht so willkürlich vor wie einem »Uneinge-weihten«. Denn 256 entspricht der Anzahl der verschiedenen Zustände, die ein Byte speichert. Ein Byte ist die Standard-Größeneinheit zur Organisation des Speichers Ihres Computers. Es besteht aus acht Bits, die acht elementare Informationen wie etwa »an« oder »aus«, »ja« oder »nein«, »Strom« oder »kein Strom« weitergeben; also lassen sich in der Tat 2^8 (also 256) verschiedene Zustände in einem Byte speichern. Jetzt überrascht es Sie sicherlich nicht, wenn im 256-Farben-Modus die VGA-Karte die Farbinformationen jedes Bildpunkts einfach in ein Byte packt. Das ist ganz praktisch, denn nun ist dieses Byte ausschließlich für diesen einen Punkt verantwortlich, und jedes Bit des Speichers wird auch für Farbinformationen genutzt. Die Vermutung liegt nahe, daß alle Punkte des Schirms – von oben links angefangen – in hintereinander liegenden Bytes zu finden sind. In der Tat ist der Speicher im 256-Farben-Modus genau in dieser Weise organisiert. Diese Tatsache ist allerdings nicht so selbstverständlich, wie es im Augenblick erscheint, weist jedoch einige Vorteile auf.

Organisation von Bildpunkten – der Schlüssel zur Grafik

Wie erwähnt liegen alle Farbinformationen des Grafikbildschirms unmittelbar hintereinander. Also liegt der Punkt mit der y-Koordinate 0 und der x-Koordinate 5 genau 5 Byte hinter dem ersten Punkt des Bildschirms (y-Koordinate 0, x-Koordinate 0). Da die VGA-Karte im 256-Farben-Modus in der Horizontalen, also der x-Achse, genau 320 Punkte darstellt, befindet sich der erste Bildpunkt der zweiten Punktzeile dann 320 Byte hinter dem ersten Punkt der ersten Punktzeile. Damit ergibt sich als Byte-Adresse für einen beliebigen Punkt mit den Koordinaten x und y:

Adresse = $y \times 320 + x$

Wenn Sie wissen, daß der VGA-Bildschirm bei der Speicheradresse A000:0000hex beginnt, können Sie jeden einzelnen Bildpunkt kontrollieren. In Pascal machen Sie sich die Arbeit sogar noch einfacher, indem Sie folgende Variable deklarieren:

```
VAR Schirm : ARRAY[0...199,0...319] OF BYTE ABSOLUTE $A000:0000;
```

Damit haben Sie die Variable »Schirm«, die genau die gleiche Struktur wie Ihr Grafikbildschirm besitzt (200x320 Byte), durch die Anweisung »ABSOLUTE \$A000:0000« an die Stelle verlegt, an der der Bildschirmspeicher tatsächlich liegt. Also greifen Sie jetzt auf jeden Bildpunkt direkt mit dieser Variable zu. Wollen Sie wissen, welche Farbe der Punkt in der 117. Zeile und 112. Spalte hat, so betrachten Sie einfach »Schirm[117,112]«. Durch die Anweisung

```
Schirm[117,112]:=farbe;
```

färben Sie diesen Punkt nach Belieben. Die Programmierung ist also auch für den ungeübten Programmierer ein Kinderspiel. Für die meisten Anwendungen reicht dieses Wissen aus, die Programme bleiben zudem übersichtlich und durchschaubar. Möchten Sie jedoch ein schnelles Actionspiel programmieren oder bewegte Grafiken darstellen, sind Sie gezwungen, sich genauer mit der VGA-Karte auseinanderzusetzen. Sie werden sehen, es lohnt sich.

Die VGA-Karte ist EGA-kompatibel. Also kann sie auch alles, was eine EGA-Grafikkarte kann, und stellt obendrein statt nur 16 Farben 256 dar. Der 16-Farben-Modus benötigt nur 4 Bit Farbinformation pro Bildpunkt. Ein ganzes Byte dafür zu verschwenden, wäre viel zu schade, denn schließlich paßt die doppelte Anzahl Bits in ein Byte. Es wäre denkbar, einfach zwei Bildpunkte in ein Byte zu packen, aber in der Praxis gibt es bessere Methoden. Schließlich muß der Grafikspeicher nicht in derselben Weise organisiert werden wie der normale Speicher. Die Entwickler der EGA-Karte konstruierten vier unabhängige Speicherebenen, in denen jeweils nur ein Bit der insgesamt vier Farbinformationsbits gespeichert wird. Die EGA-Karte ist in der Lage, alle vier Ebenen gleichzeitig zu lesen. Sie faßt jeweils alle vier ersten Bits zu einem Punkt zusammen, dann die vier zweiten Bits und so fort. Dieser Aufbau steigert zudem die Geschwindigkeit; das aber ist erforderlich, damit auch bei steigender Auflösung das Bild wenigstens 24mal in der Sekunde aufgebaut werden kann. Daß diese Leistung nicht zu unterschätzen ist, wird deutlich, wenn Sie sich vor Augen führen, daß bei 640x400 Punkten in 16 Farben knapp 3 MByte in der Sekunde gelesen und dargestellt werden müssen.

Die EGA-Karte bearbeitet vier Bitplanes gleichzeitig

Die VGA-Karte muß also ebenfalls diese vier Speicherebenen besitzen, da sie sonst nicht zur EGA-Karte kompatibel ist. Jede dieser Speicherebenen ist 64 KByte groß, alle zusammen also 256 KByte. Bei 320x200 Punkten in 256 Farben werden jedoch nur knapp 64 KByte benötigt. Wenn Sie die vier Speicherebenen vollständig und gleichzeitig nutzbar machen könnten, hätten Sie also mehr als vier Bildschirmseiten in 256 Farben zur Verfügung. Außerdem würden innerhalb des Grafikspeichers interne Operationen ebenfalls viermal so schnell ablaufen – traumhafte Voraussetzungen für Animationen. Und der Preis dieser Vorzüge? Für das Schreiben eines einzelnen Bildpunkts müssen Sie jetzt vier Ebenen selektiv ansprechen. Der einfache Pascal-Befehl »Array« genügt hier nicht mehr – diese Aufgabe übernimmt in der Unit »Vgafox« (Listing 1) die Methode »Init«.

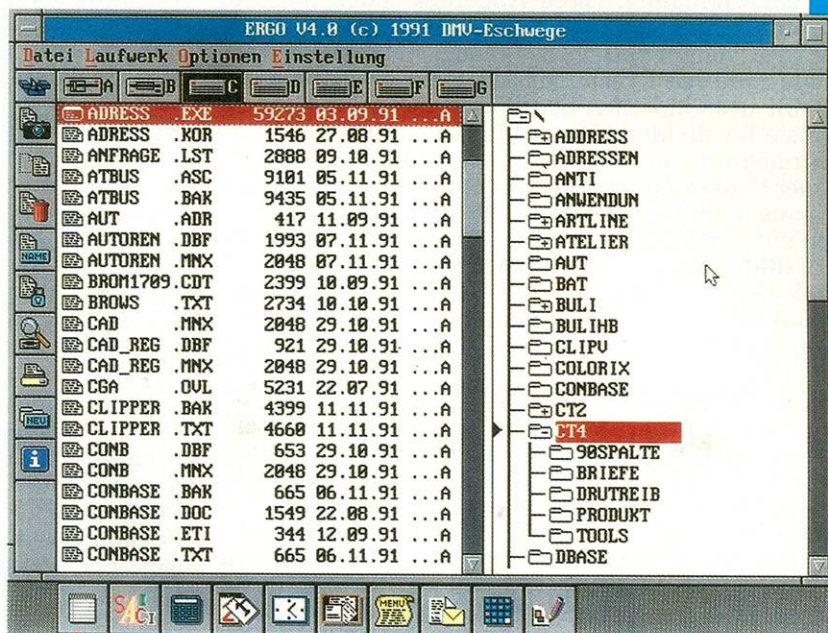
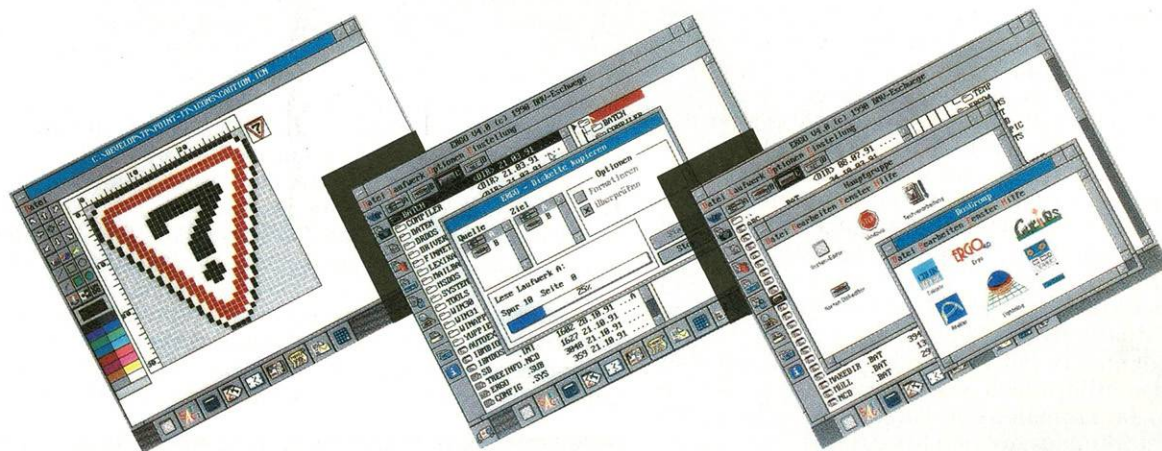
Zunächst rufen Sie den Standard-Modus 13hex für 256 Farben auf. Damit sind Sie in der korrekten Farbdarstellung,

ERGO 4.0

bringt Ihr DOS in Top-Form

ERGO 4.0 verbindet alle DOS-Standardfunktionen mit dem Komfort einer grafischen Benutzeroberfläche.

Es bieten sich Ihnen Funktionen, die weit über die Leistungsfähigkeit des DOS hinausgehen. Das Arbeiten mit ERGO 4.0 verbindet ein Höchstmaß an Bedienungskomfort, Anwendungsvielfalt und Zeitersparnis.



Wir setzen für Sie neue Maßstäbe:

- variable icon-orientierte Grafikoberfläche (SAA-Standard)
- DOS-Funktionen im Icon
- Programm-Manager für individuelle Auswahl Fenster
- automatisches Verknüpfen von Dateien und Anwendungen
- Clipboard zum Zwischenspeichern
- neue Kopierfunktionen ohne Diskettenwechsel
- Verschieben ganzer Verzeichnisse
- und, und, und ...

Das zeichnet ERGO 4.0 aus:

• Dateien: kopieren, löschen, wiederherstellen, verschieben, suchen, umbenennen, filtern, sortieren, verstecken, schützen, drucken
 • Verzeichnisse: anlegen, umbenennen, löschen, verschieben (mit Diskettenwechseln), verstecken • Disketten: formatieren (Quickformat), kopieren (xcopy/diskcopy/multicopy) • Programmstarts vom Sinnbild (Icon) aus • Editor zum Entwerfen und Ändern von Icons • integrierte Textverarbeitung • Byte-Editor für Dateien • Sektor-Editor für Disketten • Kontrolle über speicherresidente Programme • Paßwortsystem • Vergabe von Zugriffsrechten • detaillierte Suchfunktionen für Texte • Zusatzmodule: ASCII-Tabelle, Notepad, Rechner, Uhr, ewiger Kalender, Biorhythmus, Puzzle-Spiel.
 ERGO 4.0 läuft mit jedem IBM PC oder Kompatiblen, der 512 KByte Hauptspeicher, eine Festplatte, VGA-, EGA- oder Hercules-Grafikkarte und DOS ab 3.2 hat.

Achtung, Update Service:

Sie schicken uns als Benutzernachweis entweder eine Kopie Ihrer Kaufrechnung, Ihre Original-Ergo-Disketten oder die Original-Titelseite Ihres Ergo-Handbuches.

ERGO 4.0 199,— DM*

ERGO Update 99,— DM*

Für Windows-Anwender:

Die freundlichere Oberfläche für MS-Windows 3.0

Grips-Desktop und -DeskDraw 199,— DM*

* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.

Sie müssen nur noch die Adressierungsart ändern. Die lineare Speicherorganisation des Standard-Modus wird Chain-4-Adressierung genannt und durch Bit 3 des Memory-Mode-Registers des Sequenzers der Grafikkarte geschaltet. Dieses Bit löschen Sie, und sogleich stehen Ihnen die vier Ebenen nicht als Kette (Chain), sondern als parallele Ebenen zur Verfügung. Diese Änderung teilen Sie dem CRT-Controller mit, der für die Darstellung des Bilds auf dem Schirm verantwortlich ist, indem Sie vom Doppelwortmodus auf den Bytemodus schalten: Sie löschen das sechste Bit des Underline-Location-Registers und setzen das sechste Bit im Mode-Control-Register. Nun stellt eine Bildschirmadresse nicht mehr einen Bildpunkt, sondern gleich vier Punkte dar, verteilt auf die vier parallelen Speicherebenen. Damit stehen Ihnen 256 KByte Bildschirmspeicher zur Verfügung, also ein virtueller Bildschirm von 320x819 Bildpunkten. Virtuell heißt, daß diese Auflösung nicht vollständig sichtbar ist, sondern daß Sie nur den gewohnten Bereich von 320x200 Bildpunkten sehen. Sie bestimmen, welcher Ausschnitt des gesamten Bereichs jeweils sichtbar ist.

In manchen Fällen mag die Auflösung von 320x819 Punkten unpraktisch sein, zweckmäßiger wären vielleicht 200x1310 oder 500x524 Punkte. Solange das Produkt aus Zeilen- und Spaltenzahl den 256-KByte-Rahmen nicht sprengt, das heißt kleiner als 262 144 ist, können Sie die Auflösung frei wählen. Sie müssen lediglich die neue virtuelle Breite des Schirms dem CRT-Controller in seinem Offset-Register (Index 13hex) mitteilen. Da vier Bildpunkte unter einer einzigen Adresse (in jeweils verschiedenen Speicherebenen) gespeichert werden, geben Sie nur ein Viertel der tatsächlichen Spaltenanzahl ein. Bei einem virtuellen Schirm von 320x819 Bildpunkten, also einer virtuellen Breite von 320 Bildpunkten, teilen Sie folglich dem CRT-Controller den Wert 80 als Breite mit. Falls Sie im Bytemodus arbeiten, genügt sogar die Hälfte dieses Wertes, in diesem Fall sind das 40. Es folgt also, daß die virtuellen Bildschirmbreiten ganzzahlige Vielfache von 8 sein müssen. Darum kümmert sich jedoch die Init-Methode des TVGA-Objekts im Programm selbst. Ist die Breite kein Vielfaches von 8, wird die virtuelle Breite korrigiert. Da sich die virtuelle Höhe direkt aus der virtuellen Breite ergibt (wegen der Begrenzung auf 256 KByte Videospeicher insgesamt), verlangt die Init-Methode als Parameter nur die Breite des virtuellen Schirms. Die tatsächlich verwendeten Breiten und Höhen geben die beiden Funktionen »VirtualScreenWidth« und »VirtualScreenHeight« an.

Wie schon angedeutet, ist bei Verwendung von vier Ebenen das Setzen eines Bildpunkts etwas schwieriger als im Chain-4-Modus, der normalen 256-Farben-Darstellung. Die Adresse eines Bildpunktes mit den Koordinaten x und y ergibt sich jetzt wie folgt:

$y \times [\text{virtuelle Breite} / 4] + [x/4]$

Darin bedeuten die eckigen Klammern die nächstkleinere ganze Zahl.

Bei dieser Adressenberechnung gehen offensichtlich Informationen verloren, nämlich die Divisionsreste. Beispielsweise ergeben die ersten vier Bildpunkte (x-Koordinaten 0 bis 3) auf der gleichen y-Koordinate dieselbe Adresse. Diese fehlenden Informationen teilen Sie dem CRT-Controller gesondert mit: Im Map-Mask-Register legen Sie durch die ersten vier Bits fest, welche Ebene Sie ansprechen. Hier hinein gehört also der Divisionsrest $x \bmod 4$. Sie setzen dieses Register, indem Sie das gewünschte Bit um den Divisionsrest nach links verschieben:

$1 \text{ SHL } (x \text{ AND } 3)$

Da jede Division durch 4 im Binärsystem einer Division durch "0100" entspricht, liefert die AND-Operation mit 3 = "0011" gerade den Divisionsrest.

Anschließend schreiben Sie Ihren Farbpunkt an die errechnete Adresse direkt in den Videospeicher:

`Mem[$A000:y*(virtuelle Breite DIV 4)+(x DIV 4)]:=farbe;`

So kompliziert dieses Verfahren auf den ersten Blick erscheinen mag, so hat es doch große Vorzüge. Wenn Sie zum Beispiel nicht nur eine, sondern alle vier Ebenen aktivieren, (also den Wert 0Fhex in das Map-Mask-Register schreiben), dann schreiben Sie vier Bildpunkte gleichzeitig. Dies ist besonders sinnvoll, wenn Sie große Flächen füllen oder den Bildschirm löschen möchten.

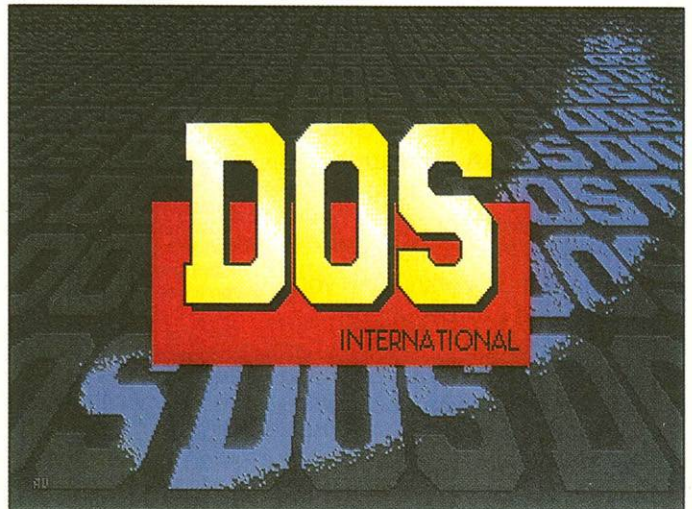
Parallele Speicherebenen erhöhen die Effektivität

Das Lesen eines Bildpunktes erfolgt auf ähnliche Weise. Nur können Sie nicht alle vier Ebenen auf einmal einlesen, da ein konventioneller Speicher eben nicht in vier Ebenen organisiert ist. Deshalb ist das Read-Map-Select-Register, durch das Sie angeben, welches der vier möglichen Punkte Sie lesen möchten, auch anders aufgebaut. Hier reicht es, den Rest der Division durch 4 anzugeben, also $(x \text{ AND } 3)$. Abgesehen davon läuft das Auslesen eines Bildpunkts wie beim Schreiben.

In der Unit »Vgafx« übernehmen das Setzen und Lesen von Bildpunkten die Methoden »SetPix« und »GetPix«, denen Sie jeweils die x- und y-Koordinaten übergeben. Bei »SetPix« kommt noch das Farbbyte hinzu, »GetPix« liefert es als Funktionswert zurück.

Nun sind 256 Farben zwar beachtlich, aber die voreingestellten Farben müssen nicht jedermanns Geschmack treffen. Also fehlen noch Methoden, mit denen Sie neue Farben aus der Palette von 262 144 Farben setzen können. Hier hilft Ihnen der Interrupt 10hex des BIOS. Sie übergeben einfach die Farbnummer dem Register »BX«, den Grün-Anteil der Farbe dem Register »CH«, den Blau-Anteil dem Register »CL« und den Rot-Anteil dem Register »DH« (siehe Methode »SetCol«). Jedoch werden nur die unteren sechs Bits der verschiedenen Farbanteile benutzt; die Bits 6 und 7 werden nicht ausgewertet.

Da es sehr umständlich ist, alle 256 Farbregister einzeln zu setzen, sind in der Unit »Vgafx« die Typen »TColor« und »TPalette« definiert, die die Rot/Grün/Blau-Anteile einer Farbe beziehungsweise der gesamten Palette von 256 Farben aufnehmen. Mit den Methoden »SetPal(s,nr,p)« und »GetPal(s,nr,p)« übertragen Sie dann, beginnend bei Farbregister »s«, insgesamt »nr« Farbregister aus oder in Palette »p«. Hier greifen Sie auf die Interruptfunktionen 1012hex und 1017hex des Interrupts 10hex zurück. Haben Sie die Farben geschickt gewählt, realisieren Sie bereits mit geringem Aufwand Animationen durch rasches Wechseln der



Bilder mit 256 Farben lassen Sie mit der Unit »Vgafx« über den Bildschirm flitzen.

Farben. Deshalb ist in »Vgafx« auch eine einfache Methode »CyclePal(s,nr)« zur Palettenrotation (Colorcycling) enthalten. Sie übergeben die Nummer der ersten Farbe, bei der die Palettenrotation beginnen soll, sowie die Anzahl der Farben. Innerhalb dieser Methode wird die Prozedur »Wait-Display« mehrfach aufgerufen. Sie bewirkt, daß zur gleichen Zeit, wenn der Kathodenstrahl des Monitors vom unteren zum oberen Bildschirmrand springt, selbst umfangreiche Paletten geändert werden. Dadurch treten in den meisten Fällen unangenehme Nebeneffekte wie Schneegestöber oder Bildschirmflackern erst gar nicht auf. Dennoch ist es bei langsamen VGA-Karten ratsam, nicht alle 256 Farben mit einem Schlag zu ändern; dazu ist die Zeitspanne zu kurz, in der der Strahl umspringt. Übergeben Sie lieber kleinere Bereiche nacheinander an die Grafikkarte.

Mit den bisherigen Errungenschaften lassen sich auch sanft gleitende Bildverschiebungen (Softscrolling) realisieren. Der CRT-Controller stellt unter den Indizes »0Chex« und »0Dhex« zwei Register für die Startadresse der Bildschirmdarstellung bereit. Erhöhen Sie diese Adresse um die virtuelle Bildschirmbreite, rückt das Bild für den Betrachter eine Zeile nach oben. Wenn Sie die Adresse um 1 erhöhen, rückt das Bild vier Punkte nach links. Entsprechendes gilt für negative Werte.

Das Rollen um vier Punkte auf einmal verläuft in horizontaler Richtung unregelmäßig. Diesen Mangel beheben Sie mit dem Horizontal-Pixel-Panning-Register. Das Prinzip ist zwar dasselbe wie beim Map-Mask-Register oder beim Read-Map-Mask-Register, jedoch der Aufwand wesentlich größer. Denn der Index des Registers und die Daten für das Register werden abwechselnd an der gleichen Portadresse erwartet. Damit Sie nicht Gefahr laufen, Daten zu liefern,

wenn ein Index gefordert ist (und umgekehrt), lesen Sie das Input-Status-Register #1 aus (siehe Dummy-Zuweisung, Listing 1, Zeile 274). Dies war aber noch nicht die letzte Hürde, die das Horizontal-Pixel-Panning-Register stellt. Es läßt nämlich entweder nur Sie (das heißt die CPU) oder nur die Video-Hardware auf sich zugreifen. Die Zugriffsrechte legen Sie im Attribut-Index-Register mit Bit 5 fest (Tabelle). Es sind somit folgende Schritte nötig:

1. Lesen des Input-Status-Registers #1
 2. Bit 5 im Attribut-Index-Register löschen und Index des Horizontal-Pixel-Panning-Registers angeben
 3. Das Horizontal-Pixel-Panning-Register mit dem gewünschten Wert laden
 4. Bit 5 des Attribut-Index-Registers wieder setzen, um der Video-Hardware erneut die Kontrolle zu übergeben
- Wenn Sie jetzt noch die richtigen Werte (x AND 3) SHL 1 an das Horizontal-Pixel-Panning-Register übergeben, haben Sie es geschafft.

Lokales Rollen – Vermeidung wandernder Menüzeilen

Damit ist sanft gleitendes Rollen in alle vier Richtungen kein Problem mehr. Die Methode »SetView(x,y)« positioniert den virtuellen Bildschirm so, daß x und y den Koordinaten der linken oberen Ecke des Monitors entsprechen. Die Methoden »ScrollLeft«, »ScrollRight«, »ScrollUp« und »ScrollDown« erhöhen beziehungsweise erniedrigen jeweils die x- oder y-Koordinate um 1 – und schon rollt der virtuelle Schirm unter dem Monitor ganz nach Wunsch. Durch die Funktionen »StartViewWidth« und »StartView-

DMV-Integrator für Clipper

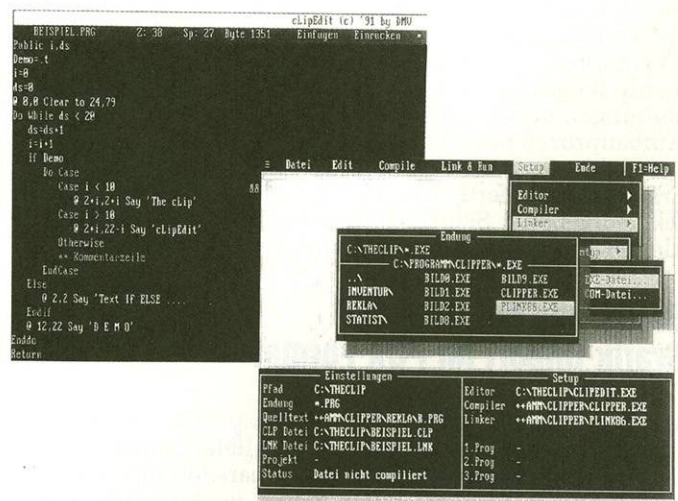
Moderne Programmiersprachen brauchen eine professionelle Entwicklungsumgebung

Der DMV-Integrator ist eine professionelle Entwicklungsumgebung, und zwar für Clipper®. Er vereinfacht und beschleunigt das Editieren, Compilieren und Linken Ihrer Programme, denn er faßt alle Programmaufgaben unter einer Oberfläche und für alle Clipper-Versionen zusammen.

Diese Leistungsmerkmale zeichnen den DMV-Integrator aus:

Die Programmierungsumgebung: Dateiauswahlbox • einstellbare Dateieinstellungen • compiliert und linkt Programme auf Knopfdruck • startet compiliertes Programm • Datei-Liste mit bis zu 10 Dateinamen für schnelles Laden • einfache Projektverwaltung • frei konfigurierbare Farbeinstellung • nutzt EMS-Speicher • arbeitet auch mit fremden Editoren zusammen • DOS-Shell • Online-Hilfe-System • Bedienung durch Pull-down-Menüs

Der Editor: • Wordstar®-kompatibler Editor • mit allen Standardfunktionen • automatisches Einrücken • Zeilenlänge bis 999 Zeichen • bis zu 6 Dateien werden gleichzeitig bearbeitet • Dateigröße nur durch freien Speicherplatz eingeschränkt • farbliche Hervorhebung von Clipper-Schlüsselworten • farbliche Kennzeichnung von Kommentarzeilen • informative Online-Hilfe



Systemkonfiguration: Festplatte, 640 KByte RAM, DOS ab 3.2 oder komp. System

DMV-Integrator für Clipper 199,— DM*

Informieren Sie sich auch über die

DMV-ProfiTools für Clipper 199,— DM*

* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung.

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV
Daten- und
Medienverlag

Height« rufen Sie die derzeitige x- und y-Koordinate der linken oberen Ecke der Monitordarstellung ab.

Falls Sie die Farbenpracht der VGA-Karte für ein Spiel ausnutzen wollen, bei dem das Spielfeld mit den bisher entwickelten Methoden beliebig rollt, stehen Sie vor einer weiteren Schwierigkeit: Die meisten Spiele benötigen eine Status-Anzeige für Punkte, gesammelte Gegenstände und dergleichen. Wenn jedoch der gesamte Bildschirm rollt, wo stellen Sie diese Informationen dar? Auch hier bietet die VGA-Karte eine elegante Lösung. Der Bildschirm läßt sich horizontal teilen. Der untere Bereich eignet sich hervorragend für die nötigen Statusanzeigen. Verantwortlich für einen solchen geteilten Bildschirm (split screen) ist im wesentlichen das Line-Compare-Register des CRT-Controllers. Damit legen Sie am unteren Bildrand einen beliebigen großen Bereich fest, der nicht rollt. Der CRT-Controller zählt mit Hilfe eines Zeilenzählers (Scan Line Counter) mit, wieviele Zeilen er schon angezeigt hat. Sobald dieser Zähler mit dem Inhalt des Line-Compare-Registers übereinstimmt, fängt der Controller einfach wieder bei Zeile Null des virtuellen Bildschirms zu zählen an. Zusätzlich zu den acht Bits des Line-Compare-Registers existieren noch ein neuntes und zehntes Bit im Overflow- beziehungsweise im Maximum-Scan-Line-Register, damit auch Angaben von bis zu 1023 Zeilen darstellbar sind (siehe Tabelle). Die VGA-Karte stellt in den 256-Farben-Modi alle Zeilen doppelt dar (double scan), um die Ausgabequalität zu erhöhen. Deshalb übergeben Sie immer das Doppelte der tatsächlich gewünschten Zeilenzahl. Sie müssen nur noch das Horizontal-Pixel-Panning-Register für den unteren Abschnitt abschalten, denn schließlich soll der untere Bereich nicht rollen. Setzen Sie dazu das Bit 5 des Mode-Control-Registers (siehe Tabelle). In der Unit »Vgafx« brauchen Sie sich darum jedoch nicht mehr zu kümmern. Sie rufen »Set Splitt(y)« mit der Anzahl der Zeilen am unteren Bildschirmrand auf, die nicht mitrollen sollen. Danach stellen Sie Ihre Statusinformationen oder Menüzeilen ab Zeile 0, Spalte 0 des virtuellen Bildschirms dar.

Im Prinzip sind Ihnen damit alle Werkzeuge für effektvolle VGA-Programmierung an die Hand gegeben. Damit es noch leichter wird und Sie schneller zu vernünftigen Ergebnissen kommen, sind noch einige nützliche Methoden in das TVGA-Objekt eingebaut. Unter Umständen dauert es ein wenig länger, die 256 KByte des Bildspeichers Ihren Vorstellungen entsprechend zu füllen. Damit niemand Ihren Aufbauprozeß bruchstückhaft verfolgen muß, schalten Sie den Schirm mit der Methode »Hide« einfach auf »dunkel«. Erst wenn alles fertig ist, rufen Sie »Show« auf, und der perfekt aufgebaute Schirm präsentiert sich dem Betrachter: kleiner Aufwand, große Wirkung.

Bilder aus der Dose – Grafik-Import im PCX-Format

Mit den beiden Methoden »Clear(x,y,w,h)« und »Copy(x,y,w,h,nx,ny)« nutzen Sie die parallele Verarbeitung aller vier Speicherebenen aus. Mit »Clear« löschen Sie einen $w \times h$ großen Bereich ab Position »x,y«. Mit einer Zuweisung löschen Sie jeweils vier horizontal benachbarte Bildpunkte gleichzeitig. Allerdings beginnt das Löschen nur an einer Spalte, die durch 4 teilbar ist. Ebenso müssen Sie die Breite »w« durch 4 teilen können. Entsprechendes gilt für die Copy-Methode. Mit ihr kopieren Sie einen Bildausschnitt an die Stelle »nx,ny«. Dabei müssen alle horizontalen Koordinaten (x,w,nx) wieder durch 4 teilbar sein. Die Copy-Methode benutzt einen speziellen Schreibmodus der VGA-Karte und kopiert damit sehr viel schneller, als wenn Sie Bildpunkt für Bildpunkt einzeln setzen. Jedoch muß dazu der Bildbereich, den Sie vervielfältigen wollen, im Videospeicher liegen. Beim ersten Mal kommen Sie also um

Port-Adresse	Index	Bit	Bedeutung
Memory-Mode-Register 3C4hex, 3C5hex	4hex	0	0:Grafikmodus, 1:Textmodus
		1	Video-RAM 0: 64 KByte, 1: > 64 KByte
		2	Odd/Even-Adressierung: 0: ein, 1: aus
		3	Chain-4-Adressierung: 0: aus, 1: ein
Underline-Location-Register 3D4hex, 3D5hex	14hex	4-7	unbenutzt
		0-4	Unterstrich-Rasterzeile
		5	0: Adreßzählertakt normal, 1: Adreßzählertakt/4
Offset-Register 3D4hex, 3D5hex	13hex	6	0: VGA-Wortmodus, 1: VGA-Doppelwortmodus
		7	unbenutzt
		0-7	Bytemodus: Bytes pro Zeile /2 Wortmodus: Bytes pro Zeile /4 Doppelwortmodus: Bytes pro Zeile /8
Mode-Control-Register 3D4hex, 3D5hex	17hex	0	Adreßbit 13: 0: Zeilenbit 0, 1: unverändert
		1	Adreßbit 14: 0: Zeilenbit 1, 1: unverändert
		2	0: Rasterzeilentakt normal , 1: Rasterzeilentakt/2
		3	0: Adreßzählertakt normal, 1: Adreßzählertakt/2
		4	EGA-Video-Treiber: 0: ein, 1: aus
		5	Adreßrotation: 0: 14 Bit, 1: 16 Bit
		6	0: Wortmodus, 1: Byte-modus
		7	CRTC-Reset und 0: anhalten, 1: beenden
Map-Mask-Register 3C4hex, 3C5hex	2hex	0	Schreiben Ebene 0, 0: aus, 1: ein
		1	Schreiben Ebene 1, 0: aus, 1: ein
		2	Schreiben Ebene 2, 0: aus, 1: ein
		3	Schreiben Ebene 3, 0: aus, 1: ein
Read-Map-Select-Register 3CEhex, 3CFhex	4hex	4-7	unbenutzt
		0-1	Nummer der Speicherebene
Horizontal-Pixel-Panning-Register 3C0hex	13hex	0-3	Anzahl der zu verschiebenden Punkte
		4-7	unbenutzt
Attribut-Index-Register 3C0hex		0-4	Registerindex (0...31)
		5	0: CPU-Zugriff, 1: Video-Hardware-Zugriff
Line-Compare-Register 3D4hex,3D5hex	18hex	6-7	unbenutzt
		0-7	Split-Screen-Rasterzeile im Overflow-Register (Index 07hex, Bit 4)
		8	im Maximum-Scanline-Register (Index 09hex, Bit 6)
Mode-Control-Register (Attribute Controller) 3C0hex	10hex	9	im Maximum-Scanline-Register (Index 09hex, Bit 6)
		0	0: Textmodus, 1: Grafikmodus
		1	0: Farbdisplay, 1: Monochromdisplay
		2	9. Zeichenbox-Pixelspalte 0: aus, 1: an
		3	Attributbit 0: Farbe, 1: Blinken
		4	unbenutzt
		5	Softscrolling 0: ganzer Schirm, 1: bis Compare Line
		6	Farbauswahl 0: 4 Bit, 1: 8 Bit
7	Farbseitenmodus 0: 4 x 64, 1: 16 x 16		

Die Tabelle zeigt die wichtigsten Register zur Programmierung der VGA-Karte.

die bildpunktweise Übergabe der Farbwerte nicht herum. Nicht jeder kann ein Pixelkünstler sein, der durch einfache SetPix-Befehle ansprechende grafische Meisterwerke auf den Bildschirm zaubert. In diesem Fall ist für Sie die Methode »PCX(f,x,y)« besonders wertvoll. Sie lädt ein PCX-Bild an die Koordinaten x,y. Dies Paintbrush-Bildformat wird von vielen Malprogrammen unterstützt. Sie können allerdings nur Bilder laden, die tatsächlich 256 Farben enthalten. Sie erhalten solche Bilder auf dem Public-Domain-Markt, einschließlich nützlicher Programme, mit denen Sie die Motive der Bilder einsehen. Mit dem Sharewareprogramm »Vpic« wandeln Sie mühelos GIF-Bilder in PCX-Bilder um; diese laden Sie dann mit »PCX« in Ihren Videospeicher. Damit steht der freien Entfaltung Ihrer Kreativität nichts mehr im Weg. Wenn Sie mehr Bewegung in Ihr Programm bringen wollen, sei Ihnen der Artikel zur Sprite-Programmierung in dieser Ausgabe empfehlen.

Auch wenn Sie keinerlei Erfahrung mit Objekten in Turbo-Pascal besitzen, werden Sie die Methoden der Unit »Vgafx« leicht für sich arbeiten lassen. Alle Prozeduren und Funktionen sind zu einem Object-Typ TVGA zusammengefaßt, die Variable »vga« ist von diesem Typ. Sie rufen die Methoden auf, indem Sie »vga.Methodenname;« eingeben; »Methodenname« steht dabei für eine beliebige Prozedur oder Funktion des Objekts TVGA. Zuletzt noch ein Tip bei der Fehlersuche in Programmen, die die Unit »Vgafx« verwenden. Falls Sie den integrierten Debugger benutzen, um sich Variableninhalte oder dergleichen anzusehen, restauriert Turbo Pascal für eigene Belange alle Videoregister, damit es Ihnen den gewohnten Textbildschirm anzeigen kann. Wenn das Programm dann weiterläuft, bleiben die meisten von Turbo Pascal eingestellten Werte erhalten. Das Programm kann keine korrekten Bildschirminhalte mehr anzeigen. Haben Sie das Programm also einmal angehalten, das heißt, wenn Sie wieder in der Entwicklungsumgebung von Turbo Pascal sind, erhalten Sie keine korrekte Bildausgabe mehr, wenn Sie nun das Programm an dieser Stelle fortsetzen. Sie sollten ausgiebig Haltepunkte (Breakpoints) verwenden, damit Sie den Programmfluß nur dort stoppen, wo es Ihnen auch wirklich nützt. Hier leistet Ihnen Turbo Pascal 6.0 mit seinen umfangreichen Debug-Funktionen gute Dienste.

Das kurze Programm »demo.pas« (Listing 2) dient als Beispiel und Anregung für eigene Experimente mit der VGA-Karte und der Unit »Vgafx«. Viel Spaß! (Frank Höppner/bm)

```
[ME3] 1: {Programm: VGAfx.pas
[N17] 2: Funktion: Scroll-Routinen für VGA-Karten
[G64] 3: Sprache: Turbo Pascal 6.0
[K34] 4: Autor: Frank Höppner
[585] 5: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 6:
[GX6] 7: {Object-Typ TVGA zur hardwarenahen Aus-
[TB6] 8: nutzung der Leistungen von VGA-Karten
[3B7] 9: Registerkompatibilität Voraussetzung! }
[000] 10:
[B12] 11: unit VGAfx;
[000] 12:
[SK1] 13: interface
[000] 14:
[6X0] 15: const
[KN3] 16: gScreenSegment = $A000;
[8W1] 17: gCRTC = $3D4;
[000] 18:
[RN0] 19: var
[CH5] 20: ScrAddr : Word absolute $0000:$044E;
[RN5] 21: ScrWidth : Word absolute $0000:$044A;
[000] 22:
[ST0] 23: type
[W85] 24: TColor = record r,g,b : Byte end;
[VK5] 25: TPalette = Array[Byte] of TColor;
[000] 26:
[E12] 27: TVGA = object
[672] 28: viewx,viewy,
[573] 29: width,height : Word;
[LS2] 30: pelpos : Byte;
[VF4] 31: Constructor Init(w:Word);
[4Y2] 32: procedure ClrScr;
```

```
[GK5] 33: function GetPix(x,y:Word):Byte;
[AZ5] 34: procedure SetPix(x,y:Word; f:Byte);
[DH6] 35: procedure SetCol(nr:Word; r,g,b:Byte);
[D66] 36: procedure SetPal(s,nr:Integer; var p);
[X46] 37: procedure GetPal(s,nr:Integer; var p);
[0N5] 38: procedure CyclePal(s,nr:Integer);
[BX4] 39: procedure SetView(x,y:Word);
[T05] 40: procedure SetSplit(y:Word);
[0R3] 41: procedure ScrollLeft;
[LY3] 42: procedure ScrollRight;
[XG3] 43: procedure ScrollUp;
[CT3] 44: procedure ScrollDown;
[K26] 45: function VirtualScreenWidth:Word;
[K56] 46: function VirtualScreenHeight:Word;
[DE5] 47: function StartViewWidth:Word;
[DH5] 48: function StartViewHeight:Word;
[7N2] 49: procedure Hide;
[YS2] 50: procedure Show;
[A45] 51: procedure Clear(x,y,w,h:Word);
[896] 52: procedure Copy(vx,vy,w,h,nx,ny:Word);
[4X5] 53: procedure PCX(f:string; x,y:Word);
[9T2] 54: Destructor Done;
[000] 55:
[2D1] 56: private
[000] 57:
[9Z3] 58: procedure WaitDisplay;
[LY3] 59: procedure WaitRetrace;
[4T4] 60: procedure SetMode(r,w:BYTE);
[ZN0] 61: end;
[000] 62:
[RN0] 63: var
[WU1] 64: vga : TVGA;
[000] 65:
[4L2] 66: implementation
[000] 67:
[TK5] 68: constructor TVGA.Init(w : Word);
[313] 69: var mem : Longint;
[MU0] 70: begin
[EX3] 71: Mem:=262080; w:=(w DIV 8)*8;
[SZ2] 72: if w<320 THEN w:=320;
[A73] 73: if w>1310 THEN w:=1310;
[6H4] 74: width:=w; height:=Mem DIV w;
[7Z3] 75: w:=w DIV 8; ScrWidth:=w;
[7V4] 76: pelpos:=0; viewx:=0; viewy:=0;
[FN0] 77: asm
[TP4] 78: ; {normaler 320x200x256 Mode}
[8N1] 79: mov AX,13h
[Y71] 80: int 10h
[VW4] 81: ; {Chain-4-Adressierung aus}
[PX1] 82: mov DX,03C4h
[5M1] 83: mov AL,04h
[6H1] 84: out DX,AL
[S11] 85: inc DX
[161] 86: in AL,DX
[ZD2] 87: and AL,(NOT 8h)
[6H1] 88: out DX,AL
[L54] 89: ; {Doppelwortmode aus}
[N42] 90: mov DX,gCRTC
[6M1] 91: mov AL,14h
[6H1] 92: out DX,al
[S11] 93: inc DX
[161] 94: in AL,DX
[MG2] 95: and AL,(NOT 40h)
[6H1] 96: out DX,AL
[Z43] 97: ; {Bytemodus ein}
[L01] 98: dec DX
[CM1] 99: mov AL,17h
[6H1] 100: out DX,AL
[S11] 101: inc DX
[161] 102: in AL,DX
[181] 103: or AL,40h
[6H1] 104: out DX,AL
[T14] 105: ; {neuen Width-Offset}
[N42] 106: mov DX,gCRTC
[GV1] 107: mov AL,13h;
[6H1] 108: out DX,AL
[S11] 109: inc DX
[GC1] 110: mov AX,w
[6H1] 111: out DX,al
[ZN0] 112: end;
[DB1] 113: ClrScr;
[ZN0] 114: end;
[000] 115:
[4L5] 116: procedure TVGA.ClrScr; ASSEMBLER;
[FN0] 117: asm
[0S3] 118: mov AX,gScreenSegment
[3H1] 119: mov ES,AX
[PX1] 120: mov DX,03C4h
[NW1] 121: mov AX,0F02h
[YH1] 122: out DX,AX
[GH1] 123: xor AX,AX
[EF1] 124: xor DI,DI
[V42] 125: mov CX,07FFFh
[KN1] 126: rep stosw
[ZN0] 127: end;
```

Listing 1. Die mit Assembler-Routinen optimierte Turbo-Pascal-Unit »vgafx.pas«


```

[000] 128:
[186] 129: function TVGA.GetPix(x,y:Word):Byte;
[PT1] 130: ASSEMBLER;
[FN0] 131: asm
[BC1] 132: push DS
[3B1] 133: push DI
[KC1] 134: mov AX,y
[WH3] 135: mov BX,DS:[DI+width]
[LA1] 136: shr BX,1
[LA1] 137: shr BX,1
[N31] 138: mul BX
[LC1] 139: mov BX,x
[LA1] 140: shr BX,1
[LA1] 141: shr BX,1
[AD1] 142: add AX,BX
[3G1] 143: mov DI,AX
[0S3] 144: mov AX,gScreenSegment
[3H1] 145: mov ES,AX
[NC1] 146: mov CX,x
[TB1] 147: and CL,3h
[4F1] 148: mov AH,CL
[TE1] 149: mov AL,4h
[UZ1] 150: mov DX,03CEh
[YH1] 151: out DX,AX
[YL3] 152: mov AL,byte ptr ES:DI
[0F1] 153: xor AH,AH
[G21] 154: pop DI
[E31] 155: pop DS
[ZN0] 156: end;
[ST6] 157: procedure TVGA.SetPix(x,y:Word; f:Byte);
[PT1] 158: ASSEMBLER;
[FN0] 159: asm
[BC1] 160: push DS
[3B1] 161: push DI
[KC1] 162: mov AX,y
[WH3] 163: mov BX,DS:[DI+width]
[LA1] 164: shr BX,1
[LA1] 165: shr BX,1
[N31] 166: mul BX
[LC1] 167: mov BX,x
[LA1] 168: shr BX,1
[LA1] 169: shr BX,1
[AD1] 170: add AX,BX
[3G1] 171: mov DI,AX
[0S3] 172: mov AX,gScreenSegment
[3H1] 173: mov ES,AX
[NC1] 174: mov CX,x
[TB1] 175: and CL,3h
[WF1] 176: mov BX,1h
[5F1] 177: shl BX,CL
[3F1] 178: mov AH,BL
[RE1] 179: mov AL,2h
[ZA1] 180: mov CL,f
[PX1] 181: mov DX,03C4h
[YH1] 182: out DX,AX
[Y64] 183: mov byte ptr ES:[DI],CL
[G21] 184: pop DI
[E31] 185: pop DS
[ZN0] 186: end;
[C47] 187: procedure TVGA.SetCol(nr:Word; r,g,b:Byte);
[PT1] 188: ASSEMBLER;
[FN0] 189: asm
[DH1] 190: mov BX,nr
[1B1] 191: mov DH,r
[MA1] 192: mov CH,g
[VA1] 193: mov CL,b
[5V1] 194: mov AX,1010h
[Y71] 195: int 10h
[ZN0] 196: end;
[CD7] 197: procedure TVGA.SetPal(s,nr:Integer; var p);
[PT1] 198: ASSEMBLER;
[FN0] 199: asm
[EC1] 200: mov BX,s
[FH1] 201: mov CX,nr
[KL1] 202: mov AH,10h
[2M1] 203: mov AL,12h
[3B1] 204: les DX,p
[Y71] 205: int 10h
[ZN0] 206: end;
[0D7] 207: procedure TVGA.GetPal(s,nr:Integer; var p);
[PT1] 208: ASSEMBLER;
[FN0] 209: asm
[EC1] 210: mov BX,s
[FH1] 211: mov CX,nr
[KL1] 212: mov AH,10h
[CM1] 213: mov AL,17h
[3B1] 214: les DX,p
[Y71] 215: int 10h
[ZN0] 216: end;
[FF6] 217: procedure TVGA.CyclePal(s,nr:Integer);
[SU2] 218: var p : TPalette;
[7V1] 219: c : TColor;
[MU0] 220: begin
[272] 221: WaitDisplay;
[LG2] 222: GetPal(s,nr,p);
[272] 223: WaitDisplay;
[AA4] 224: SetPal(Succ(s),Pred(nr),p);
[272] 225: WaitDisplay;
[KV5] 226: with p[pred(nr)] do SetCol(s,r,g,b);

```

```

[ZN0] 227: end;
[GM5] 228: procedure TVGA.SetView(x,y:Word);
[6U2] 229: var dummy:Word;
[MU0] 230: begin
[RW3] 231: if x+320>Width then Exit;
[L44] 232: if y+200>Height then Exit;
[HD3] 233: viewx:=x; viewy:=y;
[Y2] 234: PelPos:=(x MOD 4)*2;
[G74] 235: x:=x DIV 4+y*(width DIV 4);
[272] 236: WaitDisplay;
[FN0] 237: asm
[CR2] 238: ; {ScrAdresse}
[N42] 239: mov DX,gCRTC
[LC1] 240: mov BX,x
[VE1] 241: mov AH,BH
[3N1] 242: mov AL,0Ch
[YH1] 243: out DX,AX
[3F1] 244: mov AH,BL
[E01] 245: inc AL
[YH1] 246: out DX,AX
[ZN0] 247: end;
[S42] 248: WaitRetrace;
[SE2] 249: PORT[$3C0]:=$13;
[8B3] 250: PORT[$3C0]:=PelPos;
[FE2] 251: PORT[$3C0]:=$20;
[W12] 252: ScrAddr:=x;
[ZN0] 253: end;
[LS5] 254: procedure TVGA.SetSplitt(y:Word);
[3S2] 255: var dummy:Byte;
[MU0] 256: begin
[MR2] 257: y:=400-(y MOD 200)*2;
[272] 258: WaitDisplay;
[9W2] 259: PORT[gCRTC]:=$18;
[1H3] 260: PORT[gCRTC+1]:=Lo(y);
[4W2] 261: PORT[gCRTC]:=$07;
[B22] 262: if (y AND $100)>0
[DH3] 263: then PORT[gCRTC+1]:=
[853] 264: PORT[gCRTC+1] OR $10
[AK3] 265: else PORT[gCRTC+1]:=
[N04] 266: PORT[gCRTC+1] AND NOT $10;
[6W2] 267: PORT[gCRTC]:=$09;
[272] 268: WaitDisplay;
[C22] 269: if (y AND $200)>0
[DH3] 270: then PORT[gCRTC+1]:=
[M53] 271: PORT[gCRTC+1] OR $40
[AK3] 272: else PORT[gCRTC+1]:=
[S04] 273: PORT[gCRTC+1] AND NOT $40;
[GM3] 274: dummy:=port[gCRTC+6];
[CE2] 275: PORT[$3C0]:=$10;
[FL4] 276: PORT[$3C0]:=PORT[$3C1] OR $20;
[FE2] 277: PORT[$3C0]:=$20;
[ZN0] 278: end;
[GH4] 279: procedure TVGA.ScrollLeft;
[MU0] 280: begin
[MK4] 281: SetView(succ(viewx),viewy)
[ZN0] 282: end;
[KV4] 283: procedure TVGA.ScrollRight;
[MU0] 284: begin
[6E2] 285: if viewx>0 then
[MU0] 286: begin
[MW1] 287: Dec(viewx);
[3R3] 288: SetView(viewx,viewy)
[ZN0] 289: end;
[ZN0] 290: end;
[5K4] 291: procedure TVGA.ScrollDown;
[MU0] 292: begin
[7E2] 293: if viewy>0 then
[MU0] 294: begin
[PW1] 295: Dec(viewy);
[3R3] 296: SetView(viewx,viewy)
[ZN0] 297: end;
[ZN0] 298: end;
[L84] 299: procedure TVGA.ScrollUp;
[MU0] 300: begin
[FF4] 301: SetView(viewx,succ(viewy))
[ZN0] 302: end;
[XY6] 303: function TVGA.VirtualScreenWidth:Word;
[MU0] 304: begin
[LG4] 305: VirtualScreenWidth:=width
[ZN0] 306: end;
[Y57] 307: function TVGA.VirtualScreenHeight:Word;
[MU0] 308: begin
[E05] 309: VirtualScreenHeight:=height
[ZN0] 310: end;
[286] 311: function TVGA.StartViewWidth:Word;
[MU0] 312: begin
[DV3] 313: StartViewWidth:=viewx
[ZN0] 314: end;
[3F6] 315: function TVGA.StartViewHeight:Word;
[MU0] 316: begin
[W34] 317: StartViewHeight:=viewy
[ZN0] 318: end;
[KB5] 319: procedure TVGA.Hide; ASSEMBLER;
[FN0] 320: asm
[PL1] 321: mov AH,12h
[ZL1] 322: mov AL,01h
[EM1] 323: mov BL,36h
[Y71] 324: int 10h
[ZN0] 325: end;

```



```
[4F5] 326: procedure TVGA.Show; ASSEMBLER;
[FN0] 327:   asm
[PL1] 328:     mov AH,12h
[XL1] 329:     mov AL,00h
[EM1] 330:     mov BL,36h
[Y71] 331:     int 10h
[ZN0] 332:   end;
[PU5] 333: procedure TVGA.Clear(x,y,w,h:Word);
[2P2] 334:   var i : Integer;
[MU0] 335:   begin
[L82] 336:     PORT[$3C4]:=2;
[6G2] 337:     PORT[$3C5]:=$F;
[MU3] 338:     for i:=y TO Pred(y+h) do
[364] 339:       FillChar(Mem[$A000:x DIV 4+
[Z04] 340:         i*width DIV 4],w DIV 4,0);
[ZN0] 341:   end;
[D97] 342: procedure TVGA.Copy(vx,vy,w,h,nx,ny:Word);
[2P2] 343:   var i : Integer;
[MU0] 344:   begin
[L82] 345:     PORT[$3C4]:=2;
[6G2] 346:     PORT[$3C5]:=$F;
[Z21] 347:     SetMode(0,1);
[CH2] 348:     for i:=0 to h do
[GN3] 349:       Move(Mem[$A000:vx DIV 4+
[H23] 350:         (i+vy)*width DIV 4],
[ZW2] 351:         Mem[$A000:nx DIV 4+
[923] 352:         (i+ny)*width DIV 4],
[Z71] 353:         w DIV 4);
[V21] 354:     SetMode(0,0);
[ZN0] 355:   end;
[RM6] 356: procedure TVGA.PCX(f:string; x,y:Word);
[L12] 357:   var q : file;
[844] 358:   b : Array[0..2047] OF Byte;
[0M4] 359:   anz,pos,c,w,h,e,pack: Word;
[MU0] 360:   begin
[0L3] 361:     Assign(q,f); Reset(q,1);
[RM3] 362:     BlockRead(q,b,128,anz);
[0K4] 363:     if (b[0]<>10) OR (b[3]<>8) then
[L04] 364:       begin Close(q); Exit end;
[HN5] 365:     w:=succ((b[9]-b[5])*256+b[8]-b[4]);
[BY5] 366:     h:=succ((b[11]-b[7])*256+b[10]-b[6]);
[Z63] 367:     pack:=0; c:=0; e:=y+h;
[P31] 368:     repeat
[2P3] 369:       BlockRead(q,b,2048,anz);
[981] 370:       pos:=0;
[095] 371:       while (pos<anz) AND (y<e) do begin
[LF1] 372:         if pack<>0
[ZN1] 373:           then begin
[N93] 374:             for c:=c to c+pack do
[LK3] 375:               SetPix(x+c,y,b[pos]);
[BV1] 376:             pack:=0; end
[7G4] 377:           else if (b[pos] AND $C0)=$C0
[BX3] 378:             then pack:=b[pos] AND $3F
[WP1] 379:             else begin
[LK3] 380:               SetPix(x+c,y,b[pos]);
[AP1] 381:               Inc(c) end;
[8H1] 382:             Inc(pos);
[HS1] 383:             if c=w then
[6G3] 384:               begin c:=0; Inc(y) end;
[ZN0] 385:             end;
[YK3] 386:           until (anz=0) OR (y=e);
[A44] 387:           Seek(q,FileSize(q)-3*256-1);
[5G3] 388:           BlockRead(q,b,3*256+1);
[M73] 389:           if b[0]=12 then begin
[393] 390:             for x:=1 to 3*256+1 do
[8T2] 391:               b[x]:=b[x] SHR 2;
[Z03] 392:             SetPal(0,256,b[1]);
[ZN0] 393:           end;
[YE1] 394:           close(q);
[ZN0] 395:         end;
[SP6] 396: procedure TVGA.WaitDisplay; ASSEMBLER;
[BR2] 397:   label WD_R,WD_D;
[FN0] 398:   asm
[DC2] 399:     mov DX,gCRTC+6
[WX0] 400:     WD_R:
[161] 401:     in AL,DX
[KN1] 402:     test AL,8d
[6B1] 403:     jz WD_R
[EX0] 404:     WD_D:
[161] 405:     in AL,DX
[KN1] 406:     test AL,8d
[0K1] 407:     jnz WD_D
[ZN0] 408:   end;
[6N6] 409: procedure TVGA.WaitRetrace; ASSEMBLER;
[BR2] 410:   label WD_R,WD_D;
[FN0] 411:   asm
[DC2] 412:     mov DX,gCRTC+6
[WX0] 413:     WD_R:
[161] 414:     in AL,DX
[KN1] 415:     test AL,8d
[EK1] 416:     jnz WD_R
[EX0] 417:     WD_D:
[161] 418:     in AL,DX
[KN1] 419:     test AL,8d
[E91] 420:     jz WD_D
[ZN0] 421:   end;
[RH5] 422: procedure TVGA.SetMode(r,w:Byte);
```

```
[PT1] 423: ASSEMBLER;
[FN0] 424:   asm
[VA1] 425:     mov AH,r
[7B1] 426:     and AH,1h
[981] 427:     shl AH,1
[981] 428:     shl AH,1
[981] 429:     shl AH,1
[CB1] 430:     mov AL,w
[NB1] 431:     and AL,3h
[1B1] 432:     add AH,AL
[T71] 433:     or AH,40h
[UZ1] 434:     mov DX,03CEh
[UE1] 435:     mov AL,5h
[YH1] 436:     out DX,AX
[ZN0] 437:   end;
[XL3] 438: Destructor TVGA.Done;
[MU0] 439:   begin
[FN0] 440:     asm
[PC1] 441:       push AX
[0F1] 442:       xor AH,AH
[SE1] 443:       mov AL,3h
[Y71] 444:       int 10h
[R31] 445:       pop AX
[ZN0] 446:     end;
[ZN0] 447:   end;
[KN0] 448: end.
```

Listing 1. Die mit Assembler-Routinen optimierte Turbo-Pascal-Unit »vga.fx.pas«

```
[8F3] 1: {Programm: demo.pas
[HV6] 2: Funktion: Demonstration der Funktionen
[BL3] 3:   der Grafik-Unit VGAfx
[G64] 4: Sprache: Turbo Pascal 6.0
[K34] 5: Autor: Frank Höppner
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[492] 8: program Demo;
[WN2] 9: uses VGAfx,Crt;
[813] 10: var i,j : integer;
[1E2] 11:   p : TPalette;
[Z36] 12: { im aktuellen Pfad müssen sich die
[AL5] 13: vier PCX Bilder FILE1 bis FILE4 be-
[LZ5] 14: finden, die alle die gleiche Palette
[ZV6] 15: benutzen sollten (notfalls umfärben) }
[MU0] 16: begin
[W42] 17:   vga.Init(640);
[9F1] 18:   vga.Hide;
[VR3] 19:   vga.PCX('FILE4.PCX',0,0);
[UH1] 20:   vga.Show;
[L63] 21:   for j:=10 downto 1 do
[MU0] 22:     begin
[VY4] 23:       for i:=1 TO j do vga.ScrollUp;
[8B5] 24:       for i:=1 TO j do vga.ScrollLeft;
[FC5] 25:       for i:=1 TO j do vga.ScrollDown;
[SE5] 26:       for i:=1 TO j do vga.ScrollRight;
[ZN0] 27:     end;
[NN1] 28:     Delay(1000);
[8K2] 29:     for i:=1 to 64 do
[9H4] 30:       vga.SetCol(252,64-i,64-i,64-i);
[NN1] 31:     Delay(1000);
[A74] 32:     vga.PCX('FILE1.PCX',320,200);
[PT2] 33:     for i:=1 to 320 do
[0Z4] 34:       vga.SetView(i,trunc(i/320*200));
[S24] 35:       vga.PCX('FILE3.PCX',0,200);
[134] 36:       vga.PCX('FILE2.PCX',320,0);
[BN5] 37:       FOR i:=320 downto 0 do vga.SetView(i,
[2V4] 38:         200+trunc(160*sin((i-320)/320*pi)));
[TP5] 39:       for i:=1 to 199 do vga.SetSplitt(i);
[PR2] 40:       vga.SetView(0,0);
[6U2] 41:       vga.SetSplitt(0);
[ND5] 42:       for i:=1 to 319 do vga.ScrollLeft;
[Z33] 43:       vga.SetView(320,0);
[UM5] 44:       for i:=1 to 100 do vga.SetSplitt(i);
[4P5] 45:       for i:=319 downto 0 do vga.SetView(i,
[1C4] 46:         -trunc(160*sin((i-320)/320*pi)));
[HT5] 47:       for i:=100 to 199 do vga.SetSplitt(i);
[PR2] 48:       vga.SetView(0,0);
[6U2] 49:       vga.SetSplitt(0);
[8K2] 50:       for i:=1 to 64 do
[9H4] 51:         vga.SetCol(252,64-i,64-i,64-i);
[NN1] 52:       Delay(1000);
[BS2] 53:       for i:=30 to 99 do
[MU0] 54:         begin
[NY2] 55:           vga.Clear(0,i,320,1);
[LF3] 56:           vga.Clear(0,199-i,320,1);
[DF1] 57:           Delay(20);
[ZN0] 58:         end;
[VG1] 59:       vga.Done;
[KN0] 60:     end.
```

Listing 2. »demo.pas« nutzt die Leistungen Ihrer VGA-Karte.

Animation

Den Bildern Beine gemacht

Computeranimation ist die Steigerungsform von Computergrafik, die an sich schon eines der interessantesten Themen der Computertechnik ist. Für eigene Animationen von der Ganzseiten-Animation bis zu 256-Farben-Sprites vermittelt Ihnen dieser Beitrag die Grundlagen.

Computeranimationen begegnen uns täglich, wenn wir den Fernseher einschalten. Kaum ein Sender ohne aufwendiges dreidimensionales Logo, dem nicht durch leistungsfähige Computer Leben eingehaucht wurde. Mittlerweile besitzt sogar jedes Magazin, das etwas auf sich hält, neben einer Erkennungsmelodie auch seine eigene anspruchsvolle und aufwendige Erkennungsanimation. Das Herz jedes grafikbegeisterten Computerbesitzers schlägt bei solchen gelungenen Animationen höher. Ein nüchterner Blick auf die eigene Hardware-Ausstattung läßt jede Euphorie allerdings schnell wieder verfliegen. Wenn Supercomputer für Fernsehlogos Wochen oder Monate rechnen, wie lange soll dann bitte schön Ihr PC rechnen? Dennoch lassen sich – das wissen Sie spätestens seit Autodesk Animator – auch auf dem PC anspruchsvolle Animationen erzeugen. Ebenso eindrucksvoll zeigen viele Spiele, was grafisch aus einer VGA-Grafikkarte herauszuholen ist.

● **Ganzseiten-Animation:** Am schwierigsten sind sicherlich die Animationen zu erzeugen, die sich über den gesamten Bildschirm erstrecken. Sie vermitteln am ehesten Fernseh- oder Kinogeühle. Gleichzeitig stoßen Sie bei diesen Animationen jedoch am schnellsten an die Grenzen der Leistungsfähigkeit Ihres Rechners. Grundsätzliches Problem aller Computeranimationen ist die Zeit. Für einen flüssigen Bewegungsablauf sind theoretisch wenigstens 24 Bilder pro Sekunde auf den Monitor zu bringen; nur dann nimmt das menschliche Auge die kleinen ruckhaften Bewegungen nicht mehr als solche wahr. Im VGA-Modus mit 320x200 Punkten und 256 Farben benötigt ein Bild etwa 64 KByte Speicher. Bei 24 Bildern in der Sekunde kommen Sie dann auf gut 1,5 MByte an Bilddaten für eine Sekunde Animation. Auf einer 40-MByte-Festplatte ließe sich ein 26 Sekunden langer Film speichern. Für die Darstellung müßten dann 40 MByte in 26 Sekunden von der Festplatte gelesen werden. Daran scheitern bereits viele Festplatten.

Ganzseiten-Animation kostet Speicher

Schrauben Sie Ihre Anforderungen jedoch etwas herunter, läßt sich dennoch einiges mit dieser VGA-Grafikkarte anfangen. Bei vielen Animationen werden Objekte betrachtet, die sich um die eigene Achse drehen. Das sieht nicht nur gefällig aus, sondern hat bei symmetrischen Körpern auch einen praktischen Effekt: Nach einem deutlich geringeren Drehwinkel als 360 Grad (das heißt der vollen Umdrehung) wiederholen sich die Bilder. Dann genügen wenige Bilder zur

Darstellung des gesamten Drehvorgangs. Lassen sich zusätzlich alle Bilder gleichzeitig im Hauptspeicher unterbringen, so steht einer flüssigen Animation nichts mehr im Weg. Es genügt jedoch nicht, die Bilder einfach nacheinander in den Videospeicher zu kopieren, um dann rasch ein neues Bild anzuzeigen. Denn leider muß für das Kopieren aus dem Hauptspeicher in den Videospeicher der »Flaschenhals« Hauptprozessor benutzt werden. Aus dem Beitrag zur Programmierung der VGA-Grafikkarte (»Trickkiste für VGA-Karten«) wissen Sie bereits, daß die VGA-Grafikkarte intern im Doppelwortmodus arbeiten kann, so daß mit einem Schritt statt einem oder zwei Byte gleich vier Byte (zwei Worte, ein Doppelwort) kopiert werden. Vorteilhaft ist es also, wenn sich zu Beginn der Animation bereits alle Bilddaten im Videospeicher befinden. Dies ist sicher nur sehr selten der Fall, da dem Videospeicher noch engere Grenzen (256 KByte) gesetzt sind als dem Hauptspeicher. Unabhängig davon, ob die neuen Bilddaten im Video- oder im Hauptspeicher liegen, beim Kopieren bleibt immer eine Zeitverzögerung vom Setzen des ersten bis zum letzten Bildpunkt. Diese Zeitverzögerung ist je nach Computer und Grafikkarte unterschiedlich, wirkt sich aber auch bei minimalen Werten störend aus: Die Bewegungen sehen nicht flüssig aus, der untere Bildbereich hinkt dem oberen Bereich stets hinterher.

● **Mehrere Bildschirmseiten:** Um dieses Manko auszugleichen, verwenden Sie am besten zwei Bildschirmseiten.

Mehr Geschwindigkeit mit mehr Seiten

Mit der Unit »vgafx.pas« stehen Ihnen sogar vier Seiten zur Verfügung. Beschreiben Sie einfach eine Bildschirmseite mit den neuen Bilddaten, während Sie die andere Seite anzeigen. Sind Sie mit dem Aufbau fertig, schalten Sie mit dem Setview-Befehl auf die gerade erzeugte Seite und überschreiben die eben noch angezeigte Seite mit neuen Bilddaten. Da das Umschalten auf eine andere Grafikkarte durch die VGA-Grafikkarte sehr schnell erfolgt, nehmen Sie den Wechsel mit einem Schlag wahr – ohne jede Verzögerung. Welche Geschwindigkeit Ihre Animation erreicht, hängt nur noch davon ab, wie schnell Sie in der Lage sind, das jeweils neue Bild aufzubauen. In den meisten Fällen bietet es sich an, die Einzelbilder vorher zu berechnen und zu speichern, so daß sie während der Animation nur noch von der Festplatte geladen werden müssen.

● **Delta-Packing:** Wenn es sich um eine feine, also eine aus vielen Einzelbildern bestehende Animation handelt, ist es sinnvoller, nur die Änderungen zum vorherigen Einzelbild zu ermitteln. Im ersten Durchgang erzeugen und speichern Sie alle Einzelbilder, im zweiten vergleichen Sie jedes Bild mit seinem Vorgänger. Dabei ermitteln Sie die Bereiche, die auf beiden Einzelbildern identisch sind. Diese Bereiche



Bild 1. Der Entwurf einer Spielfigur

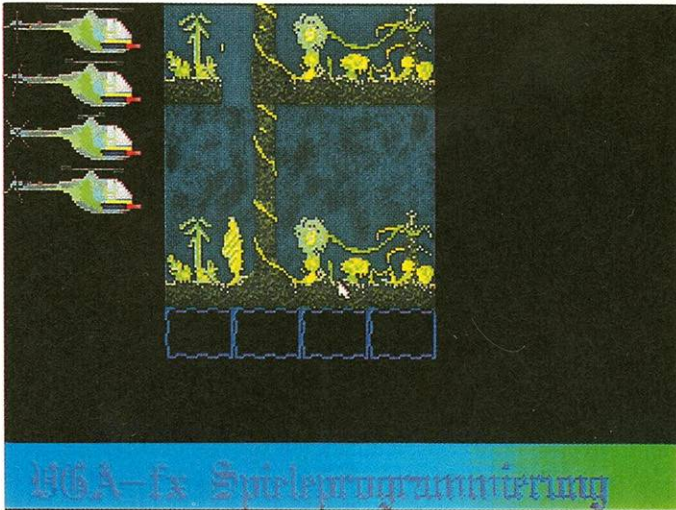


Bild 2. Spielfigur und Hintergrund in der Entwicklungsphase

brauchen Sie nicht zu speichern, es reicht, wenn Sie bei der Animation nur veränderte Ausschnitte erneuern – diese Technik heißt Delta-Packing. Damit vermindert sich die Datenmenge, so daß ein weiterer Zeitgewinn beim Lesen des Teilbilds von der Festplatte hinzukommt.

• **Sprites:** Jedes Action-Spiel auf dem PC erlaubt dem Anwender, durch Steuerbefehle seine Spielfigur auf dem Bildschirm zu bewegen. Hier liegt das Problem völlig anders: Ein kleines Bild, die Spielfigur, wird über einen beliebigen Hintergrund bewegt, so wie der Anwender es wünscht. Eine solche Animation läßt sich natürlich nicht im voraus berechnen. Dafür haben Sie es jedoch mit kleineren Bildbereichen zu tun, so daß sie den Umsatz an Bilddaten auch bewältigen können. Außerdem bestehen die animierten Spielfiguren aus einer geringen Anzahl von Einzelbildern, die Sie ohne weiteres gleichzeitig im Speicher unterbringen.

• **Zeichnen der Phasen:** Die Entwicklung einer solchen animierten Spielfigur beginnt grundsätzlich auf dem Zeichenbrett beziehungsweise im Malprogramm. Für erste eigene Experimente eignet sich das Sharewareprogramm »Improc« hervorragend. Es erlaubt eine punktgenaue Bearbeitung von 256-Farben GIF-Bildern. Zunächst legen Sie am besten die Farben für Ihre Spielfigur fest (Palettendefinition). Es empfiehlt sich, einen festen Bereich für die Farben der Spielfigur abzustecken, beispielsweise Farbe 100 bis 150, damit Sie bei späteren Palettenänderungen nicht unabsichtlich andere Figuren oder Bildteile umfärben. Die 256 Farben der VGA-Grafikkarte lassen hier viel Spielraum. Anschließend bearbeiten Sie Ihre Figur in jeder einzelnen Phase der Animation im Lupen- oder auch Fat-Bit-Modus, damit Sie trotz kleiner Figuren auf jeden Bildpunkt direkten Zugriff haben (Bild 1).

Animation in mehreren Phasen

Bild 2 zeigt einen kleinen Hubschrauber, der in vier Animationsphasen am linken oberen Rand des PCX-Bilds gezeichnet ist. In jeder Phase haben sich Rotor und Stabilisator ein Stück weitergedreht. Außerdem blinken zwei Lampen abwechselnd rot und grün. Nachdem das Bild durch »vga.PCX('HUBSCHR.PCX',0,200)« in den Videospeicher geladen ist, greifen Sie durch den Copy-Befehl der Unit »vga.fx« auf jede Phase direkt zu. Da beim Copy-Befehl intern vier Byte gleichzeitig kopiert werden und die einzelnen Animations-Einzelbilder sehr klein sind, erreichen Sie damit gute bis sehr gute Ergebnisse. Schwierigkeiten bekommen Sie jedoch dadurch, daß der Copy-Befehl nur an horizontalen Positionen korrekt arbeitet, die durch 4 teilbar sind. Wenn die Qualität des Bildaufbaus beim Copy-Befehl

BUNDESLIGA TOTAL



Alles aus 29 Jahren
Bundesliga-Fußball
Jetzt auf Ihrem Computer

aktuell – komplett – erweiterbar

Die Datenbank für jeden
Fußballfreund

Sie kennen sich aus in Sachen Bundesliga? Dann müssen Sie diese unvergleichliche Datenbank haben. Denn »Bundesliga Total« macht Sie zum unschlagbaren Experten. »Bundesliga Total« ist das Programm, mit dem Sie selbst erfahrene Sportreporter verblüffen werden. Seit 1963 gibt es die Bundesliga – dieses Programm kennt alle Fakten und Ergebnisse seit dem Bestehen!

Sie sitzen zum Beispiel am Radio oder Fernseher und verfolgen das aktuelle Geschehen – blitzschnell gibt Ihnen »Bundesliga Total« die richtigen Auskünfte: Wann hat Bayern am höchsten gegen Nürnberg verloren? Wann spielte zuletzt Schalke gegen Gladbach? Wie war der beste Tabellenstand für Hertha BSC? Wie sieht die Tagerstabelle aus? Wie steht die Heim-/Auswärts-Bilanz? Welchen Rang hat der HSV in der »Ewigen Liste«? Ob Fragen nach Punktestand, nach Toren, Siegen, Unentschieden, Niederlagen, Tordifferenzen, ob grafische Übersichten, Saisonprofil o.v.m. – »Bundesliga Total« weiß alles. Insgesamt 63 verschiedene Tabellen pro Spieltag und mehr als 3600 Resultate sind erfaßt. Hier bleiben keine Fragen offen – bis auf die zukünftigen. Sie brauchen aber nur die kommenden Fußballergebnisse einzutragen, und Sie sind wöchentlich auf dem allerneuesten Stand!

Die ideale Ergänzung für Ihren eigenen Verein:

SPORT-TABELLEN-MANAGER

Der Tabellen-Manager
für Liga-Spiele

Das Programm, mit dem Sie Ihren eigenen Verein in dessen Liga verwalten können. Natürlich ist er auch für alle anderen Sportarten geeignet, die nach dem 0-1-2-Punktesystem bewertet werden. Dargestellt in Tabellen, Grafiken und Statistiken. Laut Praxistest in Sportredaktionen für die professionelle Tabellenarbeit bestens geeignet.

Zwei Programme zum Paketpreis

Bundesliga Total + Sport-Tabellen-Manager

Ein doppeltes Muß für jeden Sportsmann, der über einen PC mit Festplatte und EGA- oder VGA-Grafik verfügt

DM **69,-***

Terminplaner

DM 29,80*

* Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland DM 4,- bzw. für das Ausland DM 6,- Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.

auch akzeptabel ist, so ist eine flüssige Animation ausgeschlossen.

● **Kombination mit dem Hintergrund:** Bevor Sie nun in den Tiefen der VGA-Register nach einer Lösung suchen, überlegen Sie folgendes: Wenn sich der Hubschrauber vor beliebigem Hintergrund bewegt, überschreiben Sie stets einen Teil der Hubschrauberumgebung mit Schwarz, denn mit dem Copy-Befehl lassen sich nur rechteckige Ausschnitte kopieren (und welche Spielfigur ist schon rechteckig?). Um hier dennoch zu brauchbaren Ergebnissen zu kommen, stellt Ihnen die Unit »spritefx.pas« (Listing 1) das notwendige Werkzeug zur Verfügung. Der hier definierte Objekttyp »TSprite« beherbergt alle relevanten Assembler-Routinen. Bevor Sie ein Sprite benutzen, initialisieren Sie es durch den Konstruktor-Aufruf »Init(w,h)«. Dabei bedeuten »w« und »h« die Breite beziehungsweise Höhe des Sprites, also Ihrer Spielfigur. Der Hubschrauber besteht aus vier Phasen, also initialisieren Sie vier Variablen des Typs »TSprite« mit den Maßen des kleinsten Rechtecks, in das der Hubschrauber in jeder Animationsphase hineinpaßt (»demo.pas«, Listing 2). Die Init-Methode reserviert nun dynamisch einen Speicherbereich, in den ein Sprite der angegebenen Größe hineinpaßt. Diesen Speicherbereich füllen Sie mit der Methode »GrabSprite(x,y)«. Sie kopiert – beginnend bei den Koordinaten x,y – einen Bildbereich der vordefinierten Größe in den internen Speicherbereich. Es handelt sich bei dieser Methode einfach um einen wiederholten Aufruf der Funktion »GetPix« aus der Unit »vgafox«. Die Funktionsergebnisse, also die Farbinformationen, schreiben Sie direkt in den reservierten Speicherbereich – so haben Sie eine Kopie des Sprites. »GetPix« errechnet bei jedem Aufruf die Bildschirmadresse, unter der sich der gewünschte Bildpunkt befindet; soviel Aufwand ist nicht notwendig. Schließlich lesen Sie meist direkt benachbarte Punkte aus, deren Adresse sich inkremental aus der vorherigen ergibt: Wenn die x-Position durch 4 teilbar ist, addieren Sie 1 zur Adresse, sonst nicht. Das Read-Map-Select-Register hingegen wird stets um 1 erhöht und logisch mit 3 »Und-verknüpft«. So ergibt sich jede horizontal benachbarte Adresse aus der vorherigen. Sind Sie bei einem Zeilensprung angelangt, das heißt, das Sprite ist für diese Zeile zu Ende, dann addieren Sie zur aktuellen Adresse einfach die virtuelle Bildschirmbreite abzüglich der Spritebreite dazu. Daraus ergibt sich die Adresse des ersten Spritepunkts der nächsten Bildschirmzeile. Bei diesem Verfahren verwenden Sie nur Inkrement- und Additionsbefehle. In der Getpix-Methode kommt jedoch eine Multiplikation mit der virtuellen Bildbreite zur Anwendung. Eine Multiplikation benötigt in der Ausführung natürlich wesentlich mehr Taktzyklen als Inkrement- oder Additionsbefehle. Aufgrund des Zeitvorteils ist damit dem beschriebenen Verfahren der Vorzug zu geben. Der Quelltext wird dadurch zwar länger, jedoch ergibt sich gleichzeitig ein Geschwindigkeitszuwachs – und ohne hohe Geschwindigkeit keine gute Animation. Die Computeranimation ist offensichtlich einer der wenigen vom Aussterben bedrohten Bereiche, bei denen noch komplizierte Programmierung der übersichtlichen Programmierung vorzuziehen ist, solange damit Zeitvorteile erreicht werden.

Nachdem Grabsprite die Animationsphasen eingefangen hat, benötigen Sie die Originalvorlagen aus dem PCX-Bild nicht mehr. Sie können den Videospeicher nun anderweitig nutzen.

Das Gegenstück zu Grabsprite ist die Methode »DropSprite(x,y)«. Sie schreibt das Sprite exakt so, wie es von »GrabSprite« gelesen wurde, wieder in den Videospeicher. Sie ist parallel zur Grabsprite-Methode aufgebaut, nur muß das Map-Mask-Register anders gesetzt werden als das Read-Map-Select-Register (siehe Unit »vgafox«). Die inkrementale Umschreibung des wiederholten Schreibens der Spritepunkte übernehmen Sie jedoch direkt aus der Grabsprite-Methode. Bis hier stellt das TSprite-Objekt eine Erweiterung des Copy-Befehls aus Vgafox dar. Die eigentliche Arbeit lei-

stet jedoch die Methode »ToggleSprite«. Sie schreibt die Bilddaten in den Videospeicher und ersetzt gleichzeitig die Bilddaten durch die Daten des Videospeichers. Es werden also Spritedaten und Videospeicherdaten vertauscht. Der Vorteil dieses Verfahrens liegt darin, daß beim zweimaligen Aufruf von Togglesprite das ursprüngliche Bild wieder hergestellt ist. Das vereinfacht die Handhabung und die Speicherverwaltung. Außerdem schreibt Togglesprite nur dann in den Videospeicher, wenn der jeweilige Punkt einen Farbcode ungleich 0 hat. Wenn also Ihre unregelmäßige Spielfigur, welche Sie mit »GrabSprite« einlesen, mit Punkten der Farbe 0 zu einem Rechteck aufgefüllt ist, so wird beim Aufruf von »ToggleSprite« ausschließlich die Figur in den Videospeicher geschrieben. So bewegen Sie jedes noch so seltsam geformte Sprite sauber über einen Bildhintergrund, ohne diesen dabei zu zerstören.

Der Hintergrund darf nicht zerstört werden

Statt des zweiten Aufrufs von Togglesprite mit denselben Koordinaten, um den alten Schirminhalt wieder zu restaurieren, rufen Sie nun »SecondToggle« auf. Diese Methode ruft ihrerseits Togglesprite mit den letzten von Ihnen verwendeten Werten auf. So brauchen Sie sich die jeweils zuletzt benutzten Koordinaten nicht zu merken.

Die Togglesprite-Methode ist etwas komplizierter als die Grabsprite- oder Dropsprite-Methode. Nachdem Sie – inkremental – die Bildschirmadresse errechnet haben, lesen Sie ein Farbbyte aus dem Spritespeicher aus. Wenn es ein Nullbyte ist, hat sich die Arbeit für diese Adresse schon erledigt. In jedem anderen Fall stellt das Byte einen Teil der Spielfigur dar. Sie müssen es also in den Videospeicher schreiben. Dabei soll jedoch der Inhalt des Videospeichers nicht überschrieben, sondern für die spätere Restauration gesichert werden. Sie merken sich also das Byte, das Sie schreiben wollen, lesen es jedoch vorher aus dem Videospeicher aus und schreiben es an die Stelle, an der ursprünglich das Spritebyte stand. Jetzt erst kommt das Spritebyte in den Videospeicher. So haben Sie Sprite- und Video-byte direkt ausgetauscht. Wenn Sie die Methode ein zweites Mal aufrufen, schreiben Sie ein Sprite, das genau dem Hintergrund entspricht, und erhalten Ihr Sprite wieder.

Des weiteren finden Sie in Spritefx die Methoden »LoadSprite« und »SaveSprite«, mit denen Sie einmal eingefangene Sprites direkt speichern und wieder laden, ohne jedesmal ein ganzes PCX-Bild zu bemühen und damit den Bildschirminhalt zu verändern. Allerdings erwarten die beiden Methoden, daß Sie das Sprite vorher initialisiert haben (Init-Aufruf). Die letzte Prozedur »Done« gibt den reservier-

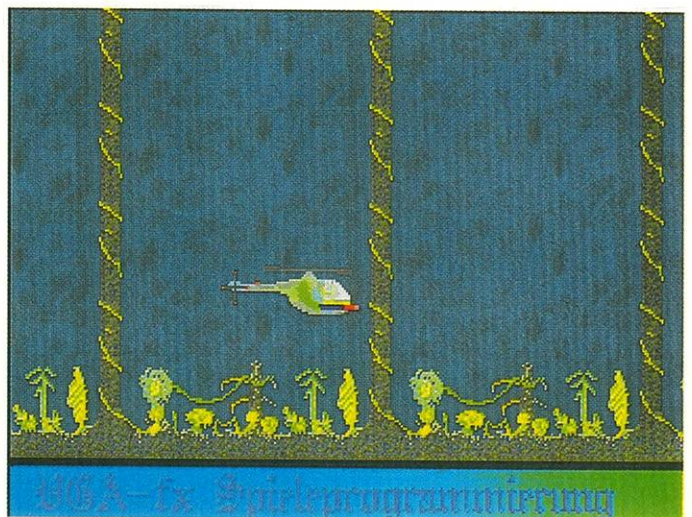


Bild 3. Eine Animation aus den Elementen von Bild 2

ten Sprite-Speicher wieder frei, muß also vor Programmende aufgerufen werden.

Das Timing-Problem

● **Verschiedene Methoden der Sprite-Programmierung:** Mit der Unit »vgafx« stellt die Unit »Spritefx« ein ausgezeichnetes Werkzeug für die Spieleprogrammierung dar. Bewegte Spielfiguren oder Raumschiffe, die sich über scrollenden Hintergrund bewegen, sind damit (fast) eine Kleinigkeit. Das einzige Problem, welches sich noch stellt, ist das Geschwindigkeitsproblem. Denn der Komfort von Toggle-Sprite hat natürlich seinen Preis. Die meisten Sprites, die Sie mit ToggleSprite über den Schirm ziehen lassen, flackern ein wenig, da der Aufbau eben nicht innerhalb eines Darstellungszyklus abgewickelt ist. Hier zeigt sich wieder das Problem der Zeitverzögerung vom Schreiben des ersten bis zum letzten Spritepunkt. Die Lösung dieses Problems ist abhängig von der Anwendung, die Sie programmieren. Läßt sich der jeweilige Bildhintergrund rasch aus wenigen Bildkomponenten generieren oder ist der Bildhintergrund nur etwa zwei Standard-Bildschirmseiten groß, so bietet sich wieder das Verfahren der zwei Bildschirmseiten an, die abwechselnd angezeigt und verändert werden (double buffering). Sind die Sprites so klein, daß sie mit dem Copy-Befehl aus »vgafx« flackerfrei kopiert werden können, stellt sich Ihnen folgende Alternative: Sie halten einfach einen kleinen Bereich des virtuellen Schirms frei, kopieren dorthin die Umgebung des Hintergrunds, in den Sie das Sprite setzen wollen, schreiben das Sprite in die Kopie und kopieren den modifizierten Hintergrund schließlich zurück. Bei diesem Verfahren bleiben nahezu alle vier Bildschirmseiten für einen extrem großen Hintergrund frei.

In den meisten Fällen ist der Spielhintergrund jedoch aus sich wiederholenden Mustern zusammengesetzt. Der tatsächliche Bildschirm ergibt sich dann aus einer Kombination dieser kleinen Bildelemente. Dieses Verfahren zeigt »demo.pas« schematisch. In Bild 2 ist neben den vier Animationsphasen des Hubschraubers auch ein Ausschnitt eines Bildhintergrunds enthalten. Aus ihm setzt sich der gesamte Hintergrund aus Bild 3 zusammen, über den dann der Hubschrauber fliegt. Im virtuellen Bildschirmspeicher legen Sie zwei Bildschirme ab, die etwas größer sind als der tatsächlich angezeigte Schirm. Innerhalb dieser Überlappungen rollen Sie mit den Vgafx-Routinen. Sobald Sie jedoch eine Überlappungsgrenze überschreiten, müssen Sie beide Bildschirme erneut aufbauen. In »demo.pas« erfolgt dies sehr simpel, indem der gesamte Bildinhalt einfach ein paar Punkte zurückkopiert und die Anzeige entsprechend zurückgesetzt wird. Wesentlich eleganter ist es, wenn dies nicht auf einen Schlag erfolgt, sondern bei jeder Animationsphase nur zu einem kleinen Teil. Damit verhindern Sie, daß beim Erreichen einer Überlappungsgrenze eine kleine Verzögerung eintritt, bis der gesamte Bereich kopiert wurde. Zur Not variieren Sie wie in »demo.pas« die Verzögerungsschleifen so lange, bis sie genau der Zeit entsprechen, die für die ganze Kopieraktion benötigt wird. Auch hier sind Sie mit ausgeklügelten, aufwendigen Verfahren oft besser beraten als mit einer überschaubaren, einfachen Methode, bei der dann das Timing nicht funktioniert.

Am schnellsten geht's mit Xor

Die letzte Methode, die sich auf dem Weg zur sauberen Animation anbietet, liefert das Data-Rotate-Register (Index drei, Adressen 3CEhex und 3CFhex). Wenn Sie nicht unbedingt auf alle 256 Farben angewiesen sind, haben Sie hiermit gleichzeitig die raffinierteste Methode zur Animation von Spielfiguren. Die ersten drei Bits erlauben es, ein vom Prozessor kommendes Byte beliebig bitweise nach rechts zu ro-

tieren. Damit setzen Sie die Einschränkungen des Copy-Befehls aus »vgafx« außer Kraft. Danach wird das Byte mit der Methode, die Sie in den Bits 3 und 4 angeben, mit dem Hintergrund verknüpft. Wählen Sie hier für beide Bits die 1, so erreichen Sie eine Xor-Verknüpfung mit dem Hintergrund. Xor bedeutet »Exclusive-Or« (Entweder-Oder). Der praktische Nutzen der Xor-Verknüpfung ist sehr groß: Ziehen Sie eine Linie beliebiger Farbe zweimal hintereinander über einen Hintergrund, so erhalten Sie wieder den ursprünglichen Hintergrund. Das liegt daran, daß die Xor-Verknüpfung den ursprünglichen Wert nicht überschreibt, sondern nur verändert. Ein Beispiel:

```
10010110 Hintergrund
01100101 Linienfarbe
```

```
11110011 Xor
01100101 Linienfarbe
```

```
10010110 Xor
```

Nach dem ersten Exklusiv-Oder hat der Bildschirm eine andere Farbe angenommen. Sie sehen nicht die Linienfarbe, aber Sie sehen, daß dort eine Linie verläuft. Nach dem zweiten Exklusiv-Oder mit der Linienfarbe erhalten Sie wieder die Daten für den Bildhintergrund. Wenn jedoch eine farbliche Verfälschung der Spielfigur nicht in Frage kommt, so greifen Sie zu einem Trick: Sie benutzen für den Hintergrund nur die letzten vier Farbbits, für die Spielfigur die ersten vier Bits. Für sich allein stellen diese die Grundfarben des Programms dar. Alle übrigen Farbwerte wählen Sie so, daß sie Exklusiv-Oder-verknüpft die Farbe der Spielfigur entsprechend den ersten vier Bits ergeben. Nun sehen Sie nach der ersten Exklusiv-Oder-Verknüpfung mit dem Hintergrund die Spielfigur; nach einer weiteren Exklusiv-Oder-Verknüpfung wieder den Hintergrund, da alle ersten vier Farbbits zurückgesetzt sind. Sie wälzen also die ganze Arbeit auf die VGA-Grafikkarte ab. Das Schreiben der Spielfiguren übernimmt dann die normale Copy-Prozedur. Den Rest erledigt die Grafikkarte. Einziger Nachteil der Methode: Sie reduzieren die maximale Anzahl der Farben auf 32, jeweils vier Bits, also 16 Farben, für Figur und Hintergrund. Viele sehr gute kommerzielle Spiele erzielen jedoch auch mit 32 Farben hervorragende Ergebnisse. Für das Maß aller Dinge, die Geschwindigkeit, ist eben kein Preis zu hoch.

(Frank Höppner/bm)

```
[M84] 1: {Programm: spritefx.pas
[WC3] 2: Funktion: Routinen zur
[RT3] 3: Sprite-Programmierung
[G64] 4: Sprache: Turbo Pascal 6.0
[X34] 5: Autor: Frank Höppner
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG
[000] 7:
[4P6] 8: { Object-Typ TSprite zur Realisierung
[LF6] 9: von Vgafx-Sprites (mit Unit Vgafx) }
[DR7] 10: { Registerkompatibilität Voraussetzung! }
[000] 11:
[0H2] 12: unit Spritefx;
[SK1] 13: interface
[912] 14: uses Vgafx;
[ST0] 15: type
[4H6] 16: TByteArray = Array[0..MaxInt] of Byte;
[BD4] 17: PByteArray = TByteArray;
[GS2] 18: TSprite = object
[102] 19: lineoffset,
[TD3] 20: scrwidth,scrheight,
[XZ2] 21: posx,posy : Word;
[E73] 22: width,height : Byte;
[9N3] 23: content : PByteArray;
[7V4] 24: constructor Init(w,h:Word);
[SD5] 25: procedure GrabSprite(x,y:Word);
[KF5] 26: procedure DropSprite(x,y:Word);
[UE5] 27: procedure LoadSprite(f:string);
[NG5] 28: procedure SaveSprite(f:string);
[DR5] 29: procedure ToggleSprite(x,y:Word);
[664] 30: procedure SecondToggle;
[9T2] 31: destructor Done;
[ZN0] 32: end;
[4L2] 33: implementation
[AB6] 34: constructor TSprite.Init(w,h:Word);
[MU0] 35: begin
```

Listing 1. Mit »spritefx.pas« bringen Sie in Ihre VGA-Bilder.


```
[ØZ2] 36: if w>100 then w:=100;
[NW2] 37: if h>100 then h:=100;
[K83] 38: width:=w; height:=h;
[GS2] 39: posx:=0; posy:=0;
[YY4] 40: GetMem(content,width*height);
[9T5] 41: FillChar(content^,width*height,0);
[LZ5] 42: scrwidth:=vga.VirtualScreenWidth;
[Ø86] 43: scrheight:=vga.VirtualScreenHeight;
[DX4] 44: lineoffset:=scrwidth-width;
[ZNØ] 45: end;
[XV6] 46: procedure TSprite.GrabSprite(x,y:Word);
[PT1] 47: assembler;
[K64] 48: label xloop,yloop,noinc;
[FNØ] 49: asm
[ULØ] 50: cld
[BC1] 51: push DS
[GC1] 52: push SI
[KC1] 53: mov AX,y
[NC1] 54: mov CX,x
[CL3] 55: mov BL,DS:[DI+height]
[CG3] 56: mov BH,DS:[DI+width]
[7D4] 57: mov DX,DS:[DI+lineoffset]
[VF4] 58: shr DX,1 { lineoffset/4 }
[RA1] 59: shr DX,1
[ZC1] 60: push DX
[TC1] 61: push BX
[A94] 62: mov BX,DS:[DI+scrwidth]
[NU3] 63: les DI,DS:[DI+content]
[LA1] 64: shr BX,1
[LA1] 65: shr BX,1
[N31] 66: mul BX
[EH1] 67: mov BX,CX
[LA1] 68: shr BX,1
[LA1] 69: shr BX,1
[AD1] 70: add AX,BX
[5H4] 71: mov SI,AX { SI Screenpos. }
[T31] 72: pop BX
[X31] 73: pop DX
[ØS3] 74: mov AX,gScreenSegment
[285] 75: mov DS,AX { DS:SI Screenpos. }
[F31] 76: yloop:
[WC1] 77: push CX
[ZC1] 78: push DX
[TC1] 79: push BX
[UZ1] 80: mov DX,Ø3CEh
[TE1] 81: mov AL,4h
[D31] 82: xloop:
[4F1] 83: mov AH,CL
[9B1] 84: and AH,3h
[YH1] 85: out DX,AX
[AF3] 86: movsb { Transfer }
[UC1] 87: cmp AH,3h
[4B1] 88: jz noinc
[SS2] 89: dec SI { div 4 }
[F11] 90: noinc:
[P11] 91: inc CX
[ZVØ] 92: dec BH
[5P1] 93: jnz xloop
[T31] 94: pop BX
[X31] 95: pop DX
[3D1] 96: add SI,DX
[BZØ] 97: dec BL
[V31] 98: pop CX
[7P1] 99: jnz yloop
[E31] 100: pop SI
[E31] 101: pop DS
[ZNØ] 102: end;
[RX6] 103: procedure TSprite.DropSprite(x,y:Word);
[PT1] 104: assembler;
[K64] 105: label xloop,yloop,noinc;
[FNØ] 106: asm
[ULØ] 107: cld
[BC1] 108: push DS
[GC1] 109: push SI
[KC1] 110: mov AX,y
[NC1] 111: mov CX,x
[CL3] 112: mov BL,DS:[DI+height]
[CG3] 113: mov BH,DS:[DI+width]
[7D4] 114: mov DX,DS:[DI+lineoffset]
[VF4] 115: shr DX,1 { lineoffset/4 }
[RA1] 116: shr DX,1
[ZC1] 117: push DX
[TC1] 118: push BX
[A94] 119: mov BX,DS:[DI+scrwidth]
[GV3] 120: lds SI,DS:[DI+content]
[LA1] 121: shr BX,1
[LA1] 122: shr BX,1
[N31] 123: mul BX
[EH1] 124: mov BX,CX
[LA1] 125: shr BX,1
[LA1] 126: shr BX,1
[AD1] 127: add AX,BX
[7G4] 128: mov DI,AX { SI Screenpos. }
[T31] 129: pop BX
[X31] 130: pop DX
[ØS3] 131: mov AX,gScreenSegment
[485] 132: mov ES,AX { DS:SI Screenpos. }
```

```
[F31] 133: yloop:
[WC1] 134: push CX
[ZC1] 135: push DX
[TC1] 136: push BX
[PK1] 137: mov DX,Ø3C4h
[RE1] 138: mov AL,2h
[D31] 139: xloop:
[WC1] 140: push CX
[TB1] 141: and CL,3h
[U91] 142: mov AH,1
[LD1] 143: shl AH,CL
[V31] 144: pop CX
[YH1] 145: out DX,AX
[AF3] 146: movsb { Transfer }
[ZC1] 147: cmp AH,8h
[4B1] 148: jz noinc
[UR2] 149: dec DI { div 4 }
[F11] 150: noinc:
[P11] 151: inc CX
[ZVØ] 152: dec BH
[5P1] 153: jnz xloop
[T31] 154: pop BX
[X31] 155: pop DX
[5C1] 156: add DI,DX
[BZØ] 157: dec BL
[V31] 158: pop CX
[7P1] 159: jnz yloop
[E31] 160: pop SI
[E31] 161: pop DS
[ZNØ] 162: end;
[ZW6] 163: procedure TSprite.LoadSprite(f:string);
[ML2] 164: var sprf : FILE;
[ZS1] 165: w,h : BYTE;
[MUØ] 166: begin
[ZW4] 167: Assign(sprf,f); Reset(sprf,1);
[B53] 168: BlockRead(sprf,w,1);
[W43] 169: BlockRead(sprf,h,1);
[M65] 170: if (w<>width) OR (h<>height) then
[5M4] 171: begin Close(sprf); Exit end;
[4L6] 172: BlockRead(sprf,content^,width*height);
[CZ1] 173: Close(sprf);
[ZNØ] 174: end;
[UY6] 175: procedure TSprite.SaveSprite(f:string);
[ML2] 176: var sprf : file;
[MUØ] 177: begin
[IA5] 178: Assign(sprf,f); Rewrite(sprf,1);
[V54] 179: BlockWrite(sprf,width,1);
[HC4] 180: BlockWrite(sprf,height,1);
[5W6] 181: BlockWrite(sprf,content^,width*height);
[CZ1] 182: Close(sprf);
[ZNØ] 183: end;
[K57] 184: procedure TSprite.ToggleSprite(x,y:Word);
[PT1] 185: assembler;
[SU5] 186: label xloop,yloop,noinc,skipbyte;
[FNØ] 187: asm
[ULØ] 188: cld
[BC1] 189: push DS
[3B1] 190: push DI
[GC1] 191: push SI
[KC1] 192: mov AX,y
[NC1] 193: mov CX,x
[RF3] 194: mov DS:[DI+posx],CX
[NF3] 195: mov DS:[DI+posy],AX
[CL3] 196: mov BL,DS:[DI+height]
[CG3] 197: mov BH,DS:[DI+width]
[E94] 198: mov DX,DS:[DI+scrwidth]
[VF4] 199: shr DX,1 { lineoffset/4 }
[RA1] 200: shr DX,1
[ZC1] 201: push DX
[TC1] 202: push BX
[A94] 203: mov BX,DS:[DI+scrwidth]
[NU3] 204: les DI,DS:[DI+content]
[LA1] 205: shr BX,1
[LA1] 206: shr BX,1
[N31] 207: mul BX
[EH1] 208: mov BX,CX
[LA1] 209: shr BX,1
[LA1] 210: shr BX,1
[AD1] 211: add AX,BX
[5H4] 212: mov SI,AX { SI Screenpos. }
[T31] 213: pop BX
[X31] 214: pop DX
[ØS3] 215: mov AX,gScreenSegment
[285] 216: mov DS,AX { DS:SI Screenpos. }
[F31] 217: yloop:
[WC1] 218: push CX
[ZC1] 219: push DX
[TC1] 220: push BX
[GC1] 221: push SI
[D31] 222: xloop:
[VL3] 223: mov AL,byte ptr ES:DI
[BK1] 224: cmp AL,Øh
[A12] 225: jz skipbyte
[4F1] 226: mov AH,CL
[9B1] 227: and AH,3h
[PC1] 228: push AX
[UZ1] 229: mov DX,Ø3CEh
```



```
[TE1] 230: mov AL,4h
[YH1] 231: out DX,AX
[PX0] 232: movsb
[401] 233: dec SI
[6Z0] 234: dec DI
[R31] 235: pop AX
[WC1] 236: push CX
[NM1] 237: xchg AH,AL
[EH1] 238: mov CX,AX
[U91] 239: mov AH,1
[LD1] 240: shl AH,CL
[PX1] 241: mov DX,03C4h
[RE1] 242: mov AL,2h
[YH1] 243: out DX,AX
[BM3] 244: mov byte ptr DS:SI,CH
[V31] 245: pop CX
[LM1] 246: skipbyte:
[WC1] 247: push CX
[TB1] 248: and CL,3h
[AD1] 249: cmp CL,3h
[7M1] 250: jnz noinc
[911] 251: inc SI
[F11] 252: noinc:
[V31] 253: pop CX
[B01] 254: inc DI
[P11] 255: inc CX
[ZV0] 256: dec BH
[5P1] 257: jnz xloop
[E31] 258: pop SI
[T31] 259: pop BX
[X31] 260: pop DX
[3D1] 261: add SI,DX
[V31] 262: pop CX
[BZ0] 263: dec BL
[7P1] 264: jnz yloop
[E31] 265: pop SI
[G21] 266: pop DI
[E31] 267: pop DS
[ZN0] 268: end;
[BM5] 269: procedure TSprite.SecondToggle;
[VU5] 270: begin ToggleSprite(posx,posy) end;
[LM4] 271: destructor TSprite.Done;
[MU0] 272: begin
[D65] 273: FreeMem(content,width*height);
[7A2] 274: content:=NIL;
[ZN0] 275: end;
[KN0] 276: end.
```

Listing 1. Mit »spritefx.pas« bringen Sie Bewegung in Ihre VGA-Bilder.

```
[8F3] 1: {Programm: demo.pas
[Y05] 2: Funktion: Demonstration zur
[RT3] 3: Sprite-Programmierung
[G64] 4: Sprache: Turbo Pascal 6.0
[K34] 5: Autor: Frank Höppner
[585] 6: (c) 1992 DMV Widuch GmbH & Co. KG }
[000] 7:
[492] 8: program Demo;
[000] 9:
[X94] 10: uses VGAfx,Spritefx,Crt;
[000] 11:
[PV5] 12: { benötigt wie in Bild 2 abgebildet,
[5X4] 13: PCX-Datei, die Hintergrund und
[SY4] 14: Animationsphasen enthält }
[000] 15:
[G46] 16: { mit den Nummern-Tasten läßt sich
[FN6] 17: der Hubschrauber bewegen, ESC beendet }
[000] 18:
[MV6] 19: var hubschrauber : Array[0..3] of TSprite;
[GZ3] 20: viewx,viewy : Integer;
[7R2] 21: procedure Init;
[8X2] 22: var i,j : Integer;
[MU0] 23: begin
[642] 24: vga.Init(400);
[9F1] 25: vga.Hide;
[453] 26: viewy:=25; viewx:=0;
[CG4] 27: vga.SetView(viewx,viewy);
[Z22] 28: vga.SetSplitt(20);
[R44] 29: vga.PCX('SCHRAUB.PCX',0,0);
[2G2] 30: for i:=0 to 3 do
[2N4] 31: hubschrauber[i].Init(68,21);
[2G2] 32: for i:=0 to 3 do
[8Z5] 33: hubschrauber[i].GrabSprite(0,i*21);
[ZF4] 34: vga.Copy(110-32,40,4*32,40,0,600);
[AP4] 35: vga.Copy(110-32,80,4*32,40,150,600);
[GR3] 36: vga.Copy(0,178,320,25,0,0);
[XV4] 37: for i:=0 to 2 do for j:=0 to 3 do
[K15] 38: vga.Copy(0,600,4*32,39,i*128,25+j*39);
[0G2] 39: for i:=0 to 2 do
[6T4] 40: vga.Copy(150,600,4*32,39,i*128,162);
[FW3] 41: vga.Copy(0,25,390,200,0,325);
[UH1] 42: vga.Show;
```

```
[ZN0] 43: end;
[GG3] 44: procedure Animation;
[B12] 45: label NoKey;
[CE4] 46: var i,r,w,oldview,rebuild,
[HY5] 47: xpos,ypos,xdelta,ydelta : Integer;
[K23] 48: c : Char;
[8R3] 49: procedure SwitchView;
[MU0] 50: begin
[FD4] 51: if viewy=325 then viewy:=25
[LM2] 52: else viewy:=325;
[NW5] 53: vga.SetView(viewx+rebuild,viewy);
[ZN0] 54: end;
[000] 55:
[MU0] 56: begin
[4M3] 57: r:=0; w:=2; rebuild:=0;
[6T3] 58: viewy:=25; oldview:=325;
[Y53] 59: xpos:=100; ypos:=100;
[V73] 60: xdelta:=0; ydelta:=0;
[000] 61:
[316] 62: { Initialisierung (Vorbelegung, d.h.
[EB3] 63: einmal schreiben) }
[F56] 64: hubschrauber[0].ToggleSprite(xpos,
[F22] 65: viewy+ypos);
[K56] 66: hubschrauber[1].ToggleSprite(xpos,
[4G2] 67: oldview+ypos);
[KL4] 68: { Schleife für Animation }
[P31] 69: repeat
[XA4] 70: { Liest Tastatureingabe,
[FZ2] 71: falls vorhanden }
[FN0] 72: asm
[AE1] 73: mov AH,1h
[G81] 74: int 16h
[WAL] 75: jz NoKey
[9E1] 76: mov AH,0h
[G81] 77: int 16h
[TA1] 78: mov c,AL
[U21] 79: NoKey:
[ZN0] 80: end;
[V93] 81: if c IN ['1'..'9'] then
[MU0] 82: begin
[V73] 83: xdelta:=0; ydelta:=0;
[AL3] 84: if c IN ['7','4','1'] then
[TK1] 85: xdelta:=-1
[5F4] 86: else if c IN ['9','6','3'] then
[1M1] 87: xdelta:=2;
[UM3] 88: if c IN ['7','8','9'] then
[VK1] 89: ydelta:=-1
[TE4] 90: else if c IN ['1','2','3'] then
[1M1] 91: ydelta:=1;
[ZN0] 92: end;
[000] 93:
[WG4] 94: { auf "fertigen" Bildbereich
[832] 95: umschalten }
[9V2] 96: oldview:=viewy;
[TX1] 97: Inc(viewx);
[X42] 98: SwitchView;
[DF3] 99: if (viewx>=3*128-320) AND
[MF4] 100: (rebuild=0) then rebuild:=1;
[TA4] 101: if rebuild=0 then delay(30);
[R26] 102: { anderen Bereich zunächst löschen
[2B5] 103: und dann aktualisiert aufbauen }
[U85] 104: hubschrauber[r].SecondToggle;
[0E2] 105: r:=succ(r) MOD 4;
[1W2] 106: if rebuild<>0 then
[MU0] 107: begin
[B15] 108: vga.Copy(viewx,oldview,320,180,
[MV1] 109: 0,oldview);
[2E5] 110: if rebuild=1 then rebuild:=-viewx
[WPI] 111: else begin
[NC3] 112: xpos:=xpos-viewx+1;
[514] 113: viewx:=0; rebuild:=0 end;
end;
[RF6] 115: xpos:=xpos+xdelta; ypos:=ypos+ydelta;
[UU3] 116: { Bereichskontrolle }
[CZ4] 117: if ypos>145 then ypos:=145 else
[0S3] 118: if ypos<0 then ypos:=0;
[XU5] 119: if xpos<viewx then xpos:=viewx+1
[MC4] 120: else if xpos>viewx+320-68
[K44] 121: then xpos:=xpos-xdelta;
[196] 122: hubschrauber[w].ToggleSprite(xpos+
[6W3] 123: rebuild,oldview+ypos);
[ME2] 124: w:=succ(w) MOD 4;
[1V1] 125: until c=#27;
[ZN0] 126: end;
[1R2] 127: procedure Exit;
[2P2] 128: var i : Integer;
[MU0] 129: begin
[2C6] 130: for i:=0 to 3 do hubschrauber[i].Done;
[VGI] 131: vga.Done;
[ZN0] 132: end;
[MU0] 133: begin
[AV0] 134: Init;
[VS1] 135: Animation;
[0W0] 136: Exit;
[KN0] 137: end.
```

Listing 2. Eine einfache Methode, Ihren VGA-Bildern Leben einzuhauchen, zeigt Ihnen »demo.pas«.

Vektor-Mania

Dreidimensionale Drahtmodelle sind Ihnen bestimmt schon begegnet. Viele CAD-Programme arbeiten mit solchen Objekten, wobei die Animation jedoch mit geringer Geschwindigkeit abläuft. Wenn Sie das hier vorgestellte Listing abtippen, werden Sie flüssige, bewegte Vektorgrafik in Echtzeit bewundern können.

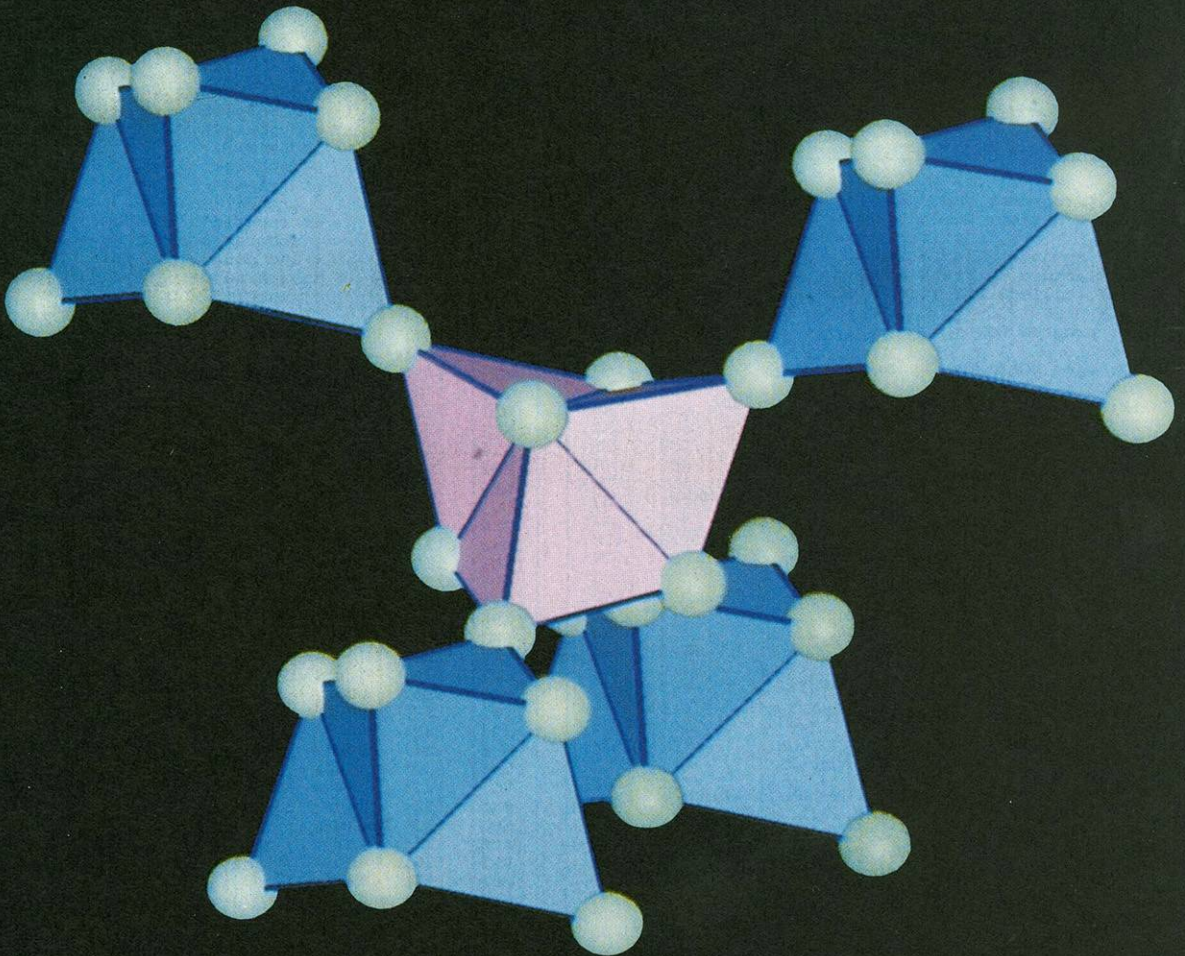
Sicherlich kennen Sie Vektoren noch aus der Schule. Ein Vektor ist ein Pfeil, der zum Beispiel im Ursprung eines Koordinatensystems beginnt und dann auf einen bestimmten Punkt im Raum zeigt. Mathematisch wird ein dreidimensionaler Vektor durch drei Koordinaten charakterisiert, den x-, y- und z-Anteil, entsprechend den drei Raumrichtungen. Das Schöne an Vektoren ist, daß sich mit ihnen leicht rechnen läßt. Findige Leute haben nämlich einen Satz Rechenoperationen ausgeklügelt, mit denen man die Vektoren verarbeiten kann. Wenn Sie die Grundrechenarten beherrschen und wissen, was ein Sinus und was ein Kosinus

ist, sind Sie bestens gerüstet, bewegte Vektorgrafik in Echtzeit zu erzeugen.

Die erste Frage, auf die Sie eine Antwort suchen, ist: Wie drehe ich einen Punkt im Raum? Nun, dafür gibt es Drehmatrizen. Schreiben Sie zwei der Vektoren untereinander, Element für Element, und schon haben Sie eine Matrix mit drei Spalten und zwei Zeilen. In diesem Fall werden die Vektoren Zeilenvektoren genannt. Wenn Sie sie nebeneinander schreiben, heißen sie Spaltenvektoren. Sie müssen sich allerdings für eine Schreibweise entscheiden. Wenn Sie Spalten mit Zeilen vertauschen, funktioniert es nicht.

Die Drehmatrix ist schon mit bestimmten Elementen gefüllt. Diese leiten Sie her, indem Sie ein kartesisches Koordinatensystem zeichnen und überlegen, mit welchen Operationen Sie einen Punkt in einen anderen überführen. Allzu kompliziert sind diese Operationen nicht. Das Bild zeigt Ihnen die Matrizen für die Drehung im dreidimensionalen Raum.

Nun zu den Rechenregeln. Ein Vektor wird gedreht, indem er mit einer Drehmatrix multipliziert wird. Das geht immer nach der Devise: Zeile mal Spalte. Damit die Vektor-mal-Matrix-Multiplikation funktioniert, muß Ihr Vektor, in diesem Fall ein Spaltenvektor, so viele Zeilen haben wie die Matrix Spalten. Nehmen Sie das erste Element des Vektors und multiplizieren Sie es mit dem ersten Element der ersten Zeile der Matrix. Dann addieren Sie das Produkt aus der zweiten Komponente des Vektors und des zweiten Elements der ersten Zeile der Matrix. Das machen Sie mit allen Elementen des Vektors. Sie erhalten das erste Element des Ergebnisvektors. Nach diesem Muster verarbeiten Sie den Rest. Sie verrechnen also die Vektorelemente mit der zweiten Zeile der Matrix, und schon erhalten Sie das zweite Ele-



ment des gedrehten Vektors. Noch einmal, und Sie haben den gedrehten Vektor fertiggestellt.

Werten Sie die Drehmatrizen aus, so erhalten Sie folgende Transformationsgleichungen, die angeben, mit welchen Rechenoperationen Sie einen Punkt durch Drehung um den Winkel »a« in einen anderen Punkt überführen. Für die Drehung um die x-Achse:

$$z' = z \cdot \cos(a) - y \cdot \sin(a)$$

$$y' = z \cdot \sin(a) + y \cdot \cos(a)$$

Für die Drehung um die y-Achse:

$$x' = x \cdot \cos(a) - z \cdot \sin(a)$$

$$z' = x \cdot \sin(a) + z \cdot \cos(a)$$

Und schließlich für die Drehung um die z-Achse:

$$x' = x \cdot \cos(a) - y \cdot \sin(a)$$

$$y' = x \cdot \sin(a) + y \cdot \cos(a)$$

In »vector.asm« (Listing) werden diese Gleichungen mit dem Makro »Rot« ausgerechnet.

Wir basteln eine Sinusfunktion

Ein Blick auf die Transformationsgleichungen zeigt Ihnen, daß Sie Sinus- und Kosinuswerte benötigen. Leider gibt es solche Funktionen auf Assembler-Ebene nicht. Ein Griff in die Trickkiste zaubert die Sinustabelle hervor. Dabei machen Sie sich die Tatsache zunutze, daß die Kosinuswerte gleich den Sinuswerten sind, nur um 90 Grad verschoben. Sie benötigen also nur eine Tabelle für beide Winkelfunktionen. Während der Animation werden ständig die Drehwinkel (xw, yw, zw) erhöht. Beim Einsprung in die Sinusroutine beschränken Sie die Größe des Winkels auf 127 Werte durch Und-Verknüpfung mit »ffhex«. Bit 7 ist das Vorzeichen-Bit. Dadurch werden nur Winkel von 0 bis 127 zugelassen. Der Wert »127« entspricht dabei π , also 180 Grad. Um beispielsweise den Sinus von 90 Grad zu erhalten, entnehmen Sie nur der Tabelle den 64sten Wert. Wird ein Winkel von 128 bis 255 eingegeben, bedeutet das, daß Sie sich im negativen Teil der Sinuskurve befinden. Dann wird der entsprechende positive Wert verwendet und negiert. Auf diese Weise halten Sie die Tabelle klein und sparen Speicherplatz. Noch kleinere Tabellen sind zwar erlaubt, rufen jedoch eine stark flimmernde Animation hervor. Wie schon erwähnt, sind die Kosinuswerte um $\pi/2$ phasenverschoben, so daß beim Eintritt in die Kosinusroutine nur 64 zu dem Winkel addiert und dann die Sinusroutine abgearbeitet wird.

Der Weg vom Raum zur Ebene

Nachdem Sie nun wissen, wie Sie einen Vektor drehen, bilden Sie im nächsten Schritt die dreidimensionale Welt auf zwei Dimensionen ab. Der einfachste Weg dorthin ist, die z-Koordinate einfach zu ignorieren. Dieses Vorgehen führt aber zu einem gewissen Verlust an Perspektive. Im Listing wird deshalb ein rechenintensiverer Weg beschritten. Im Unterprogramm »transform« wird die x-Koordinate mit der Konstanten »dist« multipliziert und durch die z-Koordinate plus »dist« geteilt. Genauso machen Sie es mit der y-Koordinate. Sie erhalten dadurch eine räumliche Wirkung, denn die Teile des Objekts, die gerade auf Sie zu gedreht werden, erscheinen größer, die weiter hinten liegenden Teile wirken kleiner. Durch Vergrößern des Parameters »dist« verkleinern Sie das Objekt (zoom). Natürlich können Sie es auch vergrößern. Achten Sie aber darauf, daß Ihr Objekt nicht die Bildschirmbegrenzung verläßt, sonst stürzt Ihr Rechner unter Umständen ab. Ein Abschneiden der Linien am Bildschirmrand (clipping) ist nicht vorgesehen. In einer der nächsten Ausgaben der DOS International wird ausführlicher darauf eingegangen.

Die so transformierten Punkte werden in den Arrays »xplot« und »yplot« gespeichert, aus denen die Drawline-Routine sie herausholt. Wie die Linien gezeichnet werden, lesen Sie

Drehung um die x-Achse:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Drehung um die y-Achse:

$$\begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Drehung um die z-Achse:

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Die Matrizen für
die Drehung im
dreidimensionalen
Raum

in der Profi-Programmierung der DOS International 2'92 (Seite 180) nach. Auch hier findet der Tabellen-Trick wieder Anwendung. Im Verlauf der Berechnung der Bildschirmspeicheradresse eines Punktes wird mit der Bildschirmbreite in Byte (80) multipliziert. Sie ersetzen diesen zeitraubenden Mul-Befehl, indem Sie nicht die x-te Bildschirmzeile mit 80 multiplizieren, sondern den x-ten Wert einer Tabelle entnehmen.

Welten im Computer nachbilden

Sie haben nun das gesamte Handwerkszeug zur Animation von Drahtgittermodellen in Händen. Das Modell, in dem die Objekte dargestellt werden, ist die denkbar einfachste Struktur. Zunächst legen Sie Ihre Weltkoordinaten fest. Dazu nehmen Sie ein Blatt kariertes Papier, zeichnen ein Koordinatenkreuz und legen Ihre Begrenzungspunkte fest. Numerieren Sie die Zentren, lesen Sie die Koordinaten ab und tragen Sie sie ab den Labels »xk«, »yk« und »zk« im Listing ein. Anschließend füllen Sie die Verbindungsliste, die Ihnen angibt, welche Zentren miteinander verbunden werden sollen. Tragen Sie die Begrenzungspunkte einer Linie ab dem Label »linie« ein. Damit haben Sie Ihr Objekt definiert. Legen Sie noch die Konstanten »NoA« (Anzahl Zentren) und »NoB« (Anzahl Linien) sowie die Drehwinkelinkremente im Unterprogramm »Input« fest.

Bald werden Sie erkennen, daß die Papiermethode zur Objektdefinition sehr mühsam ist. Ein Objekteditor wäre nützlich. Fällt Ihnen dazu eine Lösung ein? Vielleicht werten Sie ja Vektorformate gängiger CAD-Programme aus, so daß Sie Ihr Objekt zum Beispiel mit Autocad erzeugen, konvertieren und mit dem Echtzeit-Vektorgenerator beleben. Senden Sie uns doch Ihre Lösung. Damit die Animation fließend erscheint, wenden Sie das sogenannte Double-Buffering an. Sie benötigen dazu zwei Bildschirminhalte. In den einen – unsichtbaren – Bildschirm zeichnen Sie, während der andere angezeigt wird. Sobald Sie mit dem Zeichnen fertig sind, schalten Sie die Bildschirme um, so daß der Bildschirm, in dem Sie gerade gezeichnet haben, schlagartig sichtbar wird. Vielleicht werden Sie auf Ihrem Rechner ein leichtes Flackern der Grafik bemerken. Das schalten Sie ab, indem Sie auf den vertikalen Strahlrücklauf warten, bevor Sie die Bildschirme umschalten. Sie können auch ein Triple-Buffering verwenden, um noch weichere Bewegungen zu erreichen. Sicher werden Sie noch weitere verbesserungswürdige Passagen entdecken. Zögern Sie nicht, uns Ihre optimierten Routinen zuzuschicken. Die besten werden wir abdrucken.

(bm)


```
[2R3] 1: ; Programm: vector.asm
[NE5] 2: ; Funktion: Drahtmodelle drehen
[LG3] 3: ; Sprache: TASM ab 2.01
[5S4] 4: ; Autor: Burkhard Müller
[7X4] 5: ; (c) 1992 DMV Widuch GmbH & Co. KG
[000] 6:
[XY1] 7: .model small
[FS1] 8: .stack 200h
[EU4] 9: NoA = 21 ; Anzahl Zentren
[TH4] 10: NoB = 24 ; Anzahl Linien
[2S0] 11: .data
[DU1] 12: Screen dw 0
[BY3] 13: Screen0 dw 0a000h ; page 0
[9Z3] 14: Screen1 dw 0a800h ; page 1
[UU4] 15: OldMode dw 0 ; Grafikmodus
[5L1] 16: farbe db 4
[YY1] 17: zeichen dw 0
[6N2] 18: Tab80 label word
[2P1] 19: Start80 = 0
[471] 20: rept 350
[NT1] 21: dw Start80
[E93] 22: Start80 = Start80 + 80
[HP0] 23: endm
[303] 24: ; Weltkoordinaten
[H25] 25: xk dw -680,-1020,-1020,-680,-350,0,350,350
[XK4] 26: dw 0,-350,0,320,650,650,1010,1270,1010
[CP2] 27: dw 650,-350,-1270,1160
[B35] 28: yk dw 0,-150,-570,-720,-570,-720,-570,-150
[6A4] 29: dw 0,-150,400,570,400,0,-120,200,500
[PF2] 30: dw 400,-150,-720,850
[NZ4] 31: zk dw 0,150,150,300,150,300,150,0,150
[PE4] 32: dw 0,-150,0,0,130,100,-100,400,550,200
[ET0] 33: dw 100
[7R2] 34: ; Bindungsliste
[A05] 35: lines db 2,3,3,4,4,5,6,7,7,8,8,9,9,0,0,1
[CS3] 36: db 1,2,9,4,2,19,18,9,7,13,13,12
[TR3] 37: db 12,11,11,10,10,8,13,14,14,15
[MK3] 38: db 15,16,16,12,12,17,16,20,5,6
[K12] 39: Dist dw 2000
[G36] 40: ; Sinustabelle: Auflösung: 1.406 Grad
[WG5] 41: SinTab dw 0000h,0192h,0324h,04b5h,0646h
[MA5] 42: dw 07d6h,0964h,0af1h,0c7ch,0e06h,0f8dh
[YZ4] 43: dw 1112h,1294h,1413h,1590h,1709h,187eh
[6F5] 44: dw 19efh,1b5dh,1cc6h,1e2bh,1f8ch,20e7h
[N65] 45: dw 223dh,238eh,24dah,2620h,2760h,289ah
[HE5] 46: dw 29ceh,2afbh,2c21h,2d41h,2e5ah,2f6ch
[P05] 47: dw 3076h,3179h,3274h,3368h,3453h,3537h
[G55] 48: dw 3612h,36e5h,37b0h,3871h,392bh,39dbh
[ZB5] 49: dw 3a82h,3b21h,3bb6h,3c42h,3cc5h,3d3fh
[FG5] 50: dw 3dafh,3e15h,3e72h,3ec5h,3f0fh,3f4fh
[1E5] 51: dw 3f85h,3fb1h,3fd4h,3fech,3ffbh,4000h
[6K5] 52: dw 3ffbh,3fech,3fd4h,3fb1h,3f85h,3f4fh
[6G5] 53: dw 3f0fh,3ec5h,3e72h,3e15h,3dafh,3d3fh
[WAS] 54: dw 3cc5h,3c42h,3bb6h,3b21h,3a82h,39dbh
[T35] 55: dw 392bh,3871h,37b0h,36e5h,3612h,3537h
[K35] 56: dw 3453h,3368h,3274h,3179h,3076h,2f6ch
[SC5] 57: dw 2e5ah,2d41h,2c21h,2afbh,29ceh,289ah
[D55] 58: dw 2760h,2620h,24dah,238eh,223dh,20e7h
[HG5] 59: dw 1f8ch,1e2bh,1cc6h,1b5dh,19efh,187eh
[K15] 60: dw 1709h,1590h,1413h,1294h,1112h,0f8dh
[M75] 61: dw 0e06h,0c7ch,0af1h,0964h,07d6h,0646h
[AN2] 62: dw 04b5h,0324h,0192h
[0B1] 63: x1 dw 0
[3B1] 64: x2 dw 0
[2B1] 65: y1 dw 0
[5B1] 66: y2 dw 0
[Z23] 67: xw dw 0 ; Winkel
[ME1] 68: yw dw 0
[PE1] 69: zw dw 0
[9A6] 70: xneu dw 3*NoA DUP (?) ; gedrehte Koord.
[S76] 71: xplot dw 2*NoA DUP (?) ; 2D-Koordinaten
[FT0] 72: .code
[XX0] 73: JUMPS
[3N2] 74: SCR MACRO Seite
[2F1] 75: mov ah,5
[M82] 76: mov al,Seite
[G81] 77: int 10h
[HP0] 78: ENDM
[RE4] 79: Rot MACRO Win,K1,K2,Kn1,Kn2
[UM4] 80: mov ax,Win ; Drehwinkel
[M32] 81: CALL Cosinus
[FL2] 82: imul WORD PTR K1
[7W1] 83: mov cx,dx
[AY1] 84: mov ax,Win
[1Z1] 85: CALL Sinus
[HL2] 86: imul WORD PTR K2
[PUI] 87: sub dx,cx
[CD1] 88: sal dx,1
[CD1] 89: sal dx,1
[D45] 90: mov Kn1,dx ; gedrehte Koord.
[000] 91:
[UM4] 92: mov ax,Win ; Drehwinkel
[1Z1] 93: CALL Sinus
[FL2] 94: imul WORD PTR K1
[7W1] 95: mov cx,dx
[AY1] 96: mov ax,Win
[M32] 97: CALL Cosinus
[HL2] 98: imul WORD PTR K2
[7S1] 99: add dx,cx
[CD1] 100: sal dx,1
```

```
[CD1] 101: sal dx,1
[F45] 102: mov Kn2,dx ; gedrehte Koord.
[HP0] 103: ENDM
[F62] 104: mov ax,@data
[XV1] 105: mov ds,ax
[U82] 106: call ReadMode
[0M2] 107: mov OldMode,ax
[YN1] 108: mov ax,16
[F42] 109: call SetMode
[3S1] 110: call Page0
[A02] 111: mov Farbe,4
[MK1] 112: MainLoop:
[6Y1] 113: call Input
[BW1] 114: call Drehen
[DP2] 115: call Transform
[K92] 116: call DrawLine
[7S1] 117: call Page1
[PS2] 118: call ClearScreen
[BW1] 119: call Drehen
[DP2] 120: call Transform
[K92] 121: call DrawLine
[3S1] 122: call Page0
[PS2] 123: call ClearScreen
[WR2] 124: call KeyPressed
[TB2] 125: jmp MainLoop
[932] 126: KeyPressed:
[PE1] 127: mov ah,1
[P81] 128: int 16h
[8M2] 129: mov zeichen,ax
[061] 130: je NoK
[MH1] 131: jmp Ende
[DC1] 132: NoK: ret
[XD3] 133: Ende:mov ax,OldMode
[S42] 134: call SetMode
[RS1] 135: mov ah,4ch
[SB1] 136: int 21h
[XL1] 137: Page0 PROC
[CL2] 138: mov ax,Screen1
[YL2] 139: mov Screen,ax
[R11] 140: SCR 0
[3P0] 141: ret
[YK1] 142: Page0 ENDP
[ZL1] 143: Page1 PROC
[BL2] 144: mov ax,Screen0
[YL2] 145: mov Screen,ax
[V11] 146: SCR 1
[3P0] 147: ret
[OL1] 148: Page1 ENDP
[KS1] 149: Input PROC
[3E1] 150: add xw,2
[DE1] 151: add yw,4
[BE1] 152: add zw,2
[3P0] 153: ret
[LR1] 154: Input ENDP
[0W2] 155: ClearScreen PROC
[VH2] 156: mov ax,Screen
[1W1] 157: mov es,ax
[UL0] 158: cld
[6U1] 159: xor di,di
[WV1] 160: xor ax,ax
[7A2] 161: mov cx,350*80/2
[602] 162: rep stosw
[3P0] 163: ret
[ZT2] 164: ClearScreen ENDP
[VX1] 165: Drehen PROC
[GS1] 166: lea di,xk
[722] 167: lea si,xneu
[2Y1] 168: mov cx,NoA
[121] 169: Bonds:
[R91] 170: push cx
[7P6] 171: Rot zw,[di],[di+2*NoA],[si],[si+2*NoA]
[DP6] 172: Rot yw,[di],[di+4*NoA],[si],[si+4*NoA]
[PC8] 173: Rot xw,[di+2*NoA],[si+4*NoA],[si+2*NoA],
[si+4*NoA]
[151] 174: inc di
[151] 175: inc di
[X61] 176: inc si
[X61] 177: inc si
[H71] 178: pop cx
[WX1] 179: LOOP Bonds
[3P0] 180: ret
[VX1] 181: Drehen ENDP
[P82] 182: DrawLine PROC
[H72] 183: lea si,lines
[A82] 184: lea di,xplot
[3Y1] 185: mov cx,NoB
[G22] 186: Draw_Bonds:
[R91] 187: push cx
[LT1] 188: xor bh,bh
[6W4] 189: mov bl,[si] ; Nummer holen
[DD1] 190: shl bl,1
[XV3] 191: mov ax,[di+bx] ; x1
[864] 192: mov cx,[di+2*NoA+bx] ; y1
[BX1] 193: mov x1,ax
[NX1] 194: mov y1,cx
[X61] 195: inc si
[B52] 196: mov bl,[si]
[DD1] 197: shl bl,1
[YV3] 198: mov ax,[di+bx] ; x2
[964] 199: mov cx,[di+2*NoA+bx] ; y2
```



```
[CX1] 200: mov x2,ax
[PX1] 201: mov y2,cx
[8B1] 202: push si
[C91] 203: push di
[391] 204: call DL1
[671] 205: pop di
[291] 206: pop si
[X61] 207: inc si
[H71] 208: pop cx
[3U2] 209: LOOP Draw_Bonds
[3P0] 210: ret
[M72] 211: DrawLine ENDP
[9F2] 212: Transform PROC
[2Y1] 213: mov cx,NoA
[6A2] 214: lea si,xplot
[B02] 215: lea di,xneu
[795] 216: TF1: mov bx,[di+4*NoA] ; z-Koord.
[G02] 217: sub bx,Dist
[Y12] 218: mov ax,Dist
[M73] 219: imul WORD PTR [di]
[VX4] 220: idiv bx ; x*Dist/(z-Dist)
[SD1] 221: sar ax,1
[SD1] 222: sar ax,1
[SD1] 223: sar ax,1
[SD1] 224: sar ax,1
[3L1] 225: add ax,320
[YC4] 226: mov [si],ax ; in xplot
[Y12] 227: mov ax,Dist
[AB5] 228: imul WORD PTR [di+2*NoA] ; y*Dist
[S61] 229: idiv bx
[SD1] 230: sar ax,1
[SD1] 231: sar ax,1
[SD1] 232: sar ax,1
[SD1] 233: sar ax,1
[PL1] 234: add ax,175
[1X4] 235: mov [si+2*NoA],ax ; in yplot
[151] 236: inc di
[151] 237: inc di
[X61] 238: inc si
[X61] 239: inc si
[9D1] 240: LOOP TF1
[3P0] 241: ret
[AE2] 242: Transform ENDP
[492] 243: Cosinus PROC
[FL3] 244: ; Eingabe : ax = Winkel
[MR5] 245: ; Ausgabe : ax = Sinus oder Cosinus
[P30] 246: ;
[4K1] 247: add ax,64
[DS1] 248: Sinus PROC
[L91] 249: push bx
[R91] 250: push cx
[8B1] 251: push si
[UV1] 252: mov bx,ax
[ND1] 253: shl ax,1
[Y05] 254: and ax,0ffh ; 1 von 128 words
[PF2] 255: lea si,SinTab
[ET1] 256: add si,ax
[ZF5] 257: mov ax,[si] ; (Co-)Sinus holen
[PL0] 258: cld
[AW4] 259: shl bl,1 ; Bit 7 gesetzt ?
[BX1] 260: jae cospos
[HK4] 261: neg ax ; dann negieren
[EC1] 262: cospos:
[291] 263: pop si
[H71] 264: pop cx
[D71] 265: pop bx
[3P0] 266: ret
[ER1] 267: Sinus ENDP
[372] 268: Cosinus ENDP
[472] 269: ReadMode PROC
[4V5] 270: mov ah,0fh ; aktuellen Bildschirm-
[GY3] 271: int 10h ; modus lesen
[3P0] 272: ret
[262] 273: ReadMode ENDP
[562] 274: SetMode PROC
[666] 275: mov ah,00h ; Bildschirmmodus setzen
[G81] 276: int 10h
[3P0] 277: ret
[442] 278: SetMode ENDP
[PH5] 279: ; Linie von x1, y1 nach x2, y2 ziehen
[G52] 280: ; mit Farbe CH
[5B1] 281: DL1 PROC
[ZS1] 282: mov ax,x1
[HR1] 283: cmp ax,x2
[G12] 284: jz NoNullx
[ZD2] 285: jmp DL_Start
[CH1] 286: NoNullx:
[2T1] 287: mov ax,y1
[MR1] 288: cmp ax,y2
[AF2] 289: jnz DL_Start
[0Z1] 290: jmp DL_End
[PN1] 291: DL_Start:
[B91] 292: push bp
[ZS1] 293: mov ax,x1
[6T1] 294: mov bx,y1
[F91] 295: push ax
[L91] 296: push bx
[B12] 297: mov bx,4340h
[BT1] 298: mov cx,x2
```

```
[CU1] 299: sub cx,ax
[WF2] 300: jge deltax_p3
[FS1] 301: mov bl,48h
[B51] 302: neg cx
[5U1] 303: deltax_p3:
[KT1] 304: mov dx,y2
[291] 305: pop si
[8B1] 306: push si
[AU1] 307: sub dx,si
[XF2] 308: jge deltax_p3
[LT1] 309: mov bh,4bh
[F51] 310: neg dx
[8U1] 311: deltax_p3:
[LT4] 312: mov si,offset cs:dabgross3
[924] 313: mov word ptr cs:[si],bx
[NU1] 314: cmp cx,dx
[GM2] 315: jge deltax_gr3
[YR1] 316: mov bl,90h
[8V1] 317: xchg cx,dx
[KW1] 318: jmp achse3
[D22] 319: deltax_gr3:
[UR1] 320: mov bh,90h
[961] 321: achse3:
[RR4] 322: mov si,offset cs:dabklein3
[924] 323: mov word ptr cs:[si],bx
[1E1] 324: shl dx,1
[VV1] 325: mov bp,dx
[PU1] 326: sub dx,cx
[WV1] 327: mov di,dx
[PU1] 328: sub dx,cx
[D71] 329: pop bx
[971] 330: pop ax
[HN1] 331: schleife3:
[C91] 332: push di
[F91] 333: push ax
[V91] 334: push dx
[L91] 335: push bx
[R91] 336: push cx
[662] 337: mov ch,farbe
[SB2] 338: call SetPixel
[H71] 339: pop cx
[D71] 340: pop bx
[N71] 341: pop dx
[971] 342: pop ax
[671] 343: pop di
[ED1] 344: cmp di,0
[9F2] 345: jge dabgross3
[BM1] 346: dabklein3:
[451] 347: inc ax
[851] 348: inc bx
[NP1] 349: add di,bp
[SH2] 350: loop schleife3
[KH2] 351: jmp end_line3
[YP1] 352: dabgross3:
[451] 353: inc ax
[851] 354: inc bx
[VRI] 355: add di,dx
[SH2] 356: loop schleife3
[7R1] 357: end_line3:
[1D1] 358: pop bp
[XB1] 359: DL_End:
[3P0] 360: ret
[491] 361: DL1 ENDP
[5E5] 362: ;Punkt (AX,BX) setzen mit Farbe CH
[H53] 363: SetPixel proc near
[U22] 364: lea si,Tab80
[TD1] 365: shl bx,1
[NK3] 366: mov bx,[si+bx] ;y*80
[2U1] 367: mov cl,al
[DE1] 368: shr ax,1
[DE1] 369: shr ax,1
[DE1] 370: shr ax,1
[TR1] 371: add bx,ax
[HC1] 372: and cl,7
[UF1] 373: xor cl,7
[VH2] 374: mov ax,Screen
[1W1] 375: mov es,ax
[PE1] 376: mov ah,1
[BS1] 377: shl ah,cl
[TY1] 378: mov dx,3ceh
[FF1] 379: mov al,8
[GW1] 380: out dx,ax
[X02] 381: mov ax,0205h
[GW1] 382: out dx,ax
[M12] 383: mov ax,1803h
[GW1] 384: out dx,ax
[NP2] 385: mov al,es:[bx]
[XL2] 386: mov es:[bx],ch
[CF1] 387: mov ax,3
[GW1] 388: out dx,ax
[6F1] 389: mov al,5
[GW1] 390: out dx,ax
[9B2] 391: mov ax,0ff08h
[GW1] 392: out dx,ax
[3P0] 393: ret
[U92] 394: SetPixel ENDP
[4M0] 395: end
```

Listing. Mit »vector.asm« drehen Sie Drahtmodelle in Echtzeit.

Fraktal-Explorer

Apfelmännchen wird entdeckt

Sie sind ein Freund fraktaler Grafiken? Dann steigen Sie mit dem Fraktal-Explorer in Sekundenschnelle hinab in die unendlichen Tiefen der komplexen Zahlen. Begeben Sie sich mit Mandelbrot auf Reisen und entdecken Sie immer wieder neue Strukturen.

Mit dem Computer läßt sich heutzutage fast alles machen, anfangen von der einfachen Textverarbeitung bis hin zur komplexen Simulation von Vorgängen in der Erdatmosphäre. Aber eines der schönsten Dinge, zu denen diese Geräte fähig sind, ist das Berechnen von Bildern, die ohne Computer bisher undenkbar waren.

Hierbei üben Fraktale, wie zum Beispiel das Apfelmännchen, eine besondere Faszination auf den Betrachter aus. Sie haben sich bestimmt schon gefragt, wie diese Bilder eigentlich entstehen. Auch wenn Sie zur Beantwortung dieser Frage zuerst ein wenig in die Mathematik einsteigen müssen, berechnen Sie solche Bilder trotzdem recht einfach mit einem PC.

Was sind komplexe Zahlen?

Aus der Schule kennen Sie sicher noch die sogenannten reellen Zahlen. Diese Zahlenmenge brauchen Sie praktisch täglich. Die reellen Zahlen umfassen die ganzen Zahlen und die Dezimalbrüche. Doch die Mathematiker bemerkten bald, daß die reellen Zahlen nicht vollkommen sind: So gibt es keine reelle Zahl x , die die Gleichung $x \cdot x = -1$ erfüllt. Oder anders ausgedrückt, die Wurzel aus -1 läßt sich im Reellen nicht ziehen. Nun dachten die Mathematiker: »Wenn eine solche Zahl nicht existiert, dann erfinden wir sie einfach.« Und so wurde die imaginäre Einheit »i« und

mit ihr die komplexen Zahlen geschaffen. Für »i« gilt: $i \cdot i = -1$. Damit konnte fast alles, was bisher im Reellen galt, ohne Änderung jetzt auch ins Komplexe übertragen werden. Die allgemeine Darstellung einer komplexen Zahl »z« lautet: $z = x + i y$. In dieser Form wird »x« der Real- und »y« der Imaginärteil von »z« genannt. Hier sehen Sie auch, wie Sie eine komplexe Zahl »z« grafisch darstellen. Während Sie die reellen Zahlen auf einem Zahlenstrahl auftragen, benötigen Sie für die komplexen Zahlen eine Ebene, die komplexe oder auch Gaußsche Zahlenebene (Bild 1). In diese Ebene legen Sie nun ein Koordinatensystem: Auf der x-Achse tragen Sie den Real-, auf der y-Achse den Imaginärteil von »z« auf. So ordnen Sie jeder komplexen Zahl einen Punkt in der komplexen Zahlenebene zu.

Mit Mandelbrot auf Reisen

Falls Sie meinen, man könne Mandelbrot essen, täuschen Sie sich gewaltig. Benoit Mandelbrot, ein Mathematiker, untersuchte in den siebziger Jahren rekursive Folgen. Eine rekursive Folge ist eine Zahlenreihe, bei der das erste Element festgelegt ist und alle weiteren Elemente mit einer Rechenvorschrift aus dem vorherigen abgeleitet werden. So beschreibt die Folge

$$a_0 = 0, a_{n+1} = a_n + 2$$

die Zahlenreihe 0, 2, 4, 6, ..., also alle positiven, geraden Zahlen. Mandelbrot untersuchte dabei die komplexe Folge:

$$z_0 = 0, z_{n+1} = z_n \cdot z_n + c$$

»c« stellt dabei einen komplexen Parameter dar. Mandelbrot fand nun heraus, daß sich die Folge für c-Werte, die nahe am Ursprung liegen, diesem immer mehr näherte, während für weiter entfernte c-Werte die Folge sehr schnell ins Unendliche abwanderte. Er kam nun auf die geniale Idee, dieses Verhalten grafisch darzustellen. Hierzu berechnete er für jeden Punkt der komplexen Zahlenebene diese Folge. Wanderte die Folge ins Unendliche, so färbte er den Punkt weiß, näherte sie sich hingegen dem Ursprung, färbte er ihn schwarz. Mandelbrot staunte nicht schlecht, als er das Ergebnis sah: Das erste Apfelmännchen war entstanden. Die feinen Strukturen im Inneren hielt er zuerst für Rechenfehler, doch beim Vergrößern tauchten jeweils weitere Strukturen auf, die aber in gewisser Weise immer dem Grundbild ähnelten. Dieses Verhalten der Selbstähnlichkeit finden Sie in allen fraktalen Gebilden wieder.

Um zu entscheiden, wie sich ein Punkt verhält, benutzte Mandelbrot folgende einfache Regel: Er legte zunächst einen Kreis um den Ursprung mit dem Radius »3«. Verließ jetzt die Folge diesen Kreis, so wanderte die Folge ins Unendliche und er färbte den Punkt weiß. Befand sich die Fol-

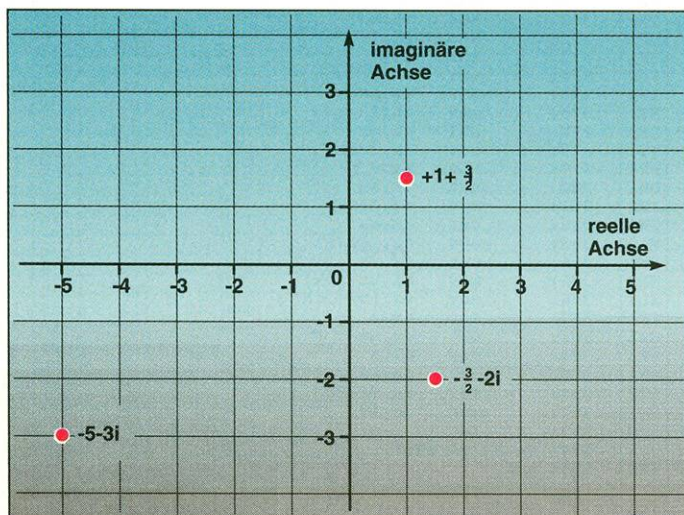


Bild 1. Die komplexe Zahlenebene

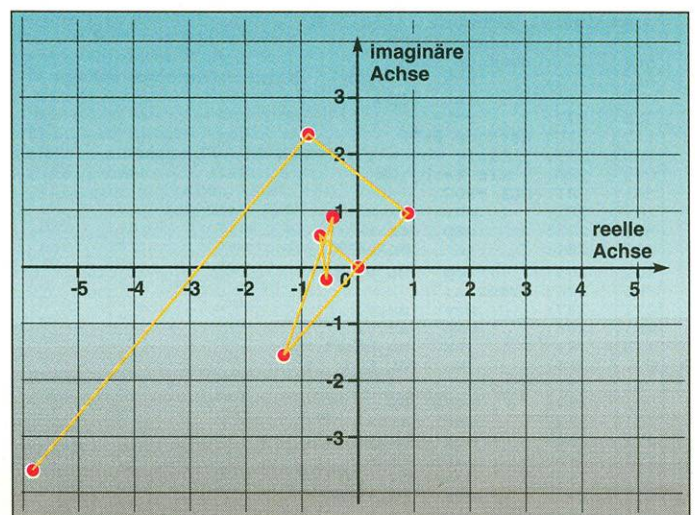


Bild 2. Das Orbital der komplexen Zahl $c = -0,5 + 0,6 i$

ge nach einer gewissen Anzahl von Iterationen immer noch im Kreis, so färbte er den Punkt schwarz.

Hier erkennen Sie auch, warum das Berechnen unter Umständen sehr lange dauern kann: Je feiner die Strukturen sind, desto mehr Iterationen sind notwendig, um zu entscheiden, wie sich ein Punkt verhält. Bild 2 zeigt diese sogenannten Orbitale der komplexen Zahl $c = -0,5 + 0,6i$. Eine andere Variante für die Darstellung besteht darin, die Anzahl der Iterationen, nach welcher die Folge den Kreis verläßt, in eine Farbe umzusetzen und den Punkt damit zu färben. Diese Form wird auch in diesem Programm benutzt.

Eigene Apfelmännchen auf Ihrem PC

Auch auf einem PC berechnen Sie recht leicht Apfelmännchen. Da außer Fortran keine der gängigen Programmiersprachen einen komplexen Datentyp besitzt, rechnen Sie zunächst von Hand. Spalten Sie die von Mandelbrot untersuchte Folge in Real- und Imaginärteil auf, damit Sie mit einer reellen Arithmetik rechnen können. Dazu multiplizieren Sie die Klammer aus:

$$z^2 + c = (x+iy)^2 + c_x + ic_y$$

$$= x^2 + 2ixy - y^2 + c_x + ic_y$$

Getrennt in Real- und Imaginärteil ergibt sich dann:

$$x_{n+1} = x_n * x_n - y_n * y_n + c_x$$

$$y_{n+1} = 2x_n y_n + c_y$$

Um festzustellen, ob die Folge ins Unendliche wandert, berechnen Sie den Abstand von »z« zum Ursprung. Diesen erhalten Sie leicht mit dem Satz des Pythagoras:

$$r^2 = x^2 + y^2$$

Die Iterationen brechen Sie ab, falls der Abstand zum Ursprung größer ist als »3«. Oder anders gesagt, wenn die Folge den Kreis mit dem Radius »3« um den Ursprung verläßt, verschwindet die Folge ins Unendliche und Sie können den Punkt einfärben. Den Radius des Kreises wählen Sie in gewissen Grenzen beliebig.

Eine Schleife in C, um die Iterationen für einen Punkt zu berechnen, zeigt Listing 1. In »x« und »y« sind hierbei die aktuellen Folgeglieder gespeichert, in »x2« und »y2« das Quadrat des jeweiligen Folgeglieds. In »maxit« steht die maximale Iterationstiefe, nach welcher die Iteration abgebrochen wird.

Wenn Sie mit dieser Schleife nun für jeden Punkt die Folge berechnen und mit »it« einfärben, besitzen Sie schon ein Apfelmännchenprogramm. In der Praxis ist dieses Programm aber relativ langsam. Ein besserer Algorithmus berechnet nicht mehr jeden Punkt einzeln, sondern manche Punkte werden gemeinsam eingefärbt. Einfacher ist es jedoch, die Schleife nicht in C, sondern in Assembler zu programmieren und den Coprozessor zu strapazieren. Am schnellsten ist das Programm dann, wenn sich alle Variablen im Prozessorstapel des 80x87-Coprozessors befinden, da so zeitaufwendige Speicherzugriffe vermieden werden.

Superschnelle Fraktale aus dem Coprozessor

Eine kurze Überlegung ergibt, daß Sie folgende Variablen in der Schleife benötigen: x, y, x2, y2, cx, cy und schließlich die Zahl 9 für den Vergleich. Das ergibt sieben Variablen auf die acht Register des Coprozessors. Das achte Register verwenden Sie als Arbeitsregister, da bei Rechenoperationen im 80x87-Coprozessor immer ein Operand im obersten Element des Registerstapels stehen muß.

Die Assembler-Routinen aus »mandel87.asm« (Listing 2) werden von dem Programm »mandel.c« (Listing 3) verwendet, um das Apfelmännchen zu berechnen (Bild 3). Die eigentlichen Berechnungen führt hierbei die Funktion »__mandel« aus. Wie beim Eintritt in die Iterationsschleife von »__mandel« der Registerstapel belegt ist, zeigt die Tabelle. Eingebettet in die Zeilen- und Spaltenschleife wird

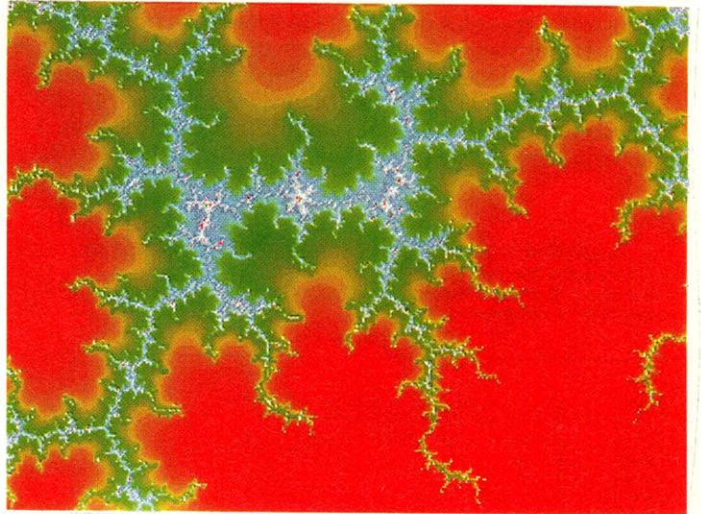


Bild 3. Ein Apfelmännchen in 256 Farben lassen Sie in wenigen Sekunden malen.

nun für jedes Pixel des Ausschnitts, das im Hauptprogramm durch »xmin«, »xmax«, »ymin« und »ymax« gegeben ist, die Farbe berechnet und in den Bildschirmspeicher geschrieben. »mandel87.asm« assemblieren Sie mit MASM ab Version 5.1 oder TASM ab 2.0. Als C-Compiler eignet sich Turbo C. Beim Kompilieren verwenden Sie das Speichermodell »large«, sonst werden die Routinen aus dem Assembler-Modul falsch aufgerufen.

Außerdem benötigen Sie folgende Hardware:

- einen 80x87-Coprozessor,
- eine VGA-Karte,
- eine Microsoft-kompatible Maus.

Nach dem Start des Programms sehen Sie in der oberen Bildschirmhälfte eine Statusmeldung, die anzeigt, welcher Ausschnitt beim nächsten Rechengang berechnet wird. Die Meldung »Bild geladen« oder »kein Bild geladen« gibt an, ob der angegebene Ausschnitt mit dem im Hauptspeicher gespeicherten Bild übereinstimmt oder nicht. Nach Drücken der Taste [1] wird in den Grafikmodus umgeschaltet und das erste Bild berechnet. Zum Menü zurück gelangen Sie durch Drücken einer beliebigen Taste. Zur Vergrößerung einer interessanten Stelle wählen Sie »Zoom«. Damit kommen Sie wieder in den Grafikmodus und wählen mit der Maus einen neuen Ausschnitt. Bei gedrückter rechter Maustaste führen Sie Bewegungen nach rechts oder links aus und verändern so die Größe des Ausschnitts. Haben Sie den gewünschten Ausschnitt gefunden, kehren Sie durch Betätigen der linken Maustaste wieder zum Menü zurück. Anschließend starten Sie mit [1] die Berechnung des neuen Ausschnitts. Erkennen Sie beim Vergrößern keine feinen Strukturen mehr, erhöhen Sie mit »3« die maximale Iterationstiefe. Zu hohe Iterationstiefen verlangsamen das Programm jedoch bei schwarzen Stellen.

Kennen Sie von einem Ausschnitt nur die Begrenzungen in der komplexen Zahlenebene, definieren Sie mit »Ausschnitt wählen« den Bereich von Hand. Dabei geben Sie nur die Werte für »xmin«, »xmax« und »ymin« ein, »ymax« wird automatisch auf das entsprechende Format angepaßt.

Register	Inhalt
st(0)	x
st(1)	y
st(2)	x2
st(3)	y2
st(4)	cx
st(5)	cy
st(6)	9
st(7)	leer

Die Tabelle zeigt die Belegung des Registerstapels.

Mit den Menüpunkten [5] und [6] speichern und laden Sie Bilder. Dabei werden automatisch die Informationen über den gezeigten Ausschnitt und die Iterationstiefe gespeichert. Befinden Sie sich im Menü, so können Sie durch Drücken der Leertaste das gespeicherte Bild betrachten. Mit einer beliebigen Taste kehren Sie wieder in das Menü zurück. (Ralph Würthner/bm)

Literatur:

H. O. Peitgen, P. H. Richter: The Beauty of Fractals, Springer-Verlag 1986, 199 Seiten, ISBN 3-540-15851-0, 88 Mark
H. O. Peitgen, D. Saupe: The Science of Fractal Images, Springer-Verlag 1988, 312 Seiten, ISBN 3-540-96608-0, 79 Mark

```
[CF4] 1: /* einfache Berechnung eines
[V13] 2:   Apfelmännchens */
[ZZ3] 3: x=0; y=0; x2=0; y2=0; it=0;
[NB0] 4: do
[P70] 5: {
[NW1] 6:   y=2*x*y+cy;
[BY1] 7:   x=x2-y2+cx;
[U61] 8:   x2=x*x;
[071] 9:   y2=y*y;
[VP0] 10:  it++;
[YM4] 11: } while (x2+y2<9 && it<maxit);
[000] 12:
```

Listing 1. Ein Beispiel für eine einfache Berechnung der Mandelbrotmenge

```
[304] 1: ; Programm: mandel87.asm
[0T5] 2: ; Funktion: Coprozessor-Routinen
[0U4] 3: ; zur Fraktalberechnung
[7Y2] 4: ; Sprache: TASM 2.0
[174] 5: ; Autor: Ralph Würthner
[7X4] 6: ; (c) 1992 DMV Widuch GmbH & Co. KG
[000] 7:
[8Z1] 8: MODEL LARGE
[331] 9: LOCALS
[XX0] 10: JUMPS
[AL0] 11: .8087
[000] 12:
[B35] 13: ; Auflösung des Bildschirms
[LN2] 14: SCR_X equ 320
[DN2] 15: SCR_Y equ 200
[000] 16:
[2S0] 17: .data
[000] 18:
[9T5] 19: RADIUS dw 9h ; Kontrollradius
[806] 20: ; Zwischenspeicher für Status-Wort
[NL1] 21: ; des 80x87
[U02] 22: TMP dw ?
[000] 23:
[FT0] 24: .code
[000] 25:
[P54] 26: ; ** Videomodus aendern **
[0B3] 27: PUBLIC _setgrammode
[7X2] 28: _setgrammode PROC
[U02] 29: ARG VMODE:word
[7F1] 30: push bp
[PU1] 31: mov bp,sp
[U04] 32: mov al,byte PTR VMODE
[0T1] 33: xor ah,ah
[YH1] 34: int 10h
[1D1] 35: pop bp
[3P0] 36: ret
[MP0] 37: ENDP
[000] 38:
[4C6] 39: ; eigentliches Apfelmännchenprogramm
[CL2] 40: PUBLIC _mandel
[L52] 41: _mandel PROC
[V09] 42: ARG X0:qword,D_X:qword,Y0:qword
,D_Y:qword,IT:word
[7F1] 43: push bp
[PU1] 44: mov bp,sp
[MF1] 45: push si
[PE1] 46: push di
[V92] 47: mov ax,0A000h
[KY5] 48: ; ES zeigt auf den Bildschirmspeicher
[3V1] 49: mov es,ax
[FP3] 50: ; Vergleichsradius und
[075] 51: ; oberen Rand des Ausschnitts in
[GZ3] 52: ; den Coprozessor laden
[9K2] 53: fild [RADIUS]
[YN1] 54: fld [Y0]
[UE2] 55: mov cx,[IT]
[GW1] 56: mov dl,253
[PF2] 57: mov di,SCR_Y
[LV1] 58: xor bx,bx
[NF1] 59: @@YLOOP:
```

```
[HS3] 60: ; Start der y-Schleife
[LG2] 61: mov si,SCR_X
[VN1] 62: fld [X0]
[HF1] 63: @@XLOOP:
[FS3] 64: ; Start der x-Schleife
[GW1] 65: fldz ; y2=0
[CW1] 66: fldz ; x2=0
[HR1] 67: fldz ; y=0
[DR1] 68: fldz ; x=0
[CV1] 69: mov ax,cx
[XH1] 70: @@ITLOOP:
[KA5] 71: ; Start der Iterationsschleife
[A25] 72: ; AX ist der Iterationszähler
[DA1] 73: dec ax
[KN1] 74: jz @@BRK
[3S1] 75: ; y=2xy+cy
[4N1] 76: fld st(0)
[982] 77: fmul st,st(2)
[H42] 78: fadd st,st(0)
[952] 79: fadd st,st(6)
[BT1] 80: fstp st(2)
[FS1] 81: ; x=x2-y2+cx
[6N1] 82: fld st(2)
[T82] 83: fsub st,st(4)
[552] 84: fadd st,st(5)
[UR1] 85: fst st(1)
[A71] 86: ; x2=x*x
[182] 87: fmul st,st(0)
[CT1] 88: fstp st(3)
[K71] 89: ; y2=y*y
[5N1] 90: fld st(1)
[182] 91: fmul st,st(0)
[XR1] 92: fst st(4)
[8W1] 93: ; cmp x2+y2,9
[X42] 94: fadd st,st(3)
[9U1] 95: fcomp st(7)
[662] 96: fstsw [TMP]
[XW3] 97: and BYTE PTR TMP+1,81h
[FM4] 98: ; falls Folge innerhalb des
[353] 99: ; Kreises -> weiter
[FG2] 100: jnz @@ITLOOP
[KB1] 101: inc ax
[HB1] 102: neg ax
[BR1] 103: add ax,cx
[7C1] 104: div dl
[E14] 105: ; Berechnen der Farbe aus
[2G3] 106: ; AX=Iterationstiefe
[2A1] 107: inc ah
[2A1] 108: inc ah
[YV0] 109: @@BRK:
[GY2] 110: mov [es:bx],ah
[MB1] 111: inc bx
[XG2] 112: fstp st ; x
[1H2] 113: fstp st ; y
[FK2] 114: fstp st ; x2
[LK2] 115: fstp st ; y2
[S94] 116: fadd [D_X] ; cx erhöhen
[4A1] 117: dec si
[XF2] 118: jnz @@XLOOP
[D53] 119: ; Ende der x-Schleife
[7G1] 120: fstp st
[SY4] 121: fsub [D_Y] ; cy erniedrigen
[691] 122: dec di
[1G2] 123: jnz @@YLOOP
[E53] 124: ; Ende der Y-Schleife
[2H2] 125: fstp st ; cy
[SN5] 126: ; und RADIUS vom Reg.-Stapel holen
[7G1] 127: fstp st
[GC1] 128: pop di
[ED1] 129: pop si
[1D1] 130: pop bp
[3P0] 131: ret
[MP0] 132: ENDP
[N85] 133: ; Setzen aller DAC-Register aus
[EK2] 134: ; einer Tabelle
[LM3] 135: PUBLIC _set_dac_regs
[Y73] 136: _set_dac_regs PROC
[CL2] 137: ARG BUF:dword
[7F1] 138: push bp
[PU1] 139: mov bp,sp
[D52] 140: mov ax,1012h
[LV1] 141: xor bx,bx
[VX1] 142: mov cx,256
[MG4] 143: mov dx,word PTR [BUF+2]
[6V1] 144: mov es,dx
[E34] 145: mov dx,word PTR [BUF]
[YH1] 146: int 10h
[131] 147: pop bp
[3P0] 148: ret
[MP0] 149: ENDP
[000] 150:
[HN5] 151: ; Zeichnen einer waagrechten Linie
[SC2] 152: PUBLIC _xline
[Z22] 153: _xline PROC
[R67] 154: ARG PX1:word,PY1:word,PX2:word,
COL:byte
[7F1] 155: push bp
[PU1] 156: mov bp,sp
[MF1] 157: push si
```



```
[PE1] 158:      push    di
[MC2] 159:      mov     si,[PX1]
[UB2] 160:      mov     di,[PX2]
[MS1] 161:      cmp     si,di
[Y91] 162:      jb      @e1
[7T1] 163:      xchg    si,di
[YH0] 164:      @@e1:
[WG2] 165:      mov     ax,SCR_X
[CS2] 166:      mov     bx,SCR_Y-1
[GB2] 167:      sub     bx,[PY1]
[ND1] 168:      mul     bx
[CV1] 169:      mov     bx,ax
[ZP1] 170:      add     bx,si
[5S1] 171:      sub     di,si
[LU1] 172:      mov     cx,di
[PB1] 173:      inc     cx
[V92] 174:      mov     ax,0A000h
[3V1] 175:      mov     es,ax
[GD2] 176:      mov     al,[COL]
[NF1] 177:      @@YLOOP:
[WY2] 178:      mov     [es:bx],al
[MB1] 179:      inc     bx
[CG2] 180:      loop    @@YLOOP
[GC1] 181:      pop     di
[ED1] 182:      pop     si
[LD1] 183:      pop     bp
[3P0] 184:      ret
[MP0] 185:      ENDP
[000] 186:
[AP5] 187:      ; Zeichnen einer senkrechten Linie
[VC2] 188:      PUBLIC  _yline
[Z32] 189:      _yline PROC
[S67] 190:      ARG     PX1:word,PY1:word,PY2:word,
          COL:byte
[7F1] 191:      push    bp
[PU1] 192:      mov     bp,sp
[MF1] 193:      push    si
[PE1] 194:      push    di
[RC2] 195:      mov     si,[PY1]
[XB2] 196:      mov     di,[PY2]
[MS1] 197:      cmp     si,di
[V91] 198:      ja      @e1
[7T1] 199:      xchg    si,di
[YH0] 200:      @@e1:
[CS2] 201:      mov     bx,SCR_Y-1
[ET1] 202:      sub     bx,si
[WG2] 203:      mov     ax,SCR_X
[ND1] 204:      mul     bx
[W82] 205:      add     ax,[PX1]
[CV1] 206:      mov     bx,ax
[MS1] 207:      sub     si,di
[V21] 208:      mov     cx,si
[PB1] 209:      inc     cx
[V92] 210:      mov     ax,0A000h
[3V1] 211:      mov     es,ax
[GD2] 212:      mov     al,[COL]
[HF1] 213:      @@XLOOP:
[WY2] 214:      mov     [es:bx],al
[XC2] 215:      add     bx,SCR_X
[8G2] 216:      loop    @@XLOOP
[GC1] 217:      pop     di
[ED1] 218:      pop     si
[LD1] 219:      pop     bp
[3P0] 220:      ret
[MP0] 221:      ENDP
[000] 222:
[7N5] 223:      ; Bewegungsfläche der Maus setzen
[6P3] 224:      PUBLIC  _setmousesize
[H93] 225:      _setmousesize PROC
[0N6] 226:      ARG     XU:word,XO:word,YU:word,YO:
          word
[7F1] 227:      push    bp
[PU1] 228:      mov     bp,sp
[X52] 229:      mov     ax,0007h
[VF2] 230:      mov     cx,[XU]
[BF2] 231:      mov     dx,[XO]
[BK1] 232:      int     33h
[162] 233:      mov     ax,0008h
[XF2] 234:      mov     cx,[YU]
[NB1] 235:      neg     cx
[0D2] 236:      add     cx,SCR_Y
[DF2] 237:      mov     dx,[YO]
[RB1] 238:      neg     dx
[2D2] 239:      add     dx,SCR_Y
[BK1] 240:      int     33h
[LD1] 241:      pop     bp
[3P0] 242:      ret
[MP0] 243:      ENDP
[000] 244:
[598] 245:      ; Mauszeiger auf eine bestimmte Position se
          tzen
[5Y2] 246:      PUBLIC  _setmouse
[GH2] 247:      _setmouse PROC
[TT3] 248:      ARG     PX:word,PY:word
[7F1] 249:      push    bp
[PU1] 250:      mov     bp,sp
```

```
[H52] 251:      mov     ax,0004h
[MF2] 252:      mov     cx,[PX]
[TF2] 253:      mov     dx,[PY]
[RB1] 254:      neg     dx
[2D2] 255:      add     dx,SCR_Y
[BK1] 256:      int     33h
[LD1] 257:      pop     bp
[3P0] 258:      ret
[MP0] 259:      ENDP
[000] 260:
[2T6] 261:      ; x-Koordinate des Mauszeigers ermitteln
[643] 262:      PUBLIC  _getxmouse
[5S2] 263:      _getxmouse PROC
[D52] 264:      mov     ax,0003h
[BK1] 265:      int     33h
[CV1] 266:      mov     ax,cx
[3P0] 267:      ret
[MP0] 268:      ENDP
[000] 269:
[SU6] 270:      ; y-Koordinate des Mauszeigers ermitteln
[843] 271:      PUBLIC  _getymouse
[7S2] 272:      _getymouse PROC
[D52] 273:      mov     ax,0003h
[BK1] 274:      int     33h
[HF2] 275:      sub     dx,SCR_Y
[RB1] 276:      neg     dx
[DV1] 277:      mov     ax,dx
[3P0] 278:      ret
[MP0] 279:      ENDP
[000] 280:
[X16] 281:      ; Stellung der Maustasten ermitteln
[343] 282:      PUBLIC  _getbottom
[2S2] 283:      _getbottom PROC
[D52] 284:      mov     ax,0003h
[BK1] 285:      int     33h
[BV1] 286:      mov     ax,bx
[3P0] 287:      ret
[MP0] 288:      ENDP
[000] 289:
[HG3] 290:      ; Maus zuruecksetzen
[9E3] 291:      PUBLIC  _resetmouse
[G03] 292:      _resetmouse PROC
[GV1] 293:      xor     ax,ax
[BK1] 294:      int     33h
[3P0] 295:      ret
[MP0] 296:      ENDP
[000] 297:
[4M0] 298:      END
```

Listing 2. »mandel87.asm« zeigt die Routinen für Hochgeschwindigkeits-Apfelmännchen.

```
[P93] 1:  /* Programm: mandel.c
[WX5] 2:  Funktion: zeichnet Mandelbrotmenge
[N53] 3:  Sprache: Turbo C++,
[544] 4:  diverse andere Compiler
[V74] 5:  Autor: Ralph Würthner
[HN4] 6:  (c) 1992 DMV Widuch GmbH & Co. KG
[000] 7:
[195] 8:  Dieses Programm läuft nur mit:
[000] 9:
[392] 10: - Koprozessor
[DZ4] 11: - Microsoft-kompatibler Maus
[3W1] 12: - VGA-Karte
[000] 13:
[ZT5] 14: Hinweis: mit Speichermodell LARGE
[KC2] 15: kompilieren */
[000] 16:
[203] 17: #include <conio.h>
[VG2] 18: #include <io.h>
[R03] 19: #include <fcntl.h>
[U83] 20: #include <stdlib.h>
[L13] 21: #include <stdio.h>
[6A3] 22: #include <string.h>
[000] 23:
[PM3] 24: /* Standardwerte für
[SE4] 25: den Ausschnitt und maxit */
[ZU3] 26: #define START_XMIN -3.0
[0V3] 27: #define START_XMAX 3.0
[YW3] 28: #define START_YMIN -2.25
[ZW3] 29: #define START_YMAX 2.25
[3U3] 30: #define START_MAXIT 50
[D15] 31: /* Auflösung 320x200, 256 Farben */
[NE3] 32: #define SCR_MODE 0x13
[XG3] 33: #define TXT_MODE 0x03
[D25] 34: /* Auflösung des Bildschirms */
[0N2] 35: #define SCR_X 320
[WM2] 36: #define SCR_Y 200
[MM6] 37: /* Seitenverhältnis des Bildschirms */
```

Listing 3. »mandel.c« malt Apfelmännchen in 256 Farben.


```
[104] 38: #define ASPECTRATIO 0.75
[SK3] 39: /* Zeiger auf Anfang des
[3U3] 40:   Bildschirmspeichers */
[4C4] 41: #define SCR_ADR 0xA0000000L
[000] 42:
[HT5] 43: /* Routinen des Assemblermoduls */
[000] 44:
[8F4] 45: void setgramode(int mode);
[FW5] 46: void xline(unsigned x0,unsigned y0,
[U04] 47:   unsigned x1,int color);
[KW5] 48: void yline(unsigned x0,unsigned y0,
[X04] 49:   unsigned y1,int color);
[PA5] 50: void set_dac_regs(char *buffer);
[L95] 51: void mandel(double x0,double dx,
[8L4] 52:   double y0,double dy,int it);
[HL5] 53: void setmouse(int xu,int xo,
[CG2] 54:   int yu,int yo);
[PY4] 55: void setmouse(int px,int py);
[5G3] 56: int getxmouse(void);
[6G3] 57: int getymouse(void);
[AG3] 58: int getbottom(void);
[AW3] 59: void resetmouse(void);
[000] 60:
[B25] 61: /* Routinen des Hauptmoduls */
[000] 62:
[4A3] 63: void graphic(void);
[DN2] 64: void text(void);
[E87] 65: void paint_mandel(double xmin,double xmax,
[T26] 66:   double ymin,double ymax,int maxit);
[CT6] 67: void fensterzeich(int px,int py,int dx,
[YA1] 68:   int dy);
[FY5] 69: void zoom(double *xmin,double *xmax,
[3L4] 70:   double *ymin,double *ymax);
[987] 71: void ausschnitt(double *xmin,double *xmax,
[3L4] 72:   double *ymin,double *ymax);
[6B5] 73: void maxit_aendern(int *maxit);
[0Z5] 74: void laden(double *xmin,double *xmax,
[M76] 75:   double *ymin,double *ymax,int *maxit);
[BN6] 76: void speichern(double xmin,double xmax,
[T26] 77:   double ymin,double ymax,int maxit);
[GM2] 78: void menu(void);
[000] 79:
[FE3] 80: /* Globale Variablen */
[000] 81:
[WX4] 82: char *screen,*buffer,pic_flag;
[KY6] 83: /* Anz. der Einträge in der Farbtabelle */
[HX2] 84: #define TABSIZE 13
[LD6] 85: char rtab[TABSIZE]={30,18,63,20,50, 30, 63,
[S03] 86:   30, 30, 63, 60, 30, 30};
[YA6] 87: char gtab[TABSIZE]={40,10,50,63, 8, 30, 10,
[G03] 88:   50, 30, 63, 8, 60, 40};
[S86] 89: char btab[TABSIZE]={20,63, 5,30,63, 30, 8,
[C03] 90:   10, 63, 63, 40, 7, 20};
[NF6] 91: int otab[TABSIZE]={ 2,22,44,66,88,110,132,
[563] 92:   154,176,198,220,242,255};
[000] 93:
[F15] 94: /* in den Grafikmodus wechseln */
[S23] 95: void graphic(void)
[NH3] 96: { char dcregs[256*3];
[VB1] 97:   int n,m;
[VD1] 98:   double s;
[G15] 99:   /* Setzen aller Farbregister */
[CK4] 100:   dcregs[0]=0; dcregs[1]=0;
[SP4] 101:   dcregs[2]=0; dcregs[3]=63;
[ST4] 102:   dcregs[4]=63; dcregs[5]=63;
[KR3] 103:   for (m=1;m<TABSIZE;m++)
[BL5] 104:     {for (n=otab[m-1];n<otab[m];n++)
[574] 105:       {s=(double)(n-otab[m-1])/
[UG3] 106:         (otab[m]-otab[m-1]);
[5X3] 107:         dcregs[n*3]=rtab[m-1]+
[ED3] 108:           s*(rtab[m]-rtab[m-1]);
[UY3] 109:         dcregs[n*3+1]=gtab[m-1]+
[SC3] 110:           s*(gtab[m]-gtab[m-1]);
[RY3] 111:         dcregs[n*3+2]=btab[m-1]+
[Z23] 112:           s*(btab[m]-btab[m-1]);}}
[2N3] 113:   setgramode(SCR_MODE);
[4S3] 114:   set_dac_regs(dcregs);
[375] 115:   /* Bild in den Bildschirmspeicher
[8V1] 116:     kopieren */
[CA6] 117:   memcpy(screen,buffer,SCR_X*SCR_Y);}
[7Z4] 118: /* in den Textmodus wechseln */
[KL2] 119: void text(void)
[PL4] 120: { /* Bild wieder in den Haupt-
[593] 121:   speicher kopieren */
[PW5] 122:   memcpy(buffer,screen,SCR_X*SCR_Y);
[A24] 123:   setgramode(TXT_MODE);}
[LV5] 124: /* Diese Funktion ruft mandel auf und
[2P5] 125:   zeichnet dann das Apfelmännchen */
[LR6] 126: void paint_mandel(xmin,xmax,ymin,ymax,
[911] 127:   maxit)
[XT4] 128: { double xmin,xmax,ymin,ymax;
[BU1] 129:   int maxit;
[WV1] 130:   { graphic();
[D26] 131:     mandel(xmin,(xmax-xmin)/SCR_X,ymax,
[2A4] 132:       (ymax-ymin)/SCR_Y,maxit);
[R61] 133:     getch();
[271] 134:     text();
[M02] 135:     pic_flag=0;}
```

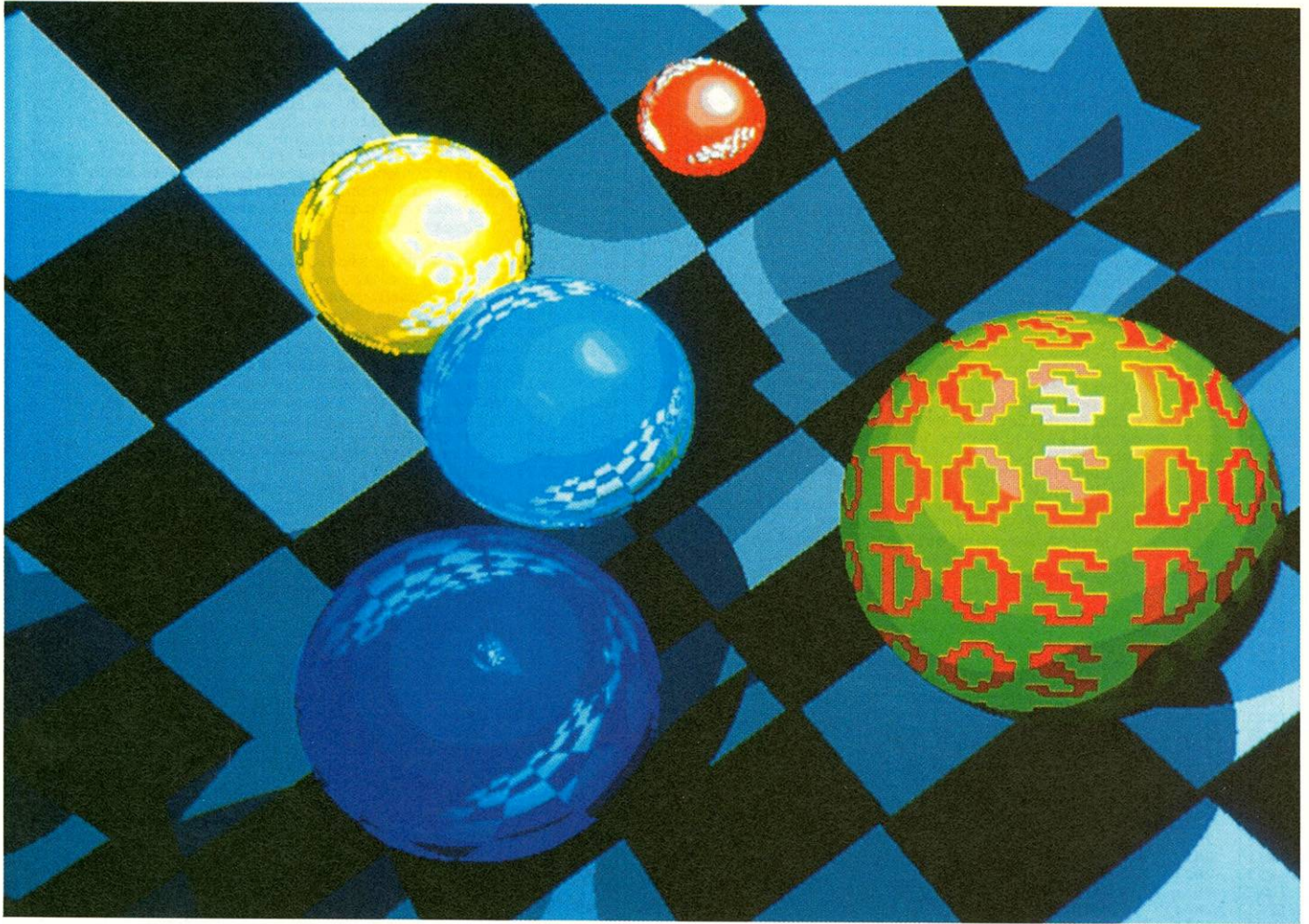
```
[H45] 136: /* Zeichnen des Zoomfensters */
[G45] 137: void fensterzeich(px,py,dx,dy)
[NR2] 138: { int px,py,dx,dy;
[0K4] 139:   {xline(px-dx,py-dy,px+dx,1);
[KH4] 140:     xline(px-dx,py+dy,px+dx,1);
[1K4] 141:     yline(px-dx,py-dy,py+dy,1);
[UN4] 142:     yline(px+dx,py-dy,py+dy,1);}
[RE7] 143: /* Diese Funktion ermittelt die Koordinaten
[YD5] 144:   des gewünschten Ausschnitts */
[M85] 145: void zoom(xmin,xmax,ymin,ymax)
[M85] 146: { double *xmin,*xmax,*ymin,*ymax;
[RY3] 147:   {int px,py,dx,dy,x1,x2;
[2G2] 148:     double dre,dim;
[Z21] 149:     if (pic_flag)
[H86] 150:       {printf("**** Fehler: Es befindet sich \
[5V4] 151:         noch kein Bild im Speicher!\n");
[8M6] 152:       printf("Weiter mit beliebiger Taste.\a");}
[R61] 153:       getch();
[FG1] 154:       return;}
[UL1] 155:   graphic();
[4G4] 156:   /* Position und Größe des
[9B3] 157:     Fenters festlegen */
[234] 158:   px=SCR_X/2; py=SCR_Y/2;
[Z34] 159:   dx=SCR_X/10; dy=SCR_Y/10;
[E85] 160:   setmouse(dx,SCR_X-1-dx,dy,
[KX1] 161:     SCR_Y-1-dy);
[PW2] 162:   setmouse(px,py);
[DB3] 163:   /* Fenster zeichnen */
[3E4] 164:   fensterzeich(px,py,dx,dy);
[IP3] 165:   while (!(getbottom() & 1))
[7X6] 166:     {while (px==getxmouse() && py==getymouse())
[KU2] 167:       && getbottom()==0);
[7X2] 168:     if (getbottom() & 2)
[7T5] 169:       { /* Verändern der Fenstergröße */
[HM3] 170:         x1=min(px,SCR_X-px-1);
[AL3] 171:         x2=min(py,SCR_Y-py-1)*
[MM3] 172:           (double)SCR_X/SCR_Y;
[BY4] 173:         setmouse(x1,min(x1,x2),-1,1);
[FN2] 174:         setmouse(dx,0);
[WD3] 175:         while (getbottom() & 2)
[B74] 176:           {while (dx==getxmouse() &&
[HG2] 177:             getbottom() & 2);
[X73] 178:             /* Fenster löschen */
[SU5] 179:             memcpy(screen,buffer,SCR_X*SCR_Y);
[3R2] 180:             dx=getxmouse();
[DH4] 181:             dy=dx*(double)SCR_Y/SCR_X;
[DB3] 182:             /* Fenster zeichnen */
[RX4] 183:             fensterzeich(px,py,dx,dy);}
[E85] 184:         setmouse(dx,SCR_X-1-dx,dy,
[KX1] 185:           SCR_Y-1-dy);
[G43] 186:         setmouse(px,py);}
[X73] 187:       /* Fenster löschen */
[SU5] 188:       memcpy(screen,buffer,SCR_X*SCR_Y);
[XG5] 189:       px=getxmouse(); py=getymouse();
[TD4] 190:       /* Fenster wieder zeichnen */
[RX4] 191:       fensterzeich(px,py,dx,dy);}
[X73] 192:     /* Fenster löschen */
[SU5] 193:     memcpy(screen,buffer,SCR_X*SCR_Y);
[X86] 194:     /* Berechnen der Ausschnittsgrenzen */
[UK4] 195:     dre=(xmax-xmin)*dx/SCR_X;
[NM3] 196:     dim=dre*ASPECTRATIO;
[PN4] 197:     *xmin=*xmin+(xmax-xmin)*px/
[AR1] 198:       SCR_X-dre;
[FW2] 199:     *xmax=*xmin+2*dre;
[YH6] 200:     *ymin=*ymin+(ymax-ymin)*py/SCR_Y-dim;
[UW2] 201:     *ymax=*ymin+2*dim;
[271] 202:     text();
[R02] 203:     pic_flag=1;}
[316] 204: /* Diese Funktion fragt den Benutzer
[9V3] 205:   nach einem Ausschnitt */
[WD6] 206: void ausschnitt(xmin,xmax,ymin,ymax)
[M85] 207: { double *xmin,*xmax,*ymin,*ymax;
[BX2] 208:   {double xi,xa,yi;
[6L5] 209:     printf("*** Ausschnitt wählen *\n");
[365] 210:     printf("xmin= "); scanf("%lf",&xi);
[X45] 211:     printf("xmax= "); scanf("%lf",&xa);
[3R1] 212:     if (xi>xa)
[5R6] 213:       {printf("**** Fehler: xmin muß kleiner \
[653] 214:         sein als xmax!\n");
[8M6] 215:       printf("Weiter mit beliebiger Taste.\a");}
[R61] 216:       getch();
[FG1] 217:       return;}
[665] 218:     printf("ymin= "); scanf("%lf",&yi);
[BK1] 219:     *xmin=xi;
[FK1] 220:     *xmax=xa;
[KK1] 221:     *ymin=yi;
[N05] 222:     *ymax=yi+(xa-xi)*ASPECTRATIO;
[Z15] 223:     printf("ymax= %12.10lf\n",*ymax);
[LF6] 224:     printf("Weiter mit beliebiger Taste...");
[R61] 225:     getch();
[R02] 226:     pic_flag=1;}
[4T6] 227: /* Diese Funktion fragt den Benutzer nach
[555] 228:   einer neuen Iterationstiefe */
[7B4] 229: void maxit_aendern(maxit)
[XZ1] 230: { int *maxit;
[3G1] 231:   {int it;
[L66] 232:     printf("*** Maximale Iterationstiefe \
[XR4] 233:       ändern *\n\nNeuer Wert: ");
[D52] 234:     scanf("%i",&it);}
```



```
[EB3] 235: if (it<1 || it>32000)
[D57] 236: {printf("**** Fehler: Der neue Wert muß \
[LV5] 237: im Bereich 1<=maxit<=32000 liegen!\n");
[8M6] 238: printf("Weiter mit beliebiger Taste.\a");
[R61] 239: getch();
[FG1] 240: return;}
[F52] 241: *maxit=it;}
[K05] 242: /* Diese Funktion lädt ein Bild
[5S2] 243: in den Speicher */
[H96] 244: void laden(xmin,xmax,ymin,ymax,maxit)
[M85] 245: double *xmin,*xmax,*ymin,*ymax;
[XZ1] 246: int *maxit;
[9S2] 247: {char name[40];
[E02] 248: int handle;
[VZ5] 249: printf("*** Bild laden *\n\nName des \
[WG3] 250: Bildes (ohne .SCR): ");
[SC2] 251: scanf("%s",name);
[843] 252: strcat(name, ".SCR");
[F46] 253: if ((handle= open(name,O_RDONLY))==-1)
[GG5] 254: {printf("**** Fehler: Die Datei %s \
[A66] 255: kann nicht geöffnet werden!\n",name);
[8M6] 256: printf("Weiter mit beliebiger Taste.\a");
[R61] 257: getch();
[FG1] 258: return;}
[MF6] 259: if (_read(handle,buffer,SCR_X*SCR_Y)!=
[192] 260: SCR_X*SCR_Y)
[FF6] 261: {printf("**** Fehler: Lesen der Daten \
[N63] 262: nicht möglich!\n");
[8M6] 263: printf("Weiter mit beliebiger Taste.\a");
[R61] 264: getch();
[WG2] 265: _close(handle);
[UW1] 266: pic_flag=1;
[FG1] 267: return;}
[G06] 268: if (_read(handle,xmin,sizeof *xmin)!=
[NZ2] 269: sizeof *xmin ||
[GG5] 270: _read(handle,xmax,sizeof *xmin)!=
[NZ2] 271: sizeof *xmin ||
[7G5] 272: _read(handle,ymin,sizeof *xmin)!=
[NZ2] 273: sizeof *xmin ||
[LG5] 274: _read(handle,ymax,sizeof *xmin)!=
[NZ2] 275: sizeof *xmin ||
[2S5] 276: _read(handle,maxit,sizeof *maxit)!=
[HB2] 277: sizeof *maxit)
[FF6] 278: {printf("**** Fehler: Lesen der Daten \
[N63] 279: nicht möglich!\n");
[8M6] 280: printf("Weiter mit beliebiger Taste.\a");
[R61] 281: getch();
[W92] 282: _close(handle);
[UW1] 283: pic_flag=1;
[FG1] 284: return;}
[YF3] 285: if (_close(handle)==-1)
[VB6] 286: {printf("**** Fehler: Die Datei %s kann \
[UY5] 287: nicht geschlossen werden!\n",name);
[8M6] 288: printf("Weiter mit beliebiger Taste.\a");
[R61] 289: getch();
[FG1] 290: return;}
[RW1] 291: pic_flag=0;
[FG1] 292: return;}
[8T6] 293: /* Diese Funktion speichert ein Bild ab */
[X27] 294: void speichern(xmin,xmax,ymin,ymax,maxit)
[XT4] 295: double xmin,xmax,ymin,ymax;
[BU1] 296: int maxit;
[9S2] 297: {char name[40];
[E02] 298: int handle;
[ZZ1] 299: if (pic_flag)
[H86] 300: {printf("**** Fehler: Es befindet sich \
[5V4] 301: noch kein Bild im Speicher!\n");
[8M6] 302: printf("Weiter mit beliebiger Taste.\a");
[R61] 303: getch();
[FG1] 304: return;}
[2S6] 305: printf("*** Bild speichern *\n\nName des \
[WG3] 306: Bildes (ohne .SCR): ");
[SC2] 307: scanf("%s",name);
[843] 308: strcat(name, ".SCR");
[PS4] 309: if ((handle= creat(name,0))==-1)
[VB6] 310: {printf("**** Fehler: Die Datei %s kann \
[685] 311: nicht erzeugt werden!\n",name);
[8M6] 312: printf("Weiter mit beliebiger Taste.\a");
[R61] 313: getch();
[FG1] 314: return;}
[0H6] 315: if (_write(handle,buffer,SCR_X*SCR_Y)!=
[192] 316: SCR_X*SCR_Y)
[P16] 317: {printf("**** Fehler: Schreiben der \
[XA4] 318: Daten nicht möglich!\n");
[8M6] 319: printf("Weiter mit beliebiger Taste.\a");
[R61] 320: getch();
[WG2] 321: _close(handle);
[FG1] 322: return;}
[356] 323: if (_write(handle,&xmin,sizeof xmin)!=
[FN2] 324: sizeof xmin ||
[7N5] 325: _write(handle,&xmax,sizeof xmin)!=
[FN2] 326: sizeof xmin ||
[SM5] 327: _write(handle,&ymin,sizeof xmin)!=
[FN2] 328: sizeof xmin ||
[8N5] 329: _write(handle,&ymax,sizeof xmin)!=
[FN2] 330: sizeof xmin ||
[SY5] 331: _write(handle,&maxit,sizeof maxit)!=
[Z52] 332: sizeof maxit)
```

```
[P16] 333: {printf("**** Fehler: Schreiben der \
[XA4] 334: Daten nicht möglich!\n");
[8M6] 335: printf("Weiter mit beliebiger Taste.\a");
[R61] 336: getch();
[WG2] 337: _close(handle);
[FG1] 338: return;}
[YF3] 339: if (_close(handle)==-1)
[VB6] 340: {printf("**** Fehler: Die Datei %s kann \
[UY5] 341: nicht geschlossen werden!\n",name);
[8M6] 342: printf("Weiter mit beliebiger Taste.\a");
[HE1] 343: getch();
[FG1] 344: return;}
[5G6] 345: /* Diese Funktion realisiert das Menü */
[NK2] 346: void menu(void)
[F15] 347: {double xmin,xmax,ymin,ymax;
[BU1] 348: int maxit;
[R42] 349: char input=0;
[X56] 350: xmin=START_XMIN; xmax=START_XMAX;
[566] 351: ymin=START_YMIN; ymax=START_YMAX;
[BC3] 352: maxit=START_MAXIT;
[BM2] 353: while (input!='0')
[3T1] 354: {clrscr();
[W21] 355: gotoxy(27,1);
[NV5] 356: printf("*** Fraktal-Explorer *\n\n");
[P21] 357: gotoxy(22,2);
[DC5] 358: printf("(c) 1992 DMV Widuch GmbH & \
[X61] 359: Co. KG");
[ZZ1] 360: gotoxy(32,4);
[PL4] 361: printf("Aktuelle Werte:\n");
[YZ1] 362: gotoxy(20,5);
[5T4] 363: printf("xmin= %12.10lf",xmin);
[055] 364: printf(" xmax= %12.10lf",xmax);
[202] 365: gotoxy(20,6);
[AT4] 366: printf("ymin= %12.10lf",ymin);
[555] 367: printf(" ymax= %12.10lf",ymax);
[1A5] 368: gotoxy(20,7); printf("Maximale ");
[856] 369: printf("Iterationstiefe: %i, ",maxit);
[D16] 370: printf("%s\n",pic_flag?"kein Bild \
[6T3] 371: geladen":"Bild geladen");
[Z65] 372: gotoxy(37,9); printf("Menü:\n");
[742] 373: gotoxy(22,11);
[FG4] 374: printf("< 1 > Zeichnen\n");
[842] 375: gotoxy(22,12);
[4V3] 376: printf("< 2 > Zoom\n");
[942] 377: gotoxy(22,13);
[8V5] 378: printf("< 3 > Maximale Iterations \
[303] 379: tiefe ändern\n");
[A42] 380: gotoxy(22,14);
[ED6] 381: printf("< 4 > Ausschnitt wählen\n");
[B42] 382: gotoxy(22,15);
[M95] 383: printf("< 5 > Bild speichern\n");
[C42] 384: gotoxy(22,16);
[BH4] 385: printf("< 6 > Bild laden\n");
[D42] 386: gotoxy(22,17);
[HT5] 387: printf("< 0 > Programm beenden\n");
[C42] 388: gotoxy(21,18);
[1F6] 389: printf("<SPACE> Bild betrachten\n\n");
[DC2] 390: input=getch();
[XA2] 391: switch (input)
[RG6] 392: {case '1': paint_mandel(xmin,xmax,ymin,
[XZ1] 393: ymax,maxit);
[P01] 394: break;
[396] 395: case '2': zoom(&xmin,&xmax,&ymin,&ymax);
[P01] 396: break;
[E55] 397: case '3': maxit_aendern(&maxit);
[P01] 398: break;
[W86] 399: case '4': ausschnitt(&xmin,&xmax,&ymin,
[371] 400: &ymax);
[P01] 401: break;
[4P6] 402: case '5': speichern(xmin,xmax,ymin,ymax,
[N81] 403: maxit);
[P01] 404: break;
[296] 405: case '6': laden(&xmin,&xmax,&ymin,&ymax,
[W71] 406: &maxit);
[P01] 407: break;
[324] 408: case '0': /* Dummy-Case */
[P01] 409: break;
[486] 410: case ' ': {graphic(); getch(); text(); }
[P01] 411: break;
[GK4] 412: default: printf("\a"); }}
[DT1] 413: void main()
[Z52] 414: {unsigned n;
[LP3] 415: setgramode(TXT_MODE);
[S62] 416: resetmouse();
[EW4] 417: buffer=malloc(SCR_X*SCR_Y);
[7S2] 418: if (buffer==NULL)
[925] 419: {printf("**** Fehler: Zu wenig \
[6H3] 420: Speicher vorhanden! \
[LC4] 421: Programm abgebrochen.\a");
[FG1] 422: return;}
[MT6] 423: for (n=0;n<(unsigned)SCR_X*SCR_Y;n++)
[742] 424: buffer[n]=0;
[5Z3] 425: screen=(void *)SCR_ADR;
[UW1] 426: pic_flag=1;
[A91] 427: menu();}
```

Listing 3. »mandel.c« malt Apfelmännchen in 256 Farben.



Raytracing in C

Schöne neue Welt

Realistisch aussehende Computergrafiken werden immer häufiger in Spielfilmen und in der Werbung eingesetzt. Sie entstehen durch Anwendung eines Verfahrens, das sich Raytracing nennt. Im folgenden werden Sie die Grundlagen dieser Methode am Beispiel eines vollständigen Raytracers kennenlernen.

Mit Raytracing wird ein Verfahren bezeichnet, das es Ihnen gestattet, extrem realistische Bilder von einer nur mathematisch vorhandenen Welt darzustellen. Bei Anwendung dieser Technik sind Sie in der Lage, mit spiegelnden und transparenten Materialien zu arbeiten und auf diese Weise sehr eindrucksvolle Szenarien zu produzieren. Da Raytracing derzeit die beste Simulation eines fiktiven Universums ist, sind nahezu alle professionellen Computergrafiken für Film und Fernsehen mit der Raytracing-Methode

hergestellt worden. So verwenden zum Beispiel auch Filme wie Star Wars oder Star Trek Raytracing-Bilder.

Raytracing kann auf deutsch am besten mit Lichtstrahlverfolgung übersetzt werden. Die grundlegende Idee dabei ist, daß rechnerisch der Weg eines Lichtstrahls – einer mathematischen Lichtquelle – durch einen mathematischen Raum verfolgt wird. Sobald dieser Lichtstrahl auf ein Objekt im Raum trifft, wird er entweder absorbiert oder reflektiert, oder er geht durch das Objekt hindurch. Auch eine Kombination aus diesen drei Alternativen kann vorkommen.

Im selben mathematischen Raum befindet sich auch ein Beobachter, der die Objekte – beleuchtet mit den Lichtquellen – durch ein Beobachtungsfenster betrachtet. Beim Raytracing stellt Ihr Monitorbild das Fenster dar, durch das Sie dieses mathematische Universum in Augenschein nehmen (Bild 1). Dabei wird jeder Punkt, den Sie sehen, als Zusammenspiel von Objekten und Lichtstrahlen aufgefaßt.

Ein Punkt auf der Oberfläche eines Objekts wird durch verschiedene Lichtquellen beleuchtet: Da ist zunächst ein allgemein in der Umgebung vorhandenes diffuses Licht (Umgebungslicht). Dann existieren reflektierte Lichtstrahlen von anderen Oberflächen. Weiterhin gibt es Lichtstrahlen, die durch das Objekt hindurchgehen, und schließlich solche, die direkt von einer Lichtquelle auf den Oberflächenpunkt fallen. Die gesamte Berechnung aller Lichtwege und Lichtintensitäten erfordert einen nicht unerheblichen Rechen- und damit auch Zeitaufwand. Üblicherweise warten Sie deshalb auf ein Raytracing-Bild mehrere Minuten, Stunden oder sogar Tage. Das hängt von der Anzahl der Objekte, deren Material, der Anzahl und Art der Lichtquellen und der Komplexität des ganzen Szenariums ab.

Um die mathematischen Objekte im Raum zu definieren und zu konstruieren, eignen sich am besten ein Editor, der

auf der Grundlage eines CAD-Programms arbeitet, und ein einfaches Malprogramm, mit dessen Hilfe Sie die Oberflächenstrukturen anfertigen. Im vorliegenden Projekt sind alle drei Komponenten vorhanden. Sie erhalten ein vollständiges, leistungsstarkes Werkzeug, das es Ihnen erlaubt, in Minutenschnelle neue Welten aufzubauen. Zunächst lernen Sie nun die Grundlagen des Raytracings kennen. Danach folgt eine Einführung in das Arbeiten mit dem Raytracer und seinen Editoren.

Der Raytracing-Algorithmus

Da der Bildschirm das Fenster zur darzustellenden mathematischen Welt ist, werden Farbe und Intensität eines jeden Pixels (Bildpunkts) dadurch berechnet, daß der Lichtstrahl, der vom Auge des Betrachters durch das Zentrum dieses Bildpunktes geht, in den mathematischen Raum auf der anderen Seite zurückverfolgt wird (Bild 2). Falls der Lichtstrahl auf ein Objekt trifft, werden lokale Farbe und Intensität des Oberflächenpunktes errechnet. Sie hängen von der Reflexion der direkten und indirekten Beleuchtung des Szenariums ab, sowie von Farbe und Beschaffenheit der Objekt-oberfläche.

Falls das Material an besagtem Punkt transparente oder spiegelnde Eigenschaften besitzt, sind auch die Lichtstrahlen, die durch diese Eigenschaften in Intensität und Farbe des Bildpunktes eingehen, bestimmt. Solche Lichtstrahlen können wiederum auf Objekte mit spiegelnden oder transparenten Eigenschaften stoßen, die dann auch wieder zurückverfolgt werden müssen. Dieses Phänomen werden Sie später noch genauer kennenlernen. Trifft einer der Lichtstrahlen kein Objekt, so wird die Hintergrundfarbe des Szenariums als seine Intensität und Farbe angenommen. Die so eben skizzierten Überlegungen treffen für jeden Bildpunkt unseres Bildschirms zu und gehen in alle entsprechenden Berechnungen ein.

Ein Vorteil des Raytracings ist, daß Sie keinen Hidden-Line-Algorithmus benötigen, wie er bei anderen dreidimensionalen Verfahren erforderlich ist. Das bedeutet, Punkte und Kanten von Objekten, die Sie nicht auf dem Bildschirm sehen, weil andere Objekte sie verdecken, werden erst gar nicht berechnet.

Rekursive Implementation des Raytracing

Wie weiter oben erwähnt, wird ein Lichtstrahl vom Auge des Betrachters in den mathematischen Raum verfolgt, bis er entweder auf ein Objekt trifft oder in den Hintergrund geht. Falls er ein transparentes oder spiegelndes Objekt trifft, sind

weitere Lichtstrahlen von diesem Schnittpunkt aus mit anderer Richtung zu verfolgen (Bild 3).

Für dieses nicht ganz einfache Problem bietet sich eine rekursive Implementation der Strahlenrückverfolgung an. Diese rekursive Funktion wird im weiteren mit »raytrace()« bezeichnet. Als Funktionsparameter werden ihr Startpunkt und Richtung der Halbgeraden übergeben, die den Lichtstrahl definiert. Mit Aufruf der Funktion wird der Lichtstrahl so lange verfolgt, bis er entweder auf den Hintergrund oder auf ein Objekt stößt.

Trifft er auf den Hintergrund, ist die Strahlenverfolgung beendet. Trifft er auf ein Objekt, so werden die beiden durch Transparenz und Reflexion des Objekts entstehenden Lichtstrahlen rekursiv weiter zurückverfolgt, und zwar durch erneuten Aufruf von »raytrace()«. Bei diesem Aufruf wird der Schnittpunkt der Geraden mit dem Objekt zum neuen Startpunkt. Die Richtung ist abhängig vom Einfallswinkel des ursprünglichen Lichtstrahls auf die Oberfläche des Objekts (Bild 3).

Mit diesem rekursiven Verfahren lassen sich praktisch alle Reflexions- und Transparenzeffekte berechnen. Es ist jedoch wichtig, die Schachtelungstiefe der Rekursion zu kontrollieren. Stellen Sie sich vor, in einem Raum sind zwei perfekte Spiegel einander genau gegenüber aufgestellt. Sie sitzen als (lichtdurchlässiger) Betrachter zwischen diesen Spiegeln, mit Blickrichtung in einen der beiden. Sie sehen in diesem Fall natürlich überhaupt nichts im Spiegel, da jeder Lichtstrahl immer perfekt reflektiert wird und so nie auf ein Objekt stößt. Wenden Sie nun einen unkontrollierten rekursiven Algorithmus auf dieses Problem an, so endet dieser nie, da sich immer der eine Spiegel im anderen spiegelt und umgekehrt. Bei einer Implementation führt dies unweigerlich zu einem Überlauf des Stacks, auf dem die Zwischenergebnisse der Rekursionen abgelegt werden, was wiederum einen Rechnerabsturz nach sich zieht.

Um dieses Phänomen in den Griff zu bekommen, wird der Funktion »raytrace()« ein weiterer Parameter mitgegeben, der die Schachtelungstiefe darstellt. Vor jedem rekursiven Aufruf wird er zunächst um eins erhöht. Beim Durchlaufen der Funktion wird er dann mit der globalen erlaubten Schachtelungstiefe verglichen. Ist sie erreicht, finden keine weiteren rekursiven Aufrufe mehr statt, statt dessen wird die Farbe schwarz zurückgeliefert.

Sie lernen nun einige der wichtigsten geometrischen Grundlagen kennen:

Alle Szenarien, die Sie mit Raytracing berechnen, werden in einem dreidimensionalen mathematischen Raum dargestellt. Dieser Raum hat die drei Koordinatenachsen x, y und z. Um einen bestimmten Punkt in diesem Raum anzugeben genügt es, die Abschnitte der Koordinatenachsen für diesen

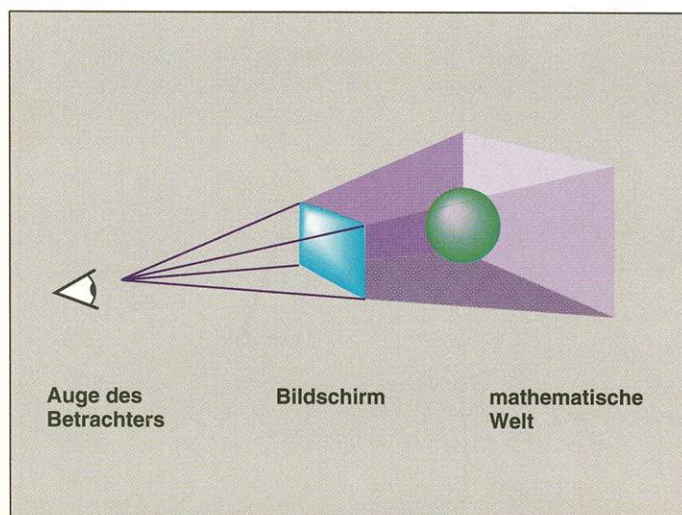


Bild 1. Ihr Monitor ist das Fenster in eine neue Welt.

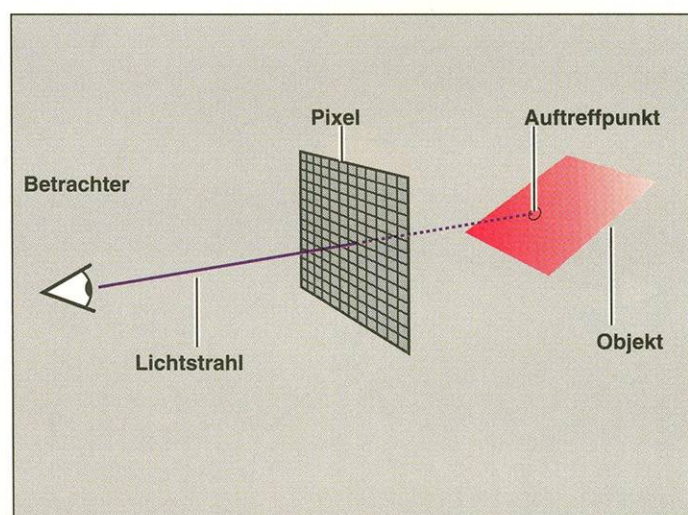


Bild 2. Eine Gerade, vom Betrachter ausgehend

Punkt anzugeben. Der Punkt, den Sie in Bild 4 sehen, hat den x-Achsenabschnitt 3, den y-Achsenabschnitt 4 und den z-Achsenabschnitt 5. Üblicherweise werden diese Angaben auf folgende Weise zusammengefaßt: (x-Wert, y-Wert, z-Wert). Außer der Lokalisierung eines Punktes im Raum wird mit dieser Angabe auch ein sogenannter Vektor spezifiziert. Der Vektor ist in diesem Fall eine Richtungsangabe im dreidimensionalen Raum, und zwar ausgehend vom Ursprung (0, 0, 0), hin zum spezifizierten Punkt (3,4,5). Richtungsvektoren werden durch Pfeile dargestellt. Um eine Halbgerade zu definieren sind genau zwei Angaben notwendig: entweder ein Startpunkt, von dem die Halbgerade ausgeht und eine Richtung oder zwei verschiedene Punkte auf der Geraden, da mit diesen die Richtung ebenfalls gegeben ist. Mit den beiden Punkten P1(x1, y1, z1) und P2(x2, y2, z2) ist die Formel für die Gerade:

$$\begin{aligned}x &= x_1 + (x_2 - x_1) * t = x_1 + i * t \\y &= y_1 + (y_2 - y_1) * t = y_1 + j * t \\z &= z_1 + (z_2 - z_1) * t = z_1 + k * t,\end{aligned}$$

wobei (i, j, k) den Richtungsvektor der Geraden darstellt. Diese Halbgerade wird nun daraufhin überprüft, ob sie eines oder mehrere Objekte des Raumes schneidet. Das einfachste Objekt ist eine begrenzte Ebene – ein sogenanntes Polygon. Hier muß zunächst die Ebenengleichung aufgestellt werden, dann mit der Halbgeraden geschnitten und schließlich überprüft werden, ob die Halbgerade die Ebene im begrenzten Polygon schneidet.

An dieser Stelle lernen Sie die verschiedenen Arten der Multiplikation von Vektoren kennen: Zunächst gibt es die skalare Multiplikation, wobei ein Parameter t mit einem Vektor I (x2, y2, z2) multipliziert wird:

$$\begin{aligned}x_1 &= t * x_2 \\R = t * I &\rightarrow y_1 = t * y_2 \\z_1 &= t * z_2\end{aligned}$$

Weiterhin existiert das Skalarprodukt, wobei zwei Vektoren, R (x1, y1, z1) und I (x2, y2, z2), auf folgende Weise miteinander multipliziert werden:

$$\begin{aligned}x_1 &= x_1 * x_2 \\R = R * I &\rightarrow y_1 = y_1 * y_2 \\z_1 &= z_1 * z_2\end{aligned}$$

Als letztes ist das Vektorprodukt, auch Kreuzprodukt genannt, zu erwähnen. Auch hier werden zwei Vektoren, R (x1, y1, z1) und I (x2, y2, z2), multipliziert. Als Ergebnis erhalten Sie wiederum einen Vektor, der senkrecht zu den beiden ursprünglichen steht und als Normale bezeichnet wird:

$$\begin{aligned}Vx &= y_1 * z_2 - z_1 * y_2 \\V &= R \times I \rightarrow Vy = -x_1 * z_2 + z_1 * x_2 \\Vz &= x_1 * y_2 - y_1 * x_2\end{aligned}$$

Um eine Ebenengleichung aufzustellen, benötigen Sie min-

destens drei verschiedene Punkte der Ebene: P1, P2 und P3. Zunächst stellen Sie zwei Richtungsvektoren R1, R2 auf:

$$\begin{aligned}R_1 &= P_3 - P_1 \\R_2 &= P_2 - P_1\end{aligned}$$

Von den beiden Vektoren R1 und R2 bilden Sie das Vektorprodukt und erhalten die Normale N(n1, n2, n3) der Ebene. Nachdem Sie die Normale normiert, das bedeutet, die Normale durch die Division ihres Betrages

$$[N] = \sqrt{n_1^2 + n_2^2 + n_3^2}$$

zum Einheitsvektor gemacht haben

$$N_0 = N / [N],$$

kommen Sie zur Ebenengleichung, die folgendes Aussehen hat:

$$E = n_1 * x + n_2 * y + n_3 * z - (n_1 * x_4 + n_2 * y_4 + n_3 * z_4),$$

wobei x4, y4, z4 die Koordinaten eines Ebenenpunktes sind. Für den Schnitt der Geraden mit der Ebene gilt folgende Gleichung:

$$t = (n_1 * x_1 + n_2 * y_1 + n_3 * z_1) / (n_1 * i + n_2 * j + n_3 * k)$$

Falls $n_1 * i + n_2 * j + n_3 * k$ gleich Null ist, liegen Ebene und Gerade parallel und schneiden sich nicht. Falls t größer als Null ist, setzen Sie den Wert einfach in die Geradengleichung ein und erhalten so den Schnittpunkt.

Um zu überprüfen, ob sich der Schnittpunkt im abgegrenzten Polygon befindet, addieren Sie die Winkel aller Vektoren, die vom Schnittpunkt zu den Polygonecken gegeben sind. Ist die Summe gleich 360 Grad, liegt der Schnittpunkt im Polygon. Als weiteres Objekt des dreidimensionalen Raumes existiert die Kugel, die durch ihren Mittelpunkt M(l, m, b) und ihren Radius r eindeutig festgelegt ist.

$$(x - l)^2 + (y - m)^2 + (z - n)^2 = r^2$$

Setzen Sie nun für x, y, z die Geradengleichungen ein und substituieren, so ergibt sich:

$$a * t^2 + b * t + c = 0$$

wobei

$$a = i^2 + j^2 + k^2$$

$$b = 2 * i * (x_1 - l) + 2 * j * (y_1 - m) + 2 * k * (z_1 - n)$$

$$c = l^2 + m^2 + n^2 - x_1^2 - y_1^2 - z_1^2 - 2 * (l * x_1 + m * y_1 + n * z_1) - r^2$$

Damit sind

$$t_1, t_2 = (-b \pm \sqrt{b^2 - 4 * a * c}) / (2 * a)$$

Falls die Gerade die Kugel nicht schneidet, ist die Determinante $b^2 - 4 * a * c$ kleiner als Null. Falls die Determinante gleich Null ist, berührt (tangiert) die Gerade die Kugel in einem Punkt. Bei Determinanten-Werten größer Null schneidet die Gerade die Kugel in zwei Punkten. Falls ein t-Wert kleiner als Null ist, wird die Kugel hinter dem Startpunkt geschnitten und ist somit uninteressant. Nur wenn ein t-Wert größer als Null ist, liegt dieser Schnittpunkt dem Startpunkt der Halbgeraden voraus (Bild 5). Um die vollständigen Koordinaten des nächsten Schnittpunktes zu be-

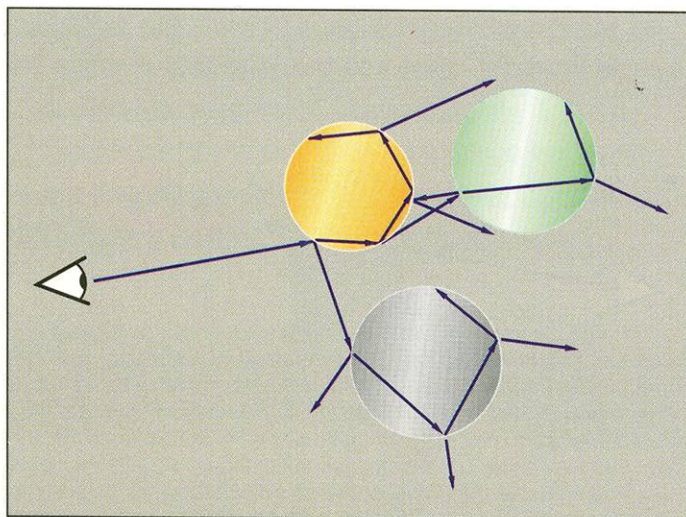


Bild 3. Ein Lichtstrahl teilt sich auf.

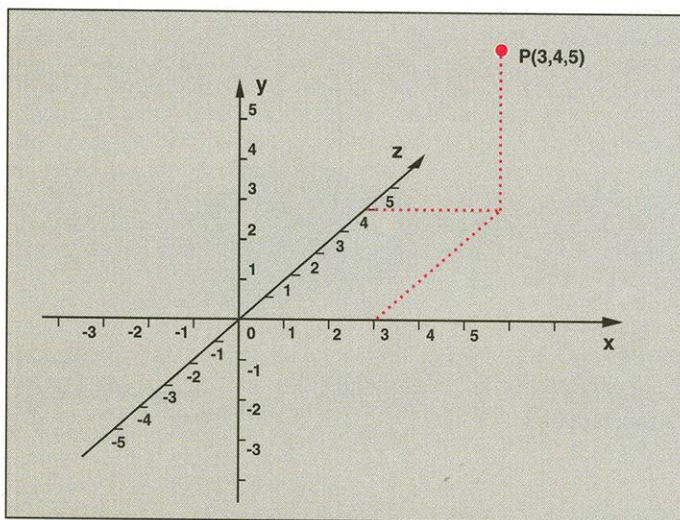


Bild 4. Ein Punkt im dreidimensionalen Raum

rechnen, nehmen Sie den kleinsten t-Wert, der größer ist als Null, und setzen ihn in die Geradengleichung ein.

Wenn Sie einen Lichtstrahl bis zu einem Objekt verfolgt und den Schnittpunkt errechnet haben, hängt es von der Beschaffenheit des Objektmaterials ab, ob Sie nun weitere Lichtstrahlen zu verfolgen haben. Dabei nehmen Sie als neuen Startpunkt für die Lichtstrahlen beziehungsweise Halbgeraden den Schnittpunkt des ursprünglichen Lichtstrahls mit dem Objekt.

Hat das Material spiegelnde Eigenschaften, so müssen Sie die Richtung des reflektierenden Lichtstrahls festlegen. Als Gesetz für Spiegel gilt hier, daß der Einfallswinkel gleich dem Ausfallswinkel ist. Wie Sie in Bild 6 sehen, ist I der Richtungsvektor des ursprünglichen Lichtstrahls, N ist die Normale (steht senkrecht) auf der Ebene und R ist die Richtung nach der Reflexion. Dabei gilt:

$$R = I + 2 * N * \cos(q)$$

wobei »cos(q)« ersetzt werden kann durch $(N \times I)$, falls N und I sogenannte Einheitsvektoren sind, was dann schließlich zu

$$R = I + 2 * N * (N \times I)$$

führt. Die Normale der Ebene gewinnen Sie, wie bereits weiter oben beschrieben. Die Normale der Kugel im Schnittpunkt ist eine Gerade durch deren Mittelpunkt und den Schnittpunkt. Wie diese Reflexionsrichtung programmtechnisch berechnet wird, sehen Sie im Unterprogramm »ref_vector()«.

Ist das Material lichtdurchlässig, so errechnen Sie die Richtung des transmittierenden Lichtstrahls, die von der sogenannten Velozität der beiden Medien oder Materialien (n_1 , n_2) abhängt (Bild 7). Die Velozität eines Mediums beeinflußt den Brechungswinkel eines Lichtstrahls. Je nachdem

von welchem Medium der Lichtstrahl in ein anderes kommt, wird er mit einem anderen Winkel gebrochen.

Dabei gelten für unseren Fall vereinfacht folgende Gesetze:

$$n = n_2/n_1$$

$$T = 1/n * I - (\sqrt{1 - 1/(n * n) * (1 - \cos(a) * \cos(a))}) - 1/n * \cos(a) * N$$

Dabei ist T die Richtung des transmittierenden Lichtstrahls, dessen programmtechnische Berechnung Sie in »ref_vector()« finden.

Wo viel Licht ist, ist auch viel Schatten

Da bei völliger Dunkelheit nichts außer schwarzen Objekten auf schwarzem Hintergrund zu berechnen ist, existieren in unserem mathematischen Raum natürlich auch Lichtquellen, von denen die eigentlichen Lichtstrahlen, die berechnet werden, ausgehen. Wenn diese Lichtquellen nun ein Objekt beleuchten, so sollte dieses Objekt auch einen Schatten werfen. Die Natürlichkeit der Darstellung erhöht sich deutlich, da die Räumlichkeit durch den Schatten verstärkt wird. Ob eine Kugel auf einer Ebene liegt oder über ihr schwebt, können Sie einzig und allein am Schatten auf der Ebene erkennen.

Nachdem also ein Lichtstrahl bis zu einem Objekt verfolgt wurde, ist festzustellen, von welchen Lichtquellen dieses wie stark beleuchtet wird. Bei diesem Berechnungsvorgang gehen die Distanz der Lichtquelle zum Objekt, ihre Stärke und ihre Farbe in die Helligkeit des Punktes ein. Um nun Schatten zu berechnen, wird wieder eine Halbgerade vom Schnittpunkt zu jeder Lichtquelle in den Raum geschickt. Trifft sie auf dem Weg zu einer Lichtquelle auf ein anderes Objekt, so kann diese Lichtquelle den Punkt nicht mehr be-

Computerunterstütztes Konstruieren und Zeichnen

DOS-CAD

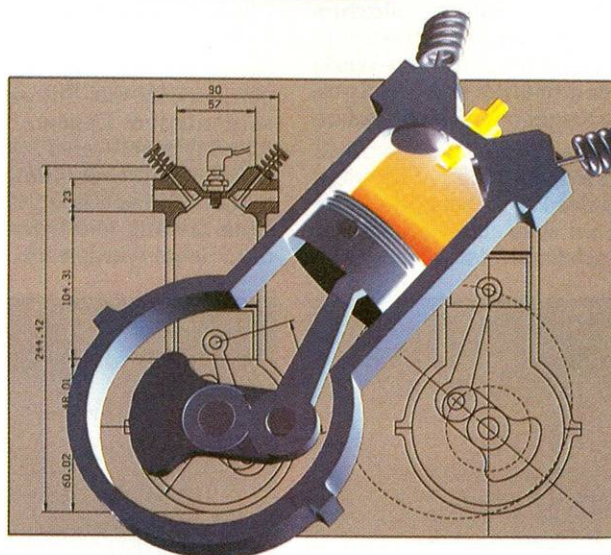
Ob Sie nun einen Schaltplan, einen Grundriß für eine Wohnungseinrichtung oder ein Strukturdiagramm entwerfen wollen, mit **DOS-CAD** nutzen Sie moderne CAD-Arbeitsmethoden und -techniken.

Die vielfältigen Funktionen umfassen das Zeichnen und Konstruieren von Polygonen, Rechtecken, Kreisen, Kreisbögen, Ellipsen, Doppellinien und vielem mehr. Komfortable Beschriftungsvarianten mit verschiedenen Zeichensätzen stehen Ihnen ebenso zur Verfügung wie alle wichtigen Funktionen zum Bemaßen der Zeichnung.

DOS-CAD arbeitet mit einer virtuellen Speicherverwaltung. Deshalb besteht keine Gefahr, daß Sie Ihre Zeichnung wegen zu geringer Speicherkapazität nicht vollenden können.

DOS-CAD ist ein leistungsfähiges und dennoch leicht zu erlernendes Programm: Die vielfältigen konstruktiven Möglichkeiten werden durch eine anwenderfreundliche und klar strukturierte Benutzeroberfläche überzeugend unterstützt.

Neben den Online-Hilftexten bietet Ihnen das umfangreiche und leicht verständliche Handbuch Auskunft bei Anwendungsfragen wie zum Beispiel beim Fan-



gen von Tangential- und Brennpunkten von Ellipsen.

Auch die Datenübergabe an andere Programme ist kein Problem.

DOS-CAD gibt Konstruktionen im HPGL-, Postscript- oder DXF-Format aus.

DOS-CAD unterstützt verschiedenste Peripheriegeräte: Alle üblichen Grafikkarten, Mäuse und Grafiktablets sowie Plotter, Matrix- oder Laserdrucker für die Zeichenausgabe.

Systemvoraussetzung:

PC oder Kompatibler mit mindestens 640 KByte RAM, MS-DOS ab Version 3.0, zwei Diskettenlaufwerke, Hercules-, CGA-, EGA- oder VGA-Grafikkarte, Maus

DOS-CAD

Das leistungsfähige Konstruktionsprogramm

für faszinierende Raytracing-Grafik:

Lightning 3D, Version 1.5

Die faszinierende Welt der Grafik

DM 199,—*

DM 199,—*

* Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland 4,— DM bzw. für das Ausland 6,— DM Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.

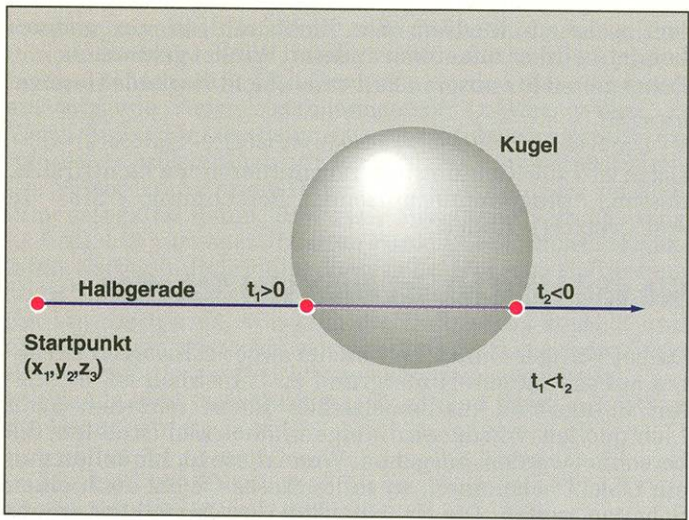


Bild 5. Eine Halbgerade schneidet eine Kugel.

leuchten, da ein zweites Objekt einen Schatten wirft (Bild 8).

Es existieren eine ganze Reihe verschiedener Algorithmen zur Berechnung eines Schattens, auf die hier aber nicht eingegangen werden soll. In dem hier vorgestellten Raytracer wird mit Hilfe der schon vorhandenen Funktion »hitobject()« der Weg zur Lichtquelle auf Objekte überprüft, die im Weg liegen, und gegebenenfalls ein Schatten erzeugt.

Eine andere, sehr wichtige Eigenschaft von Objekten ist, daß sie meist nicht einfarbig sind, sondern auf ihrer Oberfläche je nach Material verschiedene Musterungen – sogenannte Texturen – aufweisen. Holz oder Stein haben beispielsweise eine jeweils einzigartige Maserung. Die Texturen, die meist in zweidimensionaler Form vorliegen, müssen nun auf die Oberflächen der Objekte aufgetragen werden. Da allerdings die Objekte in Form und Größe meist nicht der Textur entsprechen, ist das Muster sozusagen mathematisch um das jeweilige Objekt zu wickeln, beispielsweise um eine Kugel. Dieses Verfahren, »texture mapping« genannt, ist von Objektform zu Objektform unterschiedlich. Mit dem hier vorgestellten Raytracing-Werkzeug sind Sie in der Lage, Texturen mit Hilfe eines Malprogramms anzufertigen und auf eine Kugel oder Fläche aufzutragen.

Der Algorithmus, der eine Textur auf eine Fläche aufträgt, sieht wie folgt aus: Zunächst wird der Winkel ermittelt, der durch den Vektor vom Schnittpunkt zum ersten Punkt des Polygons sowie dessen erster Seite definiert ist. Dieser Win-

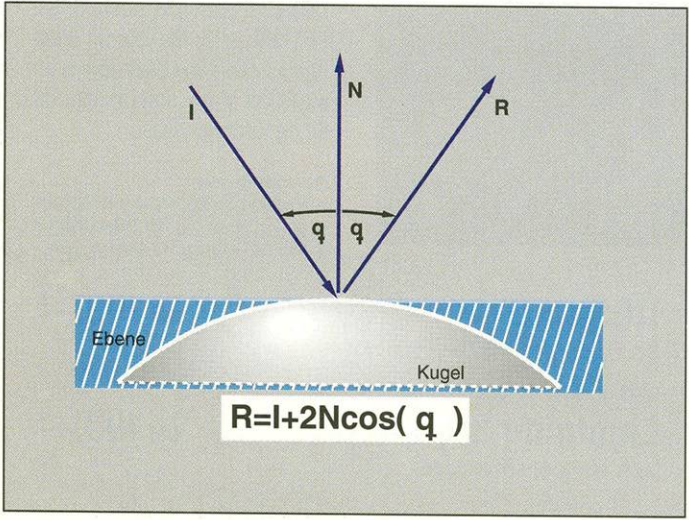


Bild 6. Spiegelung an einer Kugel oder Ebene

kel wiederum definiert einen x- und y-Wert, der auf die Textur relativ abgebildet wird und so die Farbe eines Oberflächenpunktes ergibt. Bei der Kugel findet ein ähnliches Verfahren Anwendung, bei dem der Winkel des Schnittpunktes und der z-Achse in x- und y-Richtung ausgemessen und dadurch ein bestimmter Punkt der Textur ausgewählt wird.

Jetzt kommt Farbe ins Spiel

Wichtig für eine realistische Darstellung von verschiedenen Objekten mit Mustern und Schattierungen ist die Palette der darstellbaren Farben. Eine Farbe in dem hier vorgestellten Raytracing-System setzt sich aus den drei Farbanteilen Rot, Grün und Blau zusammen, die jeweils Werte zwischen Null und Eins annehmen können. Diese Art der Farbangabe nennt man daher RGB-Modell. Es ist für Standardmonitore gedacht, deren Signale sich aus den Farben Rot, Grün und Blau zusammensetzen. Dabei ist (0,0,0) Schwarz und (1,1,1) Weiß. Dazwischen gibt es eine nahezu unbegrenzte Anzahl von Zwischenfarben, die jedoch nicht alle von einem Monitor dargestellt werden können.

Den RGB-Farbraum können Sie sich als Würfel vorstellen, bei dem eine Kante die Intensität der roten Farbe, eine andere die Intensität der grünen und eine dritte Kante die der blauen Farbe repräsentiert (Bild 9). Das hier verwendete RGB-Farbmodell schließt alle menschlich sichtbaren Farben ein. Daneben existieren einige andere bekannte Farbmodelle wie beispielsweise das CMY-Modell (Cyan Magenta Yellow) oder das HSV-Modell, auf die hier nicht näher eingegangen werden soll. Daß für eine realistische Darstellung wirklich alle sichtbaren Farben benötigt werden, sehen Sie an folgendem Beispiel:

Eine einfarbig rote Fläche (1.0,0.0,0.0), die von einer mittelstarken Lichtquelle beleuchtet wird, hat an der Stelle, an der die Lichtquelle der Fläche am nächsten ist, ein sehr helles Rot, fast schon Weiß. Je weiter ein Punkt dieser Fläche aber entfernt ist, desto dunkler wird das Rot, bis schließlich in weiter Entfernung fast kein Licht mehr ankommt und die Fläche nur noch schwarz gesehen wird, obwohl Sie eigentlich rot ist.

Sie sehen also, daß von jeder beliebigen Farbe alle Nuancen, sowohl ins Schwarze als auch ins Weiße, vorhanden sein müssen. Da aber die gebräuchlichen VGA-Karten nur 256 Register – also 256 gleichzeitig darstellbare Farben – zur Verfügung stellen, ergibt sich das Problem, alle Farben des RGB-Modells in diesen 256 Registern unterzubringen. Eine Lösung stellt die sogenannte »look-up table« (LUT) dar. Dabei werden die Werte der Farbanteile R,B und G in

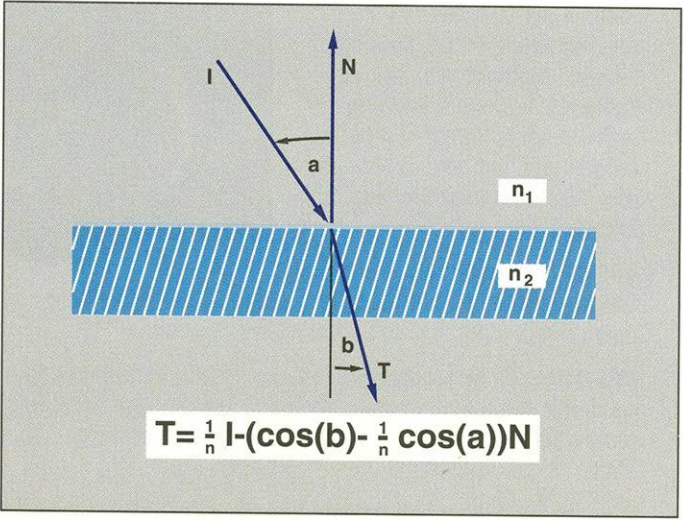


Bild 7. Durch eine Objektwand gehender Lichtstrahl

Material	Farbe (RGB-Anteil)	Undurchsichtigkeit	Spiegelung	Transparenz	Matte Reflexion	Spiegelnde Reflexion	Konzentration der spiegelnden Reflexion	Veloazität
Kunststoff	beliebig	1.0	0.0	0.0	0.7	0.7	40	1.0
Metall	0.8/0.8/1.0	0.7	0.3	0.0	0.5	0.5	100	1.0
Glas	1.0/1.0/1.0	0.2	0.0	0.8	0.5	0.7	60	1.5

Tabelle 1. Einige Materialien und ihre Definitionen

einen Zeiger umgerechnet, der auf einen Punkt im RGB-Würfel zeigt, der wiederum nur in den verfügbaren Registern existiert (Bild 10).

Mit Hilfe der Register muß ein Farbwürfel möglichst großer Kantenlänge dargestellt werden. Das bedeutet, daß bei einer maximalen Kantenlänge von sechs $6 \times 6 \times 6 = 216$ Register gebraucht werden. In diese 216 Register werden bei der Initialisierung des Raytracers durch Aufruf von »init_pal()« die dem RGB-Würfel entsprechenden Werte geladen. Der Raytracer berechnet zunächst den eigentlichen Farbwert eines Punktes im RGB-Modell und paßt ihn dann mit Hilfe eines Katalysators dem 216-elementigen Farbwürfel einer VGA-Karte an. Um den Raytracer auf anderen Rechnern mit besseren Grafikfähigkeiten einzusetzen, müssen Sie lediglich die Farbübersetzungsfunktion »to_LUT()« durch eine neue, den Grafikfähigkeiten entsprechende ersetzen.

Die Farbe eines Punktes auf der matten Oberfläche eines Objekts hängt, einfach gesagt, von vier wesentlichen Faktoren ab:

- Der erste Faktor ist das Licht, das von der Fläche reflektiert beziehungsweise absorbiert wird. Durch teilweise Absorption von Licht entsteht die Farbe eines Objekts. Welche Farben des Lichts absorbiert werden, hängt vom atomaren Aufbau des Materials ab. Das restliche, nicht absorbierte Licht wird diffus reflektiert, das heißt, es kommt nicht zum Betrachter zurück, wie wenn er in einen Spiegel schauen würde, sondern wird zerstreut. Allerdings – wie stark das Licht dieser sogenannten diffusen Reflexion ist, hängt davon ab, in welchem Winkel der Lichtstrahl auf die Fläche auftrifft (Bild 11). Dabei gilt die grundlegende Formel:

$$I_d = I \cdot k \cdot \cos(a) = I \cdot k \cdot (L \cdot N)$$

Hierbei ist I die Intensität der Lichtquelle, k eine materialabhängige Konstante und a der Winkel zwischen der Normalen auf die Oberfläche und dem auftreffenden Lichtstrahl.

- Der zweite Faktor für die Farbe eines Punktes ist ein immer vorhandenes Umgebungslicht, das aufgrund der Reflexion anderer Flächen entsteht. Dieses Umgebungslicht I_u geht nur mit einer Konstante k in die Rechnung ein:

$$I_u = I \cdot k$$

- Drittens existiert die sogenannte spiegelnde Reflexion. Auf einer roten Billardkugel entsteht beispielsweise dort ein kleiner, runder, heller Fleck, wo sie am stärksten beleuchtet wird. Diese spiegelnde Reflexion tritt nur innerhalb eines ganz bestimmten Winkels zwischen Oberfläche und Lichtstrahl auf (Bild 12). Sie wird folgendermaßen berechnet:

$$I_s = I \cdot k \cdot \cos(b) + n = I \cdot k \cdot (R_x V) + n$$

n beeinflusst dabei die Größe des Winkels und die Intensität des Lichts. Je größer n , desto kleiner und heller wird das Highlight auf der Kugel im obigen Beispiel.

- Der vierte Faktor ist die Distanz. Je weiter eine Lichtquelle von einem Objekt entfernt ist, desto weniger stark wird es beleuchtet.

Die lokale Farbe und Helligkeit FH eines Oberflächenpunktes berechnet sich aus:

$$FH = I_u + (I_d + I_s) / \text{distance}$$

Dabei sind Helligkeit und Farbe, die durch Transparenz oder Spiegeleigenschaften des Materials hinzukommen, noch nicht berücksichtigt. Diese müssen zur lokalen Farbe addiert werden und dann immer noch innerhalb des RGB-Würfels liegen. Andernfalls wurden unmögliche Materialien verwendet. Denn eine Kugel kann nicht gleichzeitig intensiv rot und völlig transparent sein.

Wir bauen uns eine Welt zusammen

Das hier vorgestellte Raytracing-Werkzeug besteht aus drei wesentlichen Komponenten. Zwei davon sind Hilfsmittel, um die Schöpfung neuer Welten zu vereinfachen. Die dritte Komponente ist die eigentliche Lichtstrahlberechnung. Falls Sie Turbo C 2.0 besitzen, übersetzen Sie das Hauptprogramm »ray.c« (Listing 8) mit Hilfe des Kommandozeilencompilers. Tippen Sie also einfach:

```
tcc ray.c
```

Besitzer von Turbo C++ sind in der Lage, alles im normalen Turbo Editor zu compilieren. »ray.c« stellt Ihnen die Werkzeuge der einzelnen Unterprogramme (Listing 1 bis 7 und 9 bis 12) zur Verfügung. Nach Aufruf von »ray.exe« landen Sie zunächst im mausgesteuerten Hauptmenü (Bild 13),

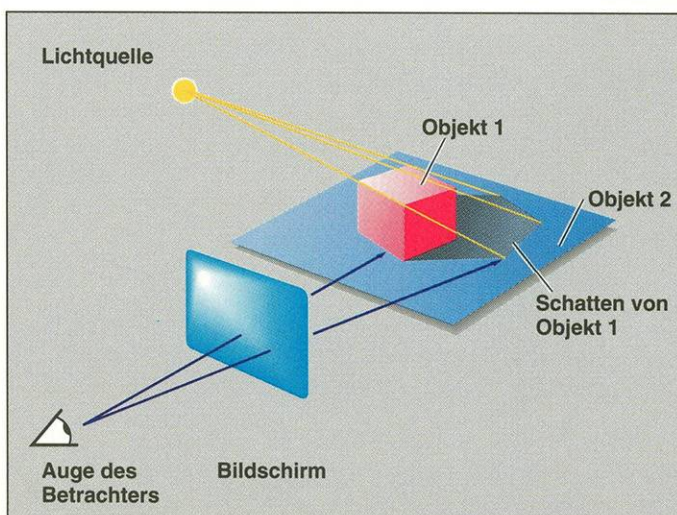


Bild 8. Schatten eines Objekts, bei einer Lichtquelle

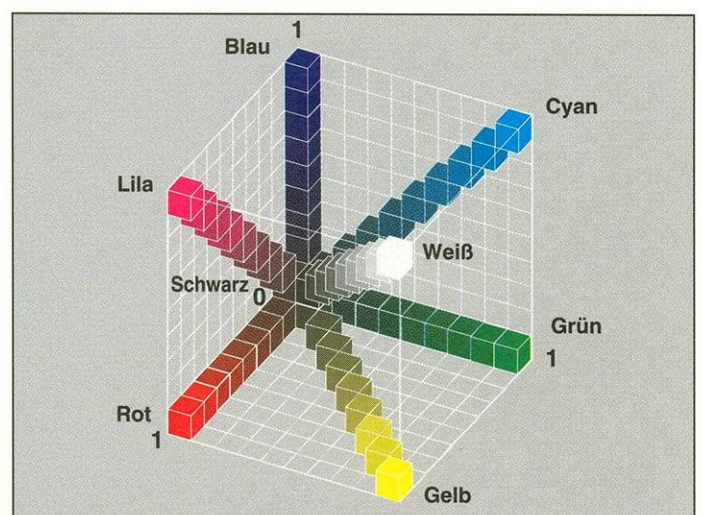


Bild 9. Das RGB-Farbmodell

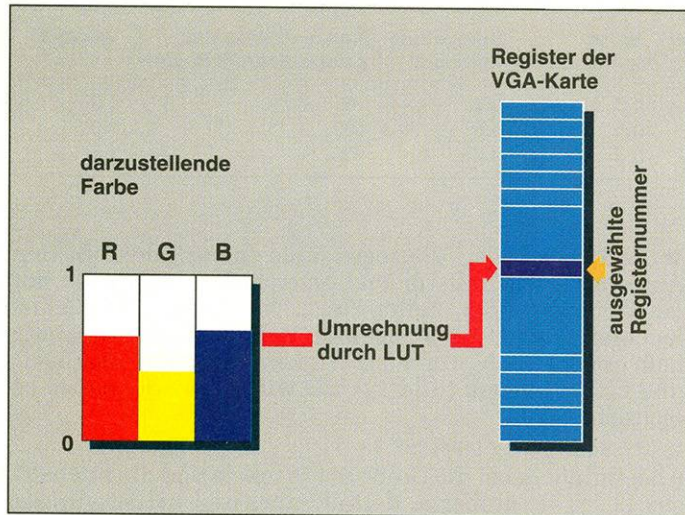


Bild 10. Eine Look-up-Tabelle zur Farbdarstellung

von wo aus alle Komponenten des Raytracers zugänglich sind. Um einen Menüpunkt anzuwählen, bewegen Sie den Mauszeiger auf diesen Punkt und drücken einmal die linke Maustaste. Im folgenden werden die Funktionen und Verwendungsmöglichkeiten der Menüpunkte kurz erläutert:

- »MAKE 3DATA« (make 3d data): ruft den CAD-ähnlichen Editor für dreidimensionale Objekte auf. Nähere Informationen finden Sie bei der Erläuterung dieses Editors.

- »LOAD PIC« (load picture): lädt ein schon berechnetes Raytracing-Bild auf den Bildschirm. Dabei haben Sie die Wahl zwischen den Auflösungen 320×200 , 640×480 , 800×600 und 1024×768 Pixel. Die entsprechenden Modi für Ihre VGA-Karte entnehmen Sie Tabelle 2. Tragen Sie diese in »ray.c« an den beiden markierten Stellen ein. Laden Sie aber keine höher aufgelösten Bilder mit niedrigerer Auflösung ein.

- »MAKE TEXT« (make texture): ruft ein einfaches Malprogramm auf, das Ihnen erlaubt, Texturen (Muster) von Objekten zu schaffen.

- »ED OBJ« (edit object): Nach Aufruf dieses Menüpunktes wählen Sie ein mit »MAKE 3DATA« erstelltes Objekt aus und definieren sein Aussehen und seine Materialeigenschaften. Dabei sind folgende Werte einzugeben:

- Nummer des Objekts: Durch Angabe der Nummer wählen Sie ein bestimmtes Objekt aus. Die Nummern werden während der Erzeugung der Objekte fortlaufend vergeben. Das zuerst generierte Objekt hat die Nummer eins, dann zwei und so weiter.

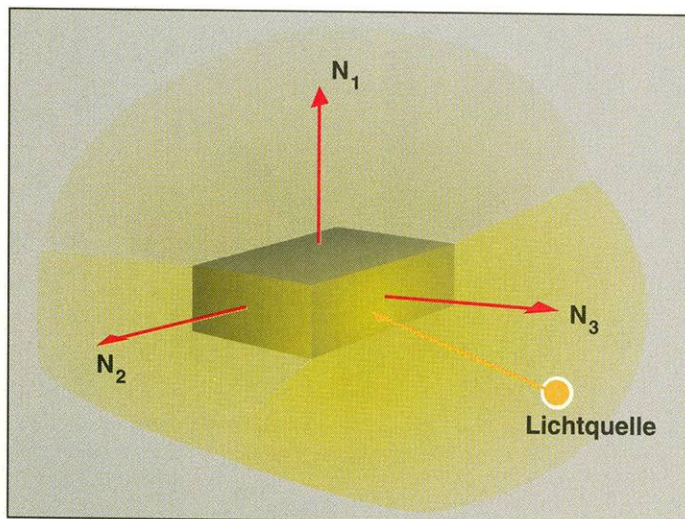


Bild 11. Diffus reflektierende Oberflächen

- Textur oder einfarbig: Mit dieser Angabe legen Sie fest, ob das Objekt mit einem von Ihnen gemalten Muster überzogen oder nur einfarbig dargestellt wird. Bei der Wahl von »Textur« geben Sie den Dateinamen der Textur und die Streckung an. Die Streckung besagt, um wieviel Mal die Textur vergrößert auf das Objekt abgebildet werden soll. Bei »einfarbig« geben Sie die Farbe an, indem Sie die Anteile Rot, Grün und Blau jeweils zwischen Null und Eins festlegen (Schwarz (0,0,0); Weiß (1,1,1)).
- Undurchsichtigkeit: Wie stark reflektiert Ihr Material einen ankommenden Lichtstrahl?
- Spiegelung: Wie stark spiegelt sich im Material die Umgebung (1.0 perfekter Spiegel)?
- Transparenz: Wie stark ist das Material transparent (1.0 perfektes Glas)?
- Matte Reflexion: Wieviel Licht wird diffus reflektiert (siehe oben »diffuse Reflexion«)?
- Spiegelnde Reflexion: Wieviel Licht wird spiegelnd reflektiert (siehe oben »spiegelnde Reflexion«)?
- Konzentration der spiegelnden Reflexion: Wie klein ist der Winkel der spiegelnden Reflexion (siehe oben »spiegelnde Reflexion«)?
- Velozität: Konstante, die die Winkelbrechung bei transparenten Materialien definiert (siehe oben »Transparenz«).

In Tabelle 1 sehen Sie die Materialien Kunststoff, Metall und Glas als Beispiele vordefiniert.

- »ED LAMP« (edit lamp): Hier wird zunächst die Strahlungsstärke einer Lichtquelle festgelegt, indem die Halbwertszeit eingegeben wird. Halbwertszeit bedeutet, daß das Licht bei der eingegebenen Entfernung nur noch halb so stark ist wie direkt bei der Lichtquelle. Anschließend wird die Farbe des Lichts durch Eingabe der Farbanteile festgelegt.

- »ED OBS« (edit observer): dient zur Definition einiger globaler Variablen, und zwar für Sie selbst als Betrachter und für die darzustellende Welt.

- Auflösung: In welcher Auflösung soll das Bild berechnet werden?
- Bildbreite: Wieviele Pixel breit soll das Bild sein?
- Bildhöhe: Wieviele Pixel hoch soll das Bild sein?
- Hintergrundfarbe: Welche Hintergrundfarbe soll das Szenario haben? Die gewählte Hintergrundfarbe wird automatisch in ihrer jeweiligen Helligkeit als Kugel über das Szenario gezogen.
- Farbe der Grundhelligkeit: Welche Farbe ist aufgrund unspezifizierter Reflexionen sowieso vorhanden? Für realistische Darstellungen ist es immer von Vorteil, ein wenig Grundhelligkeit zu haben (0.1,0.1,0.1).
- Stärke der Grundhelligkeit: Welche Helligkeit hat die Grundhelligkeit (0.1)?

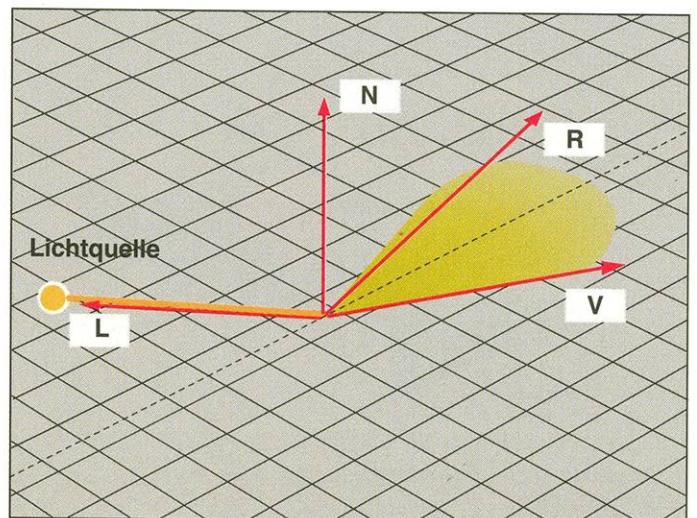


Bild 12. Die spiegelnde Reflexion innerhalb eines Winkels

Grafikkarte	640 × 480	800 × 600	1024 × 768
4Max/1MByte	2F	30	38
Ahead	61	62	—
ATI Prisma Elite	5D	5E	—
ATI VGA Wonder	62	63	—
Cardinal	79	7B	—
Genoa 6400	2E	30	—
Hi Res 512	5D	5E	—
Maxxon	5D	5E	—
OAK	53	54	—
Orchid Designer	2E	30	—
Paradise VGA Pro	5F	—	—
STB EM 16	2E	30	38
Trident 8900	5D	5E	62
Video 7 VRAM	67	—	—

Tabelle 2 zeigt die Videomodi (hex) üblicher Super-VGA-Karten (256 Farben).

- Berechnungstiefe: Zwei gegenüberliegende perfekte Spiegel spiegeln im Prinzip unendlich tief. Wie weit sie tatsächlich berechnet werden, geben Sie an dieser Stelle ein.
- »SAVE WORLD«: speichert eine generierte Welt, das heißt 3D-Daten, Materialdefinitionen und Muster. Damit sind Sie in der Lage, eine Welt zu einem späteren Zeitpunkt zu erweitern und auszubauen.
- »LOAD WORLD«: lädt eine früher erstellte Welt zur weiteren Bearbeitung wieder ein.
- »CALC WORLD«: Nach Eingabe der Auflösung und des Dateinamens für ein zu speicherndes Bild wird die Welt durch Raytracing berechnet. Ist die Berechnung beendet, taucht der Mauszeiger wieder auf. Wollen Sie nun das Bild unter dem vorher eingegebenen Namen speichern, drücken Sie die rechte Maustaste, andernfalls die linke.
- »EXIT«: beendet das Raytracing-Programm.

Mit CAD in eine neue Welt

Zur Konstruktion von Körpern wird in der Praxis ein Software-Werkzeug eingesetzt, dessen Typus auch für die Schöpfung neuer Raytracing-Welten geeignet ist. Wenn Sie also im Hauptmenü »MAKE 3DATA« aufrufen, starten Sie ein CAD-ähnliches Entwicklungswerkzeug für dreidimensionale Objekte. Der gesamte Bildschirm ist dabei in vier Fenster aufgeteilt, die folgende Funktionen erfüllen: Die oberen drei Fenster zeigen den mathematischen Raum aus drei verschiedenen Blickrichtungen. Im linken Fenster betrachten Sie die Objekte von oben. Das mittlere Fenster

zeigt die Ansicht von vorne, und das rechte Fenster stellt das gesamte Szenarium von der Seite dar. Das untere Fenster dient zu Eingaben und enthält das mausgesteuerte Menü des Programms (Bild 14).

Die drei verschiedenen Blickrichtungen auf das Szenarium versetzen Sie in die Lage, über die drei Fenster einen dreidimensionalen Punkt mit der Maus anzusteuern und die Eckpunkte von Körpern zu definieren. Bei allen Eingaben über die Ansichtsfenster ist die Maus das einzige Werkzeug, das Sie bedienen. Durch Drücken der rechten Maustaste gelangen Sie von einem Ansichtsfenster zum anderen. Der Mauszeiger springt jeweils in das weiter rechts gelegene Fenster. Sind Sie rechts außen angekommen, wird ins linke Fenster zurückverzweigt.

Um die Ansichtsfenster zu verlassen und ins Menüfenster zurückzukehren, drücken Sie zunächst die rechte Maustaste, halten sie gedrückt und drücken dann zusätzlich die linke Maustaste. Dann lassen Sie beide Tasten auf einmal los, und der Mauszeiger springt wieder ins Menüfenster. Die linke Maustaste, allein gedrückt, dient zur Festlegung einer Position im dreidimensionalen Raum, worauf später, bei der Besprechung der Menüpunkte, noch näher eingegangen wird, da die Funktion vom gewählten Menüpunkt abhängig ist. Bei allen Bewegungen des Mauszeigers in einem der drei Ansichtsfenster wird die dreidimensionale Position des Zeigers verändert, und damit auch die Position des Zeigers in den anderen Fenstern, in denen er gerade nicht aktiv ist. Ein Verändern des y-Wertes im mittleren Fenster gilt natürlich auch für die Seitenansicht im rechten Fenster. Im unteren Fenster wird links oben immer der aktuelle x-, y- und z-Wert der dreidimensionalen Position der Maus angezeigt. Im folgenden lernen Sie nun die Menüpunkte und Funktionen dieses Fensters kennen:

- »OBJ BEG« (object begin): Ein Objekt kann während der Entwicklung aus mehreren Flächen oder Kugeln bestehen. Um ein neues Objekt zu eröffnen, muß zuerst dieser Menüpunkt gewählt werden. Alle Bewegungs- und Drehaktionen beziehen sich dann auf dieses aktuelle Objekt. Dies gilt so lange, bis Sie durch erneute Wahl von »OBJ BEG« ein neues Objekt eröffnen. Zu diesem Zeitpunkt sollte Ihr altes Objekt abgeschlossen und richtig positioniert sein.

- »ADD POL« (add polygon): addiert eine Fläche zum aktuellen Objekt. Der Mauszeiger landet in den Ansichtsfenstern, und mit jedem Druck der linken Maustaste legen Sie einen Punkt der nicht obligatorisch viereckigen Fläche fest. Hat eine Fläche mehr als drei Eckpunkte, so definieren Sie diese am besten in einem einzigen Ansichtsfenster, also ohne das Fenster zu wechseln, da es schwer ist, mehrere Punkte von Hand exakt in einer Ebene zu setzen. Sie können die erzeugte Fläche dann nachträglich drehen und verschieben.

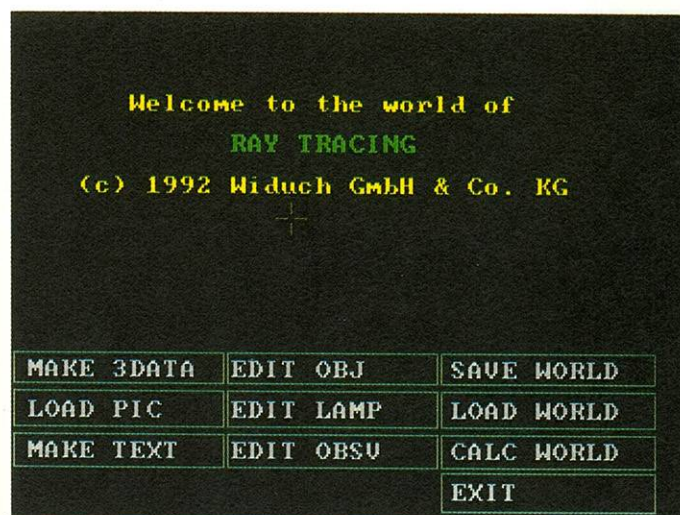


Bild 13. Das Hauptmenü des Raytracers

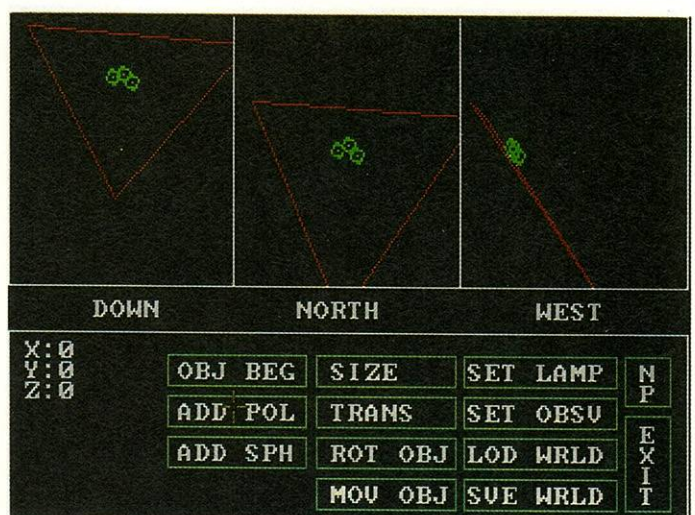


Bild 14. Das CAD-Werkzeug zur Generierung neuer Welten

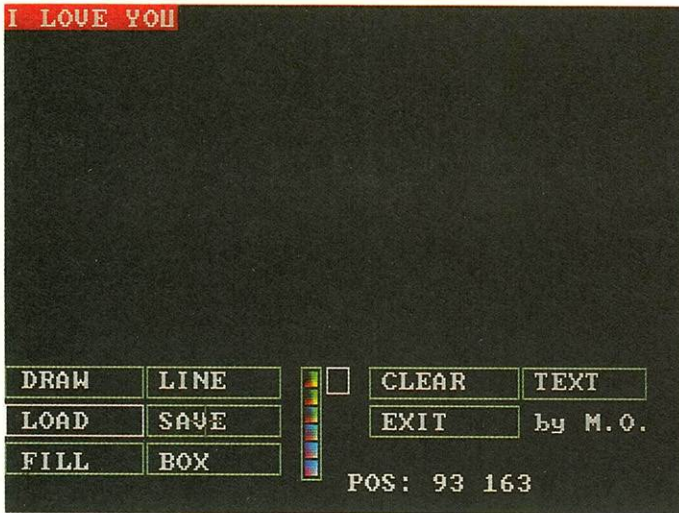


Bild 15. Das Malprogramm für Texturen

- »ADD SPH« (add sphere): addiert eine Kugel zum aktuellen Objekt. Der Mauszeiger landet dabei ebenfalls in den Ansichtsfenstern. Mit dem ersten Druck der linken Maustaste legen Sie den Mittelpunkt der Kugel fest, mit dem zweiten ihren Radius.
- »SIZE« (size): verändert die Darstellungsgröße der Ansichtsfenster, zu Beginn beträgt dieser Wert zehn. Das bedeutet, wenn Sie den Mauszeiger um ein Pixel bewegen, so entspricht das zehn Einheiten im mathematischen Raum. Durch Eingabe des neuen Maßes M legen Sie fest, wieviele Einheiten einem Pixel entsprechen. Diese Option läßt Sie das Szenarium beliebig verkleinern und vergrößern.
- »TRANS« (transformation): Um kleine Details in Positionen weit entfernt vom Ursprung (0,0,0) – der sich zu Beginn in der Mitte jedes Fensters befindet – zu editieren, setzen Sie die Transformation ein. Durch die Eingabe von x-, y- und z-Wert verändern Sie die Lage des Aussichtspunktes relativ um die eingegebene Position.
- »ROT OBJ« (rotate object): dreht das Objekt um einen Winkel W und um eine bestimmte Achse (x,y oder z). Dabei dreht sich das Objekt in sich selbst und nicht um die Raumachsen.
- »MOV OBJ« (move object): Ein Objekt wird gänzlich um einen Vektor im Raum bewegt.
- »SET LAMP« (set lamp): Mit jedem Aufruf dieses Menüpunktes addieren Sie eine neue Lichtquelle zu Ihrem Szenarium, die durch einen blauen Punkt dargestellt wird.
- »SET OBSV« (set observer): legt mit dem ersten Druck der linken Maustaste die Position des späteren Betrachters

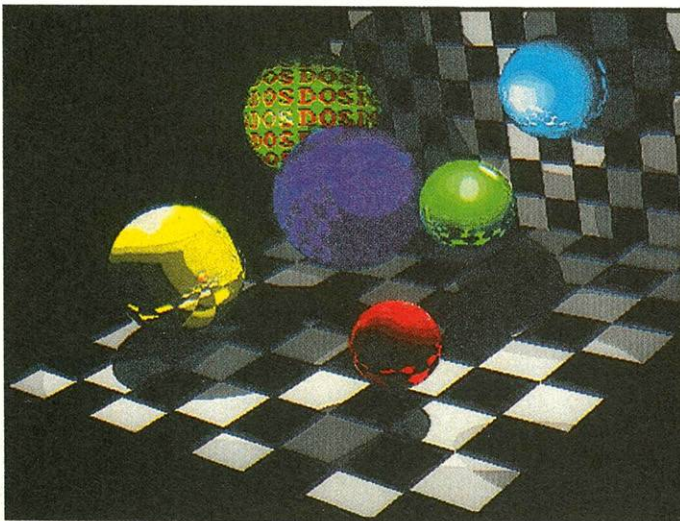


Bild 16. Ein mit »CALC WORLD« berechnetes 3D-Bild

fest und mit dem zweiten Druck die Richtung, in die er schaut. Diese Informationen werden durch den gelben Vektor veranschaulicht.

- »LOD WRLD« (load world): lädt eine früher erzeugte Welt wieder in den Editor.
- »SVE WRLD« (save world): speichert die aktuelle Welt des Editors.
- »NP« (new position): Die durch den Menüpunkt »SET OBSV« gesetzte Blickrichtung des Betrachters wird als neue Blickrichtung auf die drei Ansichtsfenster übernommen.
- »EXIT«: verläßt den 3D-Editor und kehrt zurück ins Hauptmenü, wo Sie die Daten weiterverarbeiten können.

Mit dem Malprogramm erzeugen Sie die Texturen

Das einfache Malprogramm dient zur Erzeugung von Mustern und Oberflächenstrukturen der Objekte. Der Bildschirm ist wiederum zweigeteilt. Der obere, anfänglich schwarze Bereich ist die Zeichenfläche. Im unteren Teil befindet sich das Menü samt weiteren Funktionen (Bild 15). Die zwei für das Malen wichtigen Angaben im unteren Bereich sind einerseits die Positionsdaten des Mauszeigers und andererseits die aktuelle Zeichenfarbe. Die Menüpunkte führen im einzelnen folgende Funktionen aus:

- »DRAW«: erlaubt das Freihandzeichnen mit der aktuellen Zeichenfarbe.
- »LINE«: zieht Linien von dem Punkt, an dem die linke Maustaste gedrückt wurde, bis zu dem Punkt, wo sie losgelassen wurde.
- »LOAD«: lädt eine früher erzeugte Textur oder ein Bild.
- »SAVE«: speichert einen Ausschnitt des Bildschirms unter dem eingegebenen Namen ab. Den Bildschirmausschnitt definieren Sie durch Drücken und Loslassen der linken Maustaste.
- »FILL«: füllt eine rechteckige Fläche mit der aktuellen Zeichenfarbe auf.
- »BOX«: zeichnet ein Rechteck auf den Bildschirm. Der erste Druck auf die linke Maustaste definiert die linke obere Ecke, das Loslassen die rechte untere Ecke des Rechtecks.
- Der nächste Menüpunkt bietet Zugriff auf die Farbtabelle aller verwendeten Farben. Hier wählen Sie die aktuelle Zeichenfarbe durch Druck auf die linke Maustaste.
- »CLEAR«: löscht die gesamte Zeichenfläche.
- »TEXT«: gestattet die Eingabe von Texten.
- »EXIT«: kehrt ins Hauptmenü des Raytracers zurück.



Bild 17. Hier sehen Sie Bild 16 aus anderer Perspektive.

Auf zu neuen Ufern

An dieser Stelle ist noch erwähnenswert, daß sowohl Texturen als auch berechnete Raytracing-Bilder im selben Format gespeichert werden, weshalb Sie berechnete Bilder wiederum als Texturen benutzen können. Das Speicherformat besteht aus einem Wort für die Bildschirmbreite, einem Wort für die Bildschirmhöhe und schließlich, der Reihe nach, allen Pixelwerten. Die Erzeugung eines Raytracing-Bildes verläuft in vier Schritten: Zunächst erzeugen Sie mit dem 3D-Editor eine dreidimensionale Welt. Dann malen Sie – falls erwünscht – mit dem Malprogramm die dazugehörigen Muster. Schließlich definieren Sie im Hauptmenü Objekte, Lichtquellen und Betrachter und berechnen das Bild zum Schluß mit »CALC WORLD«. Auf diese Weise erzeugen Sie sehr schnell und einfach die faszinierendsten Bilder (Bilder 16 und 17).

Das Entwicklungswerkzeug haben Sie jetzt und können es mit weiteren Fähigkeiten und Komponenten ausbauen und ergänzen. Als Anregung für Entwickler hier einige Ideen: Eine zusätzliche, interessante Erweiterung wäre ein Animationsprogramm, das es erlaubt, Kamerafahrten durch die neuen Welten zu inszenieren und mehrere berechnete Bilder hintereinander als Film abzuspielen oder die neuen Bilder auf Video zu übertragen und mit realen Bildern zu mischen ...

(Matthias Oberdorfer/bm)

Allan Watt, Fundamentals of Three-dimensional Computergraphics, 430 Seiten, Addison-Wesley, ISBN 020115442 0

```
[W60] 1: /*
[OK2] 2: Programm: cad.c
[AK4] 3: Funktion: CAD-Programm für
[6Y2] 4: Ray Tracer Welten
[9C5] 5: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 6: Autor: Matthias Oberdorfer
[V04] 7: (c) 1992 Widuch GmbH & Co. KG
[170] 8: */
[000] 9:
[FV1] 10: #ifndef CAD
[NV1] 11: #define CAD
[000] 12:
[3F2] 13: #include"vid.c"
[AL2] 14: #include"maus.c"
[1R2] 15: #include"graph.c"
[GB3] 16: #include"schalter.c"
[3H2] 17: #include"calc.c"
[DL2] 18: #include"load.c"
[000] 19:
[000] 20:
[043] 21: #include<stdlib.h>
[8W2] 22: #include<stdio.h>
[RN2] 23: #include<math.h>
[2T2] 24: #include<alloc.h>
[ML2] 25: #include<mem.h>
[000] 26:
[ZH2] 27: schalter np = {
[470] 28: 0,
[9Z1] 29: 289,133,310,152,
[B11] 30: 56,0,215,
[870] 31: 2,
[2F0] 32: "N",
[BD0] 33: };
[000] 34:
[KG2] 35: schalter sa = {
[2N0] 36: &np,
[YZ1] 37: 214,181,285,193,
[B11] 38: 56,0,215,
[870] 39: 2,
[HU1] 40: "SVE WRLD",
[BD0] 41: };
[000] 42:
[NH2] 43: schalter lo = {
[NL0] 44: &sa,
[K02] 45: 214,165,285,177,
[B11] 46: 56,0,215,
[870] 47: 2,
[ZS1] 48: "LOD WRLD",
[BD0] 49: };
[000] 50:
```

```
[AK2] 51: schalter so = {
[SM0] 52: &lo,
[102] 53: 214,149,285,161,
[B11] 54: 56,0,215,
[870] 55: 2,
[FR1] 56: "SET OBSV",
[BD0] 57: };
[000] 58:
[YH2] 59: schalter sl = {
[DN0] 60: &so,
[GZ1] 61: 214,133,285,145,
[B11] 62: 56,0,215,
[870] 63: 2,
[GP1] 64: "SET LAMP",
[BD0] 65: };
[000] 66:
[SH2] 67: schalter mo = {
[1N0] 68: &sl,
[0Z1] 69: 145,181,210,193,
[B11] 70: 56,0,215,
[870] 71: 2,
[4G1] 72: "MOV OBJ",
[BD0] 73: };
[000] 74:
[7K2] 75: schalter ro = {
[VM0] 76: &mo,
[MZ1] 77: 145,165,210,177,
[B11] 78: 56,0,215,
[870] 79: 2,
[HG1] 80: "ROT OBJ",
[BD0] 81: };
[000] 82:
[TK2] 83: schalter tr = {
[AN0] 84: &ro,
[3Z1] 85: 145,149,210,161,
[B11] 86: 56,0,215,
[870] 87: 2,
[K81] 88: "TRANS",
[BD0] 89: };
[000] 90:
[KH2] 91: schalter si = {
[WN0] 92: &tr,
[KY1] 93: 145,133,210,145,
[B11] 94: 56,0,215,
[870] 95: 2,
[V21] 96: "SIZE",
[BD0] 97: };
[000] 98:
[EK2] 99: schalter os = {
[NM0] 100: &si,
[YU1] 101: 75,133,140,145,
[B11] 102: 56,0,215,
[870] 103: 2,
[CD1] 104: "OBJ BEG",
[BD0] 105: };
[SG2] 106: schalter ap = {
[HN0] 107: &os,
[AV1] 108: 75,149,140,161,
[B11] 109: 56,0,215,
[870] 110: 2,
[VE1] 111: "ADD POL",
[BD0] 112: };
[4H2] 113: schalter as = {
[VL0] 114: &ap,
[PV1] 115: 75,165,140,177,
[B11] 116: 56,0,215,
[870] 117: 2,
[4F1] 118: "ADD SPH",
[BD0] 119: };
[000] 120:
[4K2] 121: schalter ex = {
[7M0] 122: &as,
[B02] 123: 289,156,310,193,
[B11] 124: 56,0,215,
[870] 125: 2,
[7E0] 126: "E",
[BD0] 127: };
[000] 128:
[PE3] 129: typedef struct win {
[LU2] 130: struct win *next;
[SU4] 131: word left, top, right, bottom;
[6S2] 132: word curx,cury;
[C21] 133: } win;
[000] 134:
[MU6] 135: int px,py,pz,dx,dy,dz,cx,cy,cz,tx,ty,tz;
[AT2] 136: double vx,vy,vz;
[000] 137:
[BK1] 138: win win3;
[AK1] 139: win win2;
[CK1] 140: win win4;
[P52] 141: win win1 = {
[BX0] 142: &win2,
[UD1] 143: 1,1,105,105,
[WW0] 144: 103,62,
```

Listing 1. Mit dem Werkzeug »cad.c« konstruieren Sie Ihre Welten.


```
[BD0] 145: };
[R52] 146: win win2 = {
[DX0] 147: &win3,
[1N1] 148: 107,1,211,105,
[WW0] 149: 103,62,
[BD0] 150: };
[S52] 151: win win3 = {
[FX0] 152: &win4,
[6N1] 153: 213,1,317,105,
[WW0] 154: 103,62,
[BD0] 155: };
[000] 156:
[T52] 157: win win4 = {
[9X0] 158: &win1,
[AN1] 159: 1,125,317,198,
[GE0] 160: 0,0,
[BD0] 161: };
[000] 162:
[000] 163:
[WE3] 164: typedef struct set {
[TU2] 165: struct set *next;
[SA2] 166: object *first;
[421] 167: } set;
[000] 168:
[3V3] 169: set *first=0,*actset=0;
[3R2] 170: object *actobj=0;
[RP4] 171: lamp *actlamp=0,*firstlamp=0;
[8S4] 172: vector nullpunkt, richtung;
[000] 173:
[8M3] 174: void setwin(win *wi);
[074] 175: void showwin(win *begin);
[PM6] 176: void drawlinewin(win *wi,int x0,int y0,
[0B3] 177: int x1,int y1,word c);
[6L6] 178: void line3d(win *wi,int x1,int y1,int z1,
[9P4] 179: int x2,int y2,int z2,word co);
[215] 180: void circle2d(win *wi,double x,
[753] 181: double y,double r);
[PM5] 182: void circle3d(win *wi,int x, int y,
[542] 183: int z,int r);
[LL3] 184: void clswin(win *wi);
[894] 185: void changeview(win *wi);
[R46] 186: void showpos(win *act,word x,word y);
[2C3] 187: object *genpoi(void);
[M23] 188: void genobs(void);
[B63] 189: lamp *genlamp(void);
[VB3] 190: object *gencir(void);
[AA7] 191: void rotvect(double dx,double dy,double dz,
[PT4] 192: double *x,double *y,double *z,
[N03] 193: int wi,char *str);
[EM6] 194: void rotobj(double x,double y,double z,
[W94] 195: object *o,int wi,char *str);
[5E3] 196: void rotset(set *s);
[PW6] 197: void movobj(int x,int y,int z,object *o);
[SD3] 198: void movset(set *s);
[U03] 199: void mergeword(void);
[M53] 200: int saveall(void);
[M84] 201: int loadall(int x,int y);
[SG2] 202: void cad(void);
[000] 203:
[W60] 204: /*
[W06] 205: setwin(wi): Aktiviert das Fenster wi
[170] 206: */
[000] 207:
[SN2] 208: void setwin(wi)
[TC1] 209: win *wi;
[P70] 210: {
[EF1] 211: word x,y;
[M62] 212: mouse_hide();
[935] 213: mouse_area(wi->left<<1,wi->top,
[364] 214: wi->right<<1,wi->bottom);
[202] 215: if(wi==&win1)
[Z65] 216: mouse_move((px<<1)+(wi->left<<1),
[3W1] 217: pz+wi->top);
[ZR0] 218: else
[302] 219: if(wi==&win2)
[Z65] 220: mouse_move((px<<1)+(wi->left<<1),
[0W1] 221: py+wi->top);
[ZR0] 222: else
[402] 223: if(wi==&win3)
[575] 224: mouse_move((pz<<1)+(wi->left<<1),
[0W1] 225: py+wi->top);
[ZR0] 226: else
[WR5] 227: mouse_move(wi->curx+(wi->left<<1),
[Z03] 228: wi->top+wi->cury);
[BA2] 229: mouse_show();
[U70] 230: }
[000] 231:
[W60] 232: /*
[H96] 233: showwin(begin): zeichnet alle Fenster
[T52] 234: umrandungen
[170] 235: */
[000] 236:
[ZA3] 237: void showwin(begin)
[YX1] 238: win *begin;
[P70] 239: {
[0D2] 240: win *wi=begin;
[1V0] 241: do{
[C55] 242: drawtangle(wi->left-1,wi->top-1,
[LK4] 243: wi->right+1,wi->bottom+1,215);
```

```
[G22] 244: wi=wi->next;
[5Y2] 245: }while(wi!=begin);
[U70] 246: }
[000] 247:
[W60] 248: /*
[PK4] 249: void drawline(x0,y0,x1,y1,c):
[KL4] 250: zeichnet Linie von x0,y0 nach
[Y72] 251: x1,y1 in Farbe c
[170] 252: */
[000] 253:
[KK5] 254: void drawlinewin(wi,x0,y0,x1,y1,c)
[TC1] 255: win *wi;
[CG2] 256: int x0,y0,x1,y1;
[W81] 257: word c;
[P70] 258: {
[095] 259: word ax,ay,dx,dy,ix,ix,h,of,ct;
[391] 260: ax=ay=0;
[ZC1] 261: if(x1>=x0)
[P70] 262: {
[KH2] 263: dx = x1-x0;ix = 1;
[U70] 264: }
[ZR0] 265: else
[P70] 266: {
[RR2] 267: dx = x0-x1;ix = -1;
[U70] 268: }
[1D1] 269: if(y1>=y0)
[P70] 270: {
[WH2] 271: dy = y1-y0;iy = 1;
[U70] 272: }
[ZR0] 273: else
[P70] 274: {
[1S2] 275: dy = y0-y1;iy = -1;
[U70] 276: }
[ZM1] 277: if(dx < dy)
[P70] 278: {
[NS2] 279: h=dx;dx=dy;dy=h;
[WT3] 280: ay=ix;ax=iy;ix=0;iy=0;
[U70] 281: }
[DP1] 282: of= dx >>1;
[XS0] 283: ct=1;
[000] 284:
[XS4] 285: if(x0>=wi->left && x0<=wi->right
[LW4] 286: && y0>=wi->top && y0<=wi->bottom)
[AA3] 287: put_pixel(x0,y0,c);
[000] 288:
[522] 289: while(dx>=ct)
[P70] 290: {
[PR2] 291: x0 += ix; y0 += ay;
[342] 292: ct ++; of += dy;
[LA1] 293: if(of>dx)
[P70] 294: {
[EF2] 295: of -= dx;x0 += ay;
[U51] 296: y0 += iy;
[U70] 297: }
[XS4] 298: if(x0>=wi->left && x0<=wi->right
[LW4] 299: && y0>=wi->top && y0<=wi->bottom)
[AA3] 300: put_pixel(x0,y0,c);
[U70] 301: }
[U70] 302: }
[000] 303:
[W60] 304: /*
[TD6] 305: line3d(wi,x1,y1,z1,x2,y2,z2,co): zeichnet
[L66] 306: dreidimensionale Linie in ein Fenster
[170] 307: */
[000] 308:
[BN5] 309: void line3d(wi,x1,y1,z1,x2,y2,z2,co)
[TC1] 310: win *wi;
[KB3] 311: int x1,y1,z1,x2,y2,z2;
[7D1] 312: word co;
[P70] 313: {
[PN2] 314: int xa,ya,xb,yb;
[M62] 315: mouse_hide();
[924] 316: xa=wi->left-dx-tx+x1/vx;
[D24] 317: xb=wi->left-dx-tx+x2/vx;
[HW3] 318: ya=wi->top-dz-tz-z1/vz;
[SW3] 319: yb=wi->top-dz-tz-z2/vz;
[K75] 320: drawlinewin(wi,xa,ya,xb,yb,co);
[782] 321: wi = wi->next;
[924] 322: xa=wi->left-dx-tx+x1/vx;
[D24] 323: xb=wi->left-dx-tx+x2/vx;
[8W3] 324: ya=wi->top-dy-ty-y1/vy;
[FW3] 325: yb=wi->top-dy-ty-y2/vy;
[K75] 326: drawlinewin(wi,xa,ya,xb,yb,co);
[782] 327: wi = wi->next;
[934] 328: xa=wi->left-dz-tz-z1/vz;
[D34] 329: xb=wi->left-dz-tz-z2/vz;
[8W3] 330: ya=wi->top-dy-ty-y1/vy;
[FW3] 331: yb=wi->top-dy-ty-y2/vy;
[K75] 332: drawlinewin(wi,xa,ya,xb,yb,co);
[BA2] 333: mouse_show();
[U70] 334: }
[000] 335:
[W60] 336: /*
[4K6] 337: circle2d(wi, x,y,r):zeichnet Kreis in ein
[5D1] 338: Fenster
[170] 339: */
[000] 340:
[GT3] 341: void circle2d(wi,x,y,r)
[TC1] 342: win *wi;
```



```

[512] 343: double x,y,r;
[P70] 344: {
[R55] 345:   int i,j,io=(int)(x-r),jo=(int)y;
[B94] 346:   for(i=(int)-r;i<=(int)r;i++)
[P70] 347:   {
[4U4] 348:     j=(int)sqrt((double)(r*r-i*i));
[925] 349:     drawlinewin(wi,io,jo,(int)(i+x),
[B52] 350:       (int)(j+y),30);
[A74] 351:     io=(int)(i+x);jo=(int)(j+y);
[U70] 352:   }
[3Z3] 353:   io=(int)(x+r);jo=(int)(y);
[S94] 354:   for(i=(int)r;i>=(int)-r;i--)
[P70] 355:   {
[ZV4] 356:     j=(int)-sqrt((double)(r*r-i*i));
[925] 357:     drawlinewin(wi,io,jo,(int)(i+x),
[B52] 358:       (int)(j+y),30);
[A74] 359:     io=(int)(i+x);jo=(int)(j+y);
[U70] 360:   }
[U70] 361: }
[000] 362:
[W60] 363: /*
[217] 364: circle3d(wi,x,y,z,r):zeichnet Kugel in ein
[5D1] 365: Fenster
[170] 366: */
[000] 367:
[TC1] 368: void circle3d(wi,x,y,z,r)
[TC1] 369: win *wi;
[B32] 370: int x,y,z,r;
[P70] 371: {
[VM1] 372:   int xa,ya;
[M62] 373:   mouse hide();
[044] 374:   line3d(wi,x,y,z,x,y,z,30);
[044] 375:   xa=wi->left-dx-tx+vx;
[FL3] 376:   ya=wi->top-dz-tz-z/vz;
[SB1] 377:   r=r/vz;
[PG5] 378:   circle2d(wi,(double)xa,(double)ya,
[XU1] 379:     (double)r);
[782] 380:   wi = wi->next;
[044] 381:   xa=wi->left-dx-tx+vx;
[7L3] 382:   ya=wi->top-dz-tz-z/vz;
[PG5] 383:   circle2d(wi,(double)xa,(double)ya,
[XU1] 384:     (double)r);
[782] 385:   wi = wi->next;
[A04] 386:   xa=wi->left-dz-tz-z/vz;
[7L3] 387:   ya=wi->top-dy-ty-y/vy;
[PG5] 388:   circle2d(wi,(double)xa,(double)ya,
[XU1] 389:     (double)r);
[BA2] 390:   mouse_show();
[U70] 391: }
[000] 392:
[W60] 393: /*
[L75] 394: clswin(win):löscht ein Fenster
[170] 395: */
[000] 396:
[3N2] 397: void clswin(wi)
[TC1] 398: win *wi;
[P70] 399: {
[EF1] 400:   word x,y;
[UD5] 401:   for(x=wi->left;x<=wi->right;x++)
[9K5] 402:   for(y=wi->top;y<=wi->bottom;y++)
[ZW2] 403:     put_pixel(x,y,0);
[U70] 404: }
[W60] 405: /*
[M06] 406: changeview(wi):berechnet Ansicht neu
[T03] 407: für alle Fenster
[170] 408: */
[SA3] 409: void changeview(wi)
[TC1] 410: win *wi;
[P70] 411: {
[A82] 412:   set *s=first;
[T02] 413:   polygon *p;
[PK1] 414:   sphere *sh;
[CK1] 415:   object *o;
[U13] 416:   lamp *l=firstlamp;
[F11] 417:   int i;
[M62] 418:   mouse hide();
[TV1] 419:   clswin(wi);
[0V2] 420:   clswin(wi->next);
[FY3] 421:   clswin(wi->next->next);
[000] 422:
[DB1] 423:   while(s)
[P70] 424:   {
[012] 425:     o=s->first;
[XA1] 426:     while(o)
[P70] 427:     {
[2D3] 428:       if(o->type==POLYGON)
[P70] 429:       {
[SE3] 430:         p=&(o->form.polygon);
[000] 431:
[VC4] 432:         for(i=0;i<p->num_vec-1;i++)
[P70] 433:         {
[CY4] 434:           line3d(wi,(int)p->vector[i].x,
[3F3] 435:             (int)p->vector[i].y,
[7F3] 436:             (int)p->vector[i].z,
[EP3] 437:             (int)p->vector[i+1].x,
[GP3] 438:             (int)p->vector[i+1].y,
[124] 439:             (int)p->vector[i+1].z,4);
[U70] 440:         }
[CY4] 441:         line3d(wi,(int)p->vector[i].x,

```

```

[3F3] 442:         (int)p->vector[i].y,
[7F3] 443:         (int)p->vector[i].z,
[6E3] 444:         (int)p->vector[0].x,
[AE3] 445:         (int)p->vector[0].y,
[XW3] 446:         (int)p->vector[0].z,4);
[U70] 447:       }
[ZR0] 448:     }
[P70] 449:   }
[5F3] 450:   sh=&(o->form.sphere);
[D85] 451:   circle3d(&win1,(int)sh->vector.x,
[533] 452:     (int)sh->vector.y,
[733] 453:     (int)sh->vector.z,
[5W2] 454:     (int)sh->radius);
[U70] 455: }
[9R1] 456: o=o->next;
[U70] 457: }
[1S1] 458: s=s->next;
[U70] 459: }
[HA1] 460: while(1)
[P70] 461: {
[GX4] 462:   line3d(&win1,(int)l->vector.x,
[TH5] 463:     (int)l->vector.y,(int)l->vector.z,
[PH5] 464:     (int)l->vector.x,(int)l->vector.y,
[KA3] 465:     (int)l->vector.z,150);
[4U1] 466:   l= l->next;
[U70] 467: }
[C05] 468: line3d(&win1,(int)nullpunkt.x,
[PW5] 469:   (int)nullpunkt.y,(int)nullpunkt.z,
[ML5] 470:   (int)richtung.x,(int)richtung.y,
[493] 471:   (int)richtung.z,35);
[BA2] 472: mouse_show();
[U70] 473: }
[000] 474:
[W60] 475: /*
[ZL5] 476: showpos(act,x,y):zeigt aktuelle
[7M5] 477: dreidimensionale Mausposition an
[170] 478: */
[4G3] 479: void showpos(act,x,y)
[EE1] 480: win *act;
[EF1] 481: word x,y;
[P70] 482: {
[172] 483:   int xo,yo,zo;
[583] 484:   x = x- (act->left<<1);
[4D2] 485:   y = y- act->top;
[081] 486:   x=x>>1;
[RG4] 487:   if(act==&win1||act==&win2)
[AV0] 488:     pz=x;
[TG4] 489:   if(act==&win1||act==&win3)
[HV0] 490:     pz=y;
[U42] 491:   if(act==&win2)
[EV0] 492:     py=y;
[W42] 493:   if(act==&win3)
[LB2] 494:     {pz=x;py=y;}
[783] 495:   xo=cx;yo=cy;zo=cz;
[XX3] 496:   cx=(int)((px+dx+tx)*vx);
[X14] 497:   cy=(int)((py+dy+ty)*vy);
[B24] 498:   cz=(int)((pz+dz+tz)*vz);
[D02] 499:   gotoxy(2,17);
[452] 500:   print("X:",215);
[000] 501:
[ZD1] 502:   if(cx!=xo)
[P70] 503:   {
[D02] 504:     gotoxy(2,17);
[8N2] 505:     print("X: ",215);
[F02] 506:     gotoxy(4,17);
[LA2] 507:     printnum(cx);
[U70] 508:   }
[2E1] 509:   if(cy!=yo)
[P70] 510:   {
[H02] 511:     gotoxy(2,18);
[9N2] 512:     print("Y: ",215);
[L02] 513:     gotoxy(4,18);
[RA2] 514:     printnum(cy);
[U70] 515:   }
[5E1] 516:   if(cz!=zo)
[P70] 517:   {
[N02] 518:     gotoxy(2,19);
[AN2] 519:     print("Z: ",215);
[R02] 520:     gotoxy(4,19);
[VA2] 521:     printnum(cz);
[U70] 522:   }
[U70] 523: }
[000] 524:
[W60] 525: /*
[MP6] 526: object *genpol(): generiert ein polygon
[170] 527: */
[000] 528:
[DH2] 529: object *genpol()
[P70] 530: {
[8U2] 531:   word ready=0,x,y;
[6A3] 532:   int xo,yo,zo,flag=0;
[4G2] 533:   win *act=&win1;
[CK1] 534:   object *o;
[T02] 535:   polygon *p;
[000] 536:

```

Listing 1. Mit dem Werkzeug »cad.c« konstruieren Sie Ihre Welten.


```

[T64] 537: o=malloc(sizeof(object));
[YT0] 538: if(!o)
[TL1] 539: return(0);
[HA4] 540: memset(o,0,sizeof(object));
[VM1] 541: o->next=0;
[8V2] 542: o->type=POLYGON;
[000] 543:
[SE3] 544: p=&(o->form.polygon);
[X52] 545: p->num_vec=0;
[UY1] 546: setwin(act);
[BA2] 547: mouse_show();
[FF3] 548: while(mouse_press());
[F62] 549: mouse_key(0);
[L62] 550: mouse_key(1);
[000] 551:
[1V0] 552: do{
[CF3] 553: while(!mouse_press())
[P70] 554: {
[ZT2] 555: mouse_pos(&x,&y);
[G03] 556: showpos(act,x,y);
[U70] 557: }
[FF3] 558: while(mouse_press());
[000] 559:
[MB4] 560: /* wechseln des Fensters */
[000] 561:
[VH2] 562: if(mouse_key(1))
[P70] 563: {
[ZT2] 564: mouse_pos(&x,&y);
[SN4] 565: act->curx = x- (act->left<<1);
[RU3] 566: act->cury = y- act->top;
[EL2] 567: act = act->next;
[Y42] 568: if(act==&win4)
[EL2] 569: act = act->next;
[UY1] 570: setwin(act);
[SH2] 571: if(mouse_key(0))
[391] 572: ready=1;
[U70] 573: }
[XS2] 574: /* Punkt setzen */
[SH2] 575: if(mouse_key(0))
[P70] 576: {
[MA6] 577: p->vector[p->num_vec].x=(double)cx;
[VA6] 578: p->vector[p->num_vec].y=(double)cy;
[ZB6] 579: p->vector[p->num_vec].z=(double)cz;
[G32] 580: p->num_vec++;
[K91] 581: if(!flag)
[P70] 582: {
[Y31] 583: flag++;
[783] 584: xo=cx;yo=cy;zo=cz;
[SK4] 585: line3d(&win1,cx,cy,cz,cx,cy,
[VV0] 586: cz,4);
[U70] 587: }
[ZR0] 588: else
[ZM4] 589: line3d(&win1,xo,yo,zo,cx,cy,
[VV0] 590: cz,4);
[783] 591: xo=cx;yo=cy;zo=cz;
[U70] 592: }
[U70] 593: }
[Z25] 594: while(!ready && p->num_vec<MAXVECTS);
[AA2] 595: if(p->num_vec)
[DL3] 596: line3d(&win1,xo,yo,zo,
[6E3] 597: (int)p->vector[0].x,
[AE3] 598: (int)p->vector[0].y,
[XW3] 599: (int)p->vector[0].z,4);
[ZR0] 600: else
[P70] 601: {
[R71] 602: free(o);
[GK0] 603: o=0;
[U70] 604: }
[SM1] 605: return(o);
[U70] 606: }
[000] 607:
[W60] 608: /*
[CU4] 609: genobs(): generiert Observer
[170] 610: */
[000] 611:
[Z52] 612: void genobs()
[P70] 613: {
[DX3] 614: word ready=0,x,y,flag=0;
[4G2] 615: win *act=&win1;
[000] 616:
[UY1] 617: setwin(act);
[BA2] 618: mouse_show();
[FF3] 619: while(mouse_press());
[F62] 620: mouse_key(0);
[L62] 621: mouse_key(1);
[000] 622:
[1V0] 623: do{
[CF3] 624: while(!mouse_press())
[P70] 625: {
[ZT2] 626: mouse_pos(&x,&y);
[G03] 627: showpos(act,x,y);
[U70] 628: }
[FF3] 629: while(mouse_press());
[000] 630:
[MB4] 631: /* wechseln des Fensters */
[000] 632:
[VH2] 633: if(mouse_key(1))
[P70] 634: {
[ZT2] 635: mouse_pos(&x,&y);

```

```

[SN4] 636: act->curx = x- (act->left<<1);
[RU3] 637: act->cury = y- act->top;
[EL2] 638: act = act->next;
[Y42] 639: if(act==&win4)
[EL2] 640: act = act->next;
[UY1] 641: setwin(act);
[SH2] 642: if(mouse_key(0))
[391] 643: ready=1;
[U70] 644: }
[XS2] 645: /* Punkt setzen */
[SH2] 646: if(mouse_key(0))
[P70] 647: {
[K91] 648: if(!flag)
[P70] 649: {
[Y31] 650: flag++;
[M64] 651: nullpunkt.x=(double)cx;
[V64] 652: nullpunkt.y=(double)cy;
[Z74] 653: nullpunkt.z=(double)cz;
[Z33] 654: line3d(&win1,cx,cy,cz,cx,
[0E1] 655: cy,cz,35);
[U70] 656: }
[ZR0] 657: else
[P70] 658: {
[391] 659: ready=1;
[LT3] 660: richtung.x=(double)cx;
[ST3] 661: richtung.y=(double)cy;
[XT3] 662: richtung.z=(double)cz;
[C05] 663: line3d(&win1,(int)nullpunkt.x,
[PW5] 664: (int)nullpunkt.y,(int)nullpunkt.z,
[ML5] 665: (int)richtung.x,(int)richtung.y,
[493] 666: (int)richtung.z,35);
[U70] 667: }
[U70] 668: }
[U70] 669: }
[742] 670: while(!ready);
[U70] 671: }
[000] 672:
[W60] 673: /*
[RM4] 674: genlamp(): generiert Lampe im
[X93] 675: dreidimensionalen Raum
[170] 676: */
[000] 677:
[BC2] 678: lamp *genlamp()
[P70] 679: {
[8U2] 680: word ready=0,x,y;
[4G2] 681: win *act=&win1;
[881] 682: lamp *l;
[000] 683:
[NY3] 684: l=malloc(sizeof(lamp));
[ST0] 685: if(!l)
[TL1] 686: return(0);
[KW3] 687: memset(l,0,sizeof(lamp));
[NM1] 688: l->next=0;
[UY1] 689: setwin(act);
[BA2] 690: mouse_show();
[FF3] 691: while(mouse_press());
[F62] 692: mouse_key(0);
[L62] 693: mouse_key(1);
[000] 694:
[1V0] 695: do{
[CF3] 696: while(!mouse_press())
[P70] 697: {
[ZT2] 698: mouse_pos(&x,&y);
[G03] 699: showpos(act,x,y);
[U70] 700: }
[FF3] 701: while(mouse_press());
[000] 702:
[MB4] 703: /* wechseln des Fensters */
[000] 704:
[VH2] 705: if(mouse_key(1))
[P70] 706: {
[ZT2] 707: mouse_pos(&x,&y);
[SN4] 708: act->curx = x- (act->left<<1);
[RU3] 709: act->cury = y- act->top;
[EL2] 710: act = act->next;
[Y42] 711: if(act==&win4)
[EL2] 712: act = act->next;
[UY1] 713: setwin(act);
[SH2] 714: if(mouse_key(0))
[791] 715: ready=2;
[U70] 716: }
[XS2] 717: /* Punkt setzen */
[SH2] 718: if(mouse_key(0))
[P70] 719: {
[P23] 720: l->vector.x=(double)cx;
[XZ3] 721: l->vector.y=(double)cy;
[404] 722: l->vector.z=(double)cz;
[SK4] 723: line3d(&win1,cx,cy,cz,cx,cy,
[E21] 724: cz,150);
[391] 725: ready=1;
[U70] 726: }
[U70] 727: }
[742] 728: while(!ready);
[ET1] 729: if(ready==2)
[P70] 730: {
[E71] 731: free(l);
[AK0] 732: l=0;
[U70] 733: }
[NM1] 734: return(l);

```



```

[U70] 735: }
[000] 736:
[W60] 737: /*
[9N4] 738: gencir(): generiert Kugel im
[X3] 739: dreidimensionalen Raum
[170] 740: */
[000] 741:
[6H2] 742: object *gencir()
[P70] 743: {
[8U2] 744: word ready=0,x,y;
[WR2] 745: double xo,yo,zo;
[XW1] 746: int flag=0;
[4G2] 747: win *act=&win1;
[CK1] 748: object *o;
[2W1] 749: sphere *sh;
[000] 750:
[T64] 751: o=malloc(sizeof(object));
[YT0] 752: if(!o)
[TL1] 753: return(0);
[HA4] 754: memset(o,0,sizeof(object));
[VM1] 755: o->next=0;
[YN2] 756: o->type=SPHERE;
[000] 757:
[5F3] 758: sh=&(o->form.sphere);
[UY1] 759: setwin(act);
[BA2] 760: mouse_show();
[FF3] 761: while(mouse_press());
[F62] 762: mouse_key(0);
[L62] 763: mouse_key(1);
[000] 764:
[1V0] 765: do{
[CF3] 766: while(!mouse_press())
[P70] 767: {
[ZT2] 768: mouse_pos(&x,&y);
[G03] 769: showpos(act,x,y);
[U70] 770: }
[FF3] 771: while(mouse_press());
[000] 772:
[MB4] 773: /* wechseln des Fensters */
[000] 774:
[VH2] 775: if(mouse_key(1))
[P70] 776: {
[ZT2] 777: mouse_pos(&x,&y);
[SN4] 778: act->curx = x- (act->left<<1);
[RU3] 779: act->cury = y- act->top;
[EL2] 780: act = act->next;
[Y42] 781: if(act==&win4)
[EL2] 782: act = act->next;
[UY1] 783: setwin(act);
[SH2] 784: if(mouse_key(0))
[791] 785: ready=2;
[U70] 786: }
[XS2] 787: /* Punkt setzen */
[SH2] 788: if(mouse_key(0))
[P70] 789: {
[K91] 790: if(!flag)
[P70] 791: {
[Y31] 792: flag++;
[VY3] 793: sh->vector.x=(double)cx;
[0Z3] 794: sh->vector.y=(double)cy;
[S23] 795: sh->vector.z=(double)cz;
[SK4] 796: line3d(&win1,cx,cy,cz,cx,cy,
[VV0] 797: cz,4);
[PS4] 798: xo=(double)cx;yo=(double)cy;
[VB2] 799: zo=(double)cz;
[U70] 800: }
[ZR0] 801: else
[P70] 802: {
[391] 803: ready=1;
[GH1] 804: xo=cx-xo;
[SH1] 805: yo=cy-yo;
[0K1] 806: zo=cz-zo;
[C64] 807: sh->radius=sqrt((double){
[2H4] 808: (double)xo*xo+yo*yo+zo*zo});
[D85] 809: circle3d(&win1,(int)sh->vector.x,
[533] 810: (int)sh->vector.y,
[733] 811: (int)sh->vector.z,
[5W2] 812: (int)sh->radius);
[U70] 813: }
[U70] 814: }
[U70] 815: }
[742] 816: while(!ready);
[ET1] 817: if(ready==2)
[P70] 818: {
[R71] 819: free(o);
[GK0] 820: o=0;
[U70] 821: }
[SM1] 822: return(o);
[U70] 823: }
[000] 824:
[W60] 825: /*
[RM6] 826: rotvect(): dreht Vector x,y,z um dx,dy,
[KK2] 827: dz mit Winkel wi
[170] 828: */
[000] 829:
[D16] 830: void rotvect(dx,dy,dz,x,y,z,wi,str)
[FV3] 831: double dx,dy,dz,*x,*y,*z;
[1B1] 832: int wi;

```

```

[7L1] 833: char *str;
[P70] 834: {
[1B5] 835: double xn,yn,tx,ty,w,*nx,*ny,*nz;
[4X3] 836: w=(double)wi*3.1415927/180;
[Y22] 837: switch(*str)
[P70] 838: {
[681] 839: case 'X':
[681] 840: case 'x':
[E52] 841: xn=*z;tx=dz;
[NZ1] 842: yn=*y;nx=z;
[412] 843: ty=dy;ny=y;
[P01] 844: break;
[981] 845: case 'Y':
[981] 846: case 'y':
[K12] 847: xn=*x;tx=dx;
[FZ1] 848: yn=*z;nx=x;
[A12] 849: ty=dz;ny=z;
[P01] 850: break;
[C81] 851: case 'Z':
[C81] 852: case 'z':
[K12] 853: xn=*x;tx=dx;
[DZ1] 854: yn=*y;nx=x;
[412] 855: ty=dy;ny=y;
[P01] 856: break;
[8E1] 857: default:
[8C1] 858: return;
[U70] 859: }
[KT5] 860: *nx=(xn-tx)*cos(w)-(yn-ty)*sin(w)+tx;
[HT5] 861: *ny=(yn-ty)*cos(w)+(xn-tx)*sin(w)+ty;
[U70] 862: }
[000] 863:
[W60] 864: /*
[SF4] 865: rotobj(): dreht Object o um
[8A3] 866: x,y,z mit Winkel wi
[170] 867: */
[000] 868:
[DN4] 869: void rotobj(x,y,z,o,wi,str)
[D12] 870: double x,y,z;
[CK1] 871: object *o;
[1B1] 872: int wi;
[7L1] 873: char *str;
[P70] 874: {
[F11] 875: int i;
[0K2] 876: switch(o->type)
[P70] 877: {
[R82] 878: case POLYGON:
[KZ5] 879: for(i=0;i<o->form.polygon.num_vec;
[8F0] 880: i++)
[V63] 881: rotvect(x,y,z,&(o->
[CA5] 882: form.polygon.vector[i].x),&(o->
[DA5] 883: form.polygon.vector[i].y),&(o->
[C06] 884: form.polygon.vector[i].z),wi,str);
[P01] 885: break;
[932] 886: case SPHERE:
[V63] 887: rotvect(x,y,z,&(o->
[ZE4] 888: form.sphere.vector.x),&(o->
[0F4] 889: form.sphere.vector.y),&(o->
[Z45] 890: form.sphere.vector.z),wi,str);
[P01] 891: break;
[U70] 892: }
[U70] 893: }
[000] 894:
[W60] 895: /*
[PG3] 896: rotset(): dreht set s
[170] 897: */
[000] 898:
[NE2] 899: void rotset(s)
[Y81] 900: set *s;
[P70] 901: {
[B32] 902: char str[10];
[1B1] 903: int wi;
[CK1] 904: object *o;
[D12] 905: double x,y,z;
[000] 906:
[SZ1] 907: gotoxy(2,21);
[7N2] 908: print("W: ",215);
[UZ1] 909: gotoxy(4,21);
[E02] 910: getstr(str);
[RN2] 911: if(strlen(str)>0)
[892] 912: wi=atoi(str);
[WZ1] 913: gotoxy(2,22);
[0Z3] 914: print("Achse: ",215);
[Z02] 915: gotoxy(8,22);
[E02] 916: getstr(str);
[9N0] 917: if(s)
[P70] 918: {
[012] 919: o=s->first;
[5N0] 920: if(o)
[0K2] 921: switch(o->type)
[P70] 922: {
[R82] 923: case POLYGON:
[9C5] 924: x=o->form.polygon.vector[0].x;
[DC5] 925: y=o->form.polygon.vector[0].y;
[HC5] 926: z=o->form.polygon.vector[0].z;
[P01] 927: break;

```

Listing 1. Mit dem Werkzeug »cad.c« konstruieren Sie Ihre Welten.


```

[932] 928:      case SPHERE:
[TL4] 929:          x=o->form.sphere.vector.x;
[XL4] 930:          y=o->form.sphere.vector.y;
[1M4] 931:          z=o->form.sphere.vector.z;
[P01] 932:          break;
[U70] 933:      }
[XA1] 934:      while(o)
[P70] 935:      {
[4S3] 936:          rotobj(x,y,z,o,wi,str);
[9R1] 937:          o=o->next;
[U70] 938:      }
[U70] 939:  }
[U70] 940:  }
[000] 941:  }
[W60] 942:  /*
[NM6] 943:  movobj(x,y,z,o): bewegt Objekt o um x,y,z
[N31] 944:  weiter
[170] 945:  */
[000] 946:
[FE3] 947:  void movobj(x,y,z,o)
[NS1] 948:  int x,y,z;
[CK1] 949:  object *o;
[P70] 950:  {
[F11] 951:      int i;
[0K2] 952:      switch(o->type)
[P70] 953:      {
[R82] 954:          case POLYGON:
[KZ5] 955:              for(i=0; i<o->form.polygon.num_vec;
[8F0] 956:                  i++)
[P70] 957:              {
[5A5] 958:                  o->form.polygon.vector[i].x +=
[LN1] 959:                      (double)x;
[9A5] 960:                  o->form.polygon.vector[i].y +=
[NN1] 961:                      (double)y;
[DA5] 962:                  o->form.polygon.vector[i].z +=
[RN1] 963:                      (double)z;
[U70] 964:              }
[P01] 965:              break;
[932] 966:          case SPHERE:
[RZ5] 967:              o->form.sphere.vector.x +=(double)x;
[ZZ5] 968:              o->form.sphere.vector.y +=(double)y;
[706] 969:              o->form.sphere.vector.z +=(double)z;
[P01] 970:              break;
[U70] 971:      }
[U70] 972:  }
[000] 973:  }
[W60] 974:  /*
[WW4] 975:  movset(s):bewegt set s im Raum
[170] 976:  */
[000] 977:
[8E2] 978:  void movset(s)
[Y81] 979:  set *s;
[P70] 980:  {
[P72] 981:      int tx,ty,tz;
[CK1] 982:      object *o;
[B32] 983:      char str[10];
[SZ1] 984:      gotoxy(2,21);
[8N2] 985:      print("X:      ",215);
[UZ1] 986:      gotoxy(4,21);
[E02] 987:      getstr(str);
[RN2] 988:      if(strlen(str)>0)
[FA2] 989:          tx=atoi(str);
[WZ1] 990:      gotoxy(2,22);
[9N2] 991:      print("Y:      ",215);
[YZ1] 992:      gotoxy(4,22);
[E02] 993:      getstr(str);
[RN2] 994:      if(strlen(str)>0)
[KA2] 995:          ty=atoi(str);
[002] 996:      gotoxy(2,23);
[AN2] 997:      print("Z:      ",215);
[Z02] 998:      gotoxy(4,23);
[E02] 999:      getstr(str);
[RN2] 1000:      if(strlen(str)>0)
[NA2] 1001:          tz=atoi(str);
[9N0] 1002:      if(s)
[P70] 1003:      {
[012] 1004:          o=s->first;
[XA1] 1005:          while(o)
[P70] 1006:          {
[EA3] 1007:              movobj(tx,ty,tz,o);
[9R1] 1008:              o=o->next;
[U70] 1009:          }
[U70] 1010:      }
[U70] 1011:  }
[000] 1012:  }
[W60] 1013:  /*
[MW6] 1014:  mergeworld(): berechnet Ansicht der Welt
[7P4] 1015:  nach dem gesetzten Observer
[170] 1016:  */
[000] 1017:
[8X2] 1018:  void mergeworld()
[P70] 1019:  {
[ZC2] 1020:      set *s=0,*sl=0;
[VK3] 1021:      int tx,ty,tz,wil,wi2;
[9Z2] 1022:      object *o=0,*old=0;
[881] 1023:      lamp *l;

```

```

[6F1] 1024:      s=first;
[M33] 1025:      /* transformation */
[7R3] 1026:      tx=(int)-nullpunkt.x;
[BR3] 1027:      ty=(int)-nullpunkt.y;
[FR3] 1028:      tz=(int)-nullpunkt.z;
[DB1] 1029:      while(s)
[P70] 1030:      {
[012] 1031:          o=s->first;
[XA1] 1032:          while(o)
[P70] 1033:          {
[EA3] 1034:              movobj(tx,ty,tz,o);
[Y01] 1035:              old=o;
[9R1] 1036:              o=o->next;
[U70] 1037:          }
[PR0] 1038:          sl=s;
[1S1] 1039:          s=s->next;
[9N0] 1040:          if(s)
[V21] 1041:              if(old)
[VD3] 1042:                  old->next=s->first;
[GY1] 1043:              if(sl!=first)
[ZC1] 1044:                  free(sl);
[7R0] 1045:              else
[0U1] 1046:                  sl->next=0;
[U70] 1047:          }
[CA2] 1048:          actset=first;
[D62] 1049:          l=firstlamp;
[HA1] 1050:          while(l)
[P70] 1051:          {
[DP4] 1052:              l->vector.x += -nullpunkt.x;
[NP4] 1053:              l->vector.y += -nullpunkt.y;
[XP4] 1054:              l->vector.z += -nullpunkt.z;
[RP1] 1055:              l=l->next;
[U70] 1056:          }
[V74] 1057:      /*Rotation des Ursprung */
[CM4] 1058:      if(richtung.z!=nullpunkt.z)
[P70] 1059:      {
[FK6] 1060:          wil=(int)(atan((richtung.x-nullpunkt.x)/
[MK4] 1061:              (richtung.z-nullpunkt.z))*
[DY1] 1062:              180/3.1415927);
[TK6] 1063:          wi2=(int)(atan((richtung.y-nullpunkt.y)/
[MK4] 1064:              (richtung.z-nullpunkt.z))*
[DY1] 1065:              180/3.1415927);
[6F1] 1066:          s=first;
[9N0] 1067:          if(s)
[P70] 1068:          {
[012] 1069:              o=s->first;
[XA1] 1070:              while(o)
[P70] 1071:              {
[Y64] 1072:                  rotobj(0.0,0.0,0.0,o,wil,"y");
[W64] 1073:                  rotobj(0.0,0.0,0.0,o,-wi2,"x");
[9R1] 1074:                  o=o->next;
[U70] 1075:              }
[U70] 1076:          }
[000] 1077:      }
[D62] 1078:      l=firstlamp;
[HA1] 1079:      while(l)
[P70] 1080:      {
[6C5] 1081:          rotvect(0.0,0.0,0.0,&(l->vector.x),&
[M82] 1082:              (l->vector.y),&
[MC3] 1083:              (l->vector.z),wil,"y");
[6C5] 1084:          rotvect(0.0,0.0,0.0,&(l->vector.x),&
[M82] 1085:              (l->vector.y),&
[RB3] 1086:              (l->vector.z),-wi2,"x");
[RP1] 1087:          l=l->next;
[U70] 1088:      }
[U70] 1089:  }
[M54] 1090:      richtung.x=richtung.y=0;
[Y72] 1091:      richtung.z=1;
[N17] 1092:      nullpunkt.x=nullpunkt.y=nullpunkt.z=0;
[U70] 1093:  }
[000] 1094:  }
[W60] 1095:  /*
[XG6] 1096:  saveall(): Speichert 3d daten als Welt ab
[170] 1097:  */
[282] 1098:  int saveall()
[P70] 1099:  {
[H92] 1100:      char name[30];
[000] 1101:
[S52] 1102:      mergeworld();
[SZ2] 1103:      changeview(&win1);
[000] 1104:
[D14] 1105:      w1.object=first->first;
[V23] 1106:      w1.lamp=firstlamp;
[000] 1107:
[402] 1108:      gotoxy(2,24);
[LF3] 1109:      print("Name:      ",215);
[902] 1110:      gotoxy(7,24);
[852] 1111:      getstr(name);
[402] 1112:      gotoxy(2,24);
[E23] 1113:      print("      ",215);
[PK4] 1114:      return(saveworld(&w1,name));
[U70] 1115:  }
[000] 1116:  }
[W60] 1117:  /*
[U55] 1118:  loadall(x,y): lädt 3D Daten ein
[170] 1119:  */

```



```
[000] 1120:
[ST2] 1121: int loadall(x,y)
[ZD1] 1122: int x,y;
[P70] 1123: {
[H92] 1124:   char name[30];
[881] 1125:   lamp *l;
[CK1] 1126:   object *o;
[000] 1127:
[F02] 1128:   gotoxy(x,y);
[LF3] 1129:   print("Name:           ",215);
[E62] 1130:   gotoxy(x+5,y);
[852] 1131:   getstr(name);
[F02] 1132:   gotoxy(x,y);
[E23] 1133:   print("           ",215);
[S52] 1134:   mergeworld();
[VL1] 1135:   if(actset)
[K52] 1136:     free(actset);
[BN3] 1137:   if(loadworld(&w1,name))
[P70] 1138:   {
[553] 1139:     firstlamp=w1.lamp;
[AK1] 1140:     if(!first)
[NL5] 1141:       first=actset=malloc(sizeof(set));
[7D2] 1142:     first->next=0;
[724] 1143:     first->first=w1.object;
[YP1] 1144:     actlamp=0;
[6F1] 1145:     actobj=0;
[UL1] 1146:     return(1);
[U70] 1147:   }
[000] 1148:
[AK1] 1149:   if(!first)
[NL5] 1150:     first=actset=malloc(sizeof(set));
[AK1] 1151:   if(!first)
[UL1] 1152:     return(1);
[000] 1153:
[553] 1154:   firstlamp=w1.lamp;
[7D2] 1155:   first->next=0;
[724] 1156:   first->first=w1.object;
[000] 1157:
[2T2] 1158:   o=first->first;
[5N0] 1159:   if(o)
[PA2] 1160:     while(o->next)
[9R1] 1161:       o=o->next;
[5G1] 1162:     actobj=o;
[D62] 1163:     l=firstlamp;
[2N0] 1164:     if(l)
[AA2] 1165:       while(l->next)
[RP1] 1166:         l=l->next;
[PS1] 1167:     actlamp=l;
[TL1] 1168:     return(0);
[U70] 1169:   }
[000] 1170:
[W60] 1171: /*
[XT4] 1172:   cad(): Editor für Körperdaten
[MF2] 1173:   der neuen Welt
[170] 1174: */
[000] 1175:
[XG1] 1176: void cad()
[P70] 1177: {
[8U2] 1178:   word x,y,ready=0;
[172] 1179:   int xo,yo,zo;
[E32] 1180:   char str[20];
[DG2] 1181:   win *act=&win4;
[WW2] 1182:   schalter *sch=&ex;
[1L2] 1183:   object *newobj;
[EA2] 1184:   lamp *newlamp;
[1R2] 1185:   nullpunkt.x = 0;
[5R2] 1186:   nullpunkt.y = 0;
[9R2] 1187:   nullpunkt.z = 0;
[B92] 1188:   richtung.x = 0;
[E92] 1189:   richtung.y = 0;
[H92] 1190:   richtung.z = 0;
[A02] 1191:   textattr(0);
[0E1] 1192:   clrscr();
[UY1] 1193:   setwin(act);
[M92] 1194:   showwin(act);
[P72] 1195:   showsch(sch);
[902] 1196:   gotoxy(6,15);
[XC4] 1197:   print("DOWN          NORTH \
[FN1] 1198:   WEST",215);
[W42] 1199:   gotoxy(38,19);
[K42] 1200:   print("P",215);
[T42] 1201:   gotoxy(38,22);
[U42] 1202:   print("X",215);
[U42] 1203:   gotoxy(38,23);
[B42] 1204:   print("I",215);
[V42] 1205:   gotoxy(38,24);
[P42] 1206:   print("T",215);
[AY3] 1207:   mouse_def(pointer,17,6);
[BA2] 1208:   mouse_show();
[F62] 1209:   mouse_key(0);
[B02] 1210:   px=py=pz=52;
[102] 1211:   tx=ty=tz=0;
[S02] 1212:   dx=dy=dz=-52;
[112] 1213:   vx=vy=vz=10;
[D02] 1214:   gotoxy(2,17);
[A92] 1215:   print("X:0",215);
[H02] 1216:   gotoxy(2,18);
[TC2] 1217:   print("Y:0 ",215);
```

```
[N02] 1218:   gotoxy(2,19);
[0K2] 1219:   print("Z:0 ",215);
[SZ2] 1220:   changeview(&win1);
[1V0] 1221:   do{
[CF3] 1222:     while(!mouse_press())
[P70] 1223:     {
[ZT2] 1224:       mouse_pos(&x,&y);
[G03] 1225:       showpos(act,x,y);
[2W3] 1226:       act->curx=x-act->left;
[FR3] 1227:       act->cury=y-act->top;
[081] 1228:       x=x>>1;
[U70] 1229:     }
[M62] 1230:   mouse_hide();
[WM4] 1231:   switch(testsch(x,y,&ex,&sch))
[P70] 1232:   {
[N41] 1233:     case 1:
[391] 1234:       ready=1;
[S52] 1235:       mergeworld();
[P01] 1236:       break;
[L51] 1237:     case 13:
[S52] 1238:       mergeworld();
[SZ2] 1239:       changeview(&win1);
[P01] 1240:       break;
[F51] 1241:     case 12:
[7P1] 1242:       saveall();
[P01] 1243:       break;
[B51] 1244:     case 11:
[F22] 1245:       /* loadall(); */
[K32] 1246:       loadall(2,24);
[SZ2] 1247:       changeview(&win1);
[P01] 1248:       break;
[751] 1249:     case 10:
[0D1] 1250:       genobs();
[SZ2] 1251:       changeview(&win1);
[P01] 1252:       break;
[D51] 1253:     case 9:
[E53] 1254:       newlamp=genlamp();
[Z82] 1255:       if(!firstlamp)
[773] 1256:         firstlamp=newlamp;
[ZR0] 1257:       else
[ZW3] 1258:         actlamp->next=newlamp;
[0W1] 1259:       if(newlamp)
[4Y2] 1260:         actlamp=newlamp;
[P01] 1261:       break;
[A51] 1262:     case 8:
[VN2] 1263:       movset(actset);
[SZ2] 1264:       changeview(&win1);
[P01] 1265:       break;
[C04] 1266:     case 7: /* Rotate Object */
[XN2] 1267:       rotset(actset);
[SZ2] 1268:       changeview(&win1);
[P01] 1269:       break;
[451] 1270:     case 6:
[SZ1] 1271:       gotoxy(2,21);
[8N2] 1272:       print("X:           ",215);
[UZ1] 1273:       gotoxy(4,21);
[E02] 1274:       getstr(str);
[RN2] 1275:       if(strlen(str)>0)
[FA2] 1276:         tx=atoi(str);
[WZ1] 1277:       gotoxy(2,22);
[9N2] 1278:       print("Y:           ",215);
[YZ1] 1279:       gotoxy(4,22);
[E02] 1280:       getstr(str);
[RN2] 1281:       if(strlen(str)>0)
[KA2] 1282:         ty=atoi(str);
[002] 1283:       gotoxy(2,23);
[AN2] 1284:       print("Z:           ",215);
[Z02] 1285:       gotoxy(4,23);
[E02] 1286:       getstr(str);
[RN2] 1287:       if(strlen(str)>0)
[NA2] 1288:         tz=atoi(str);
[SZ2] 1289:       changeview(&win1);
[P01] 1290:       break;
[151] 1291:     case 5:
[402] 1292:       gotoxy(2,24);
[XM2] 1293:       print("M:           ",215);
[G02] 1294:       gotoxy(4,24);
[E02] 1295:       getstr(str);
[RE5] 1296:       if(atoi(str)!=0.0 &&atoi(str)<=600.0)
[4D3] 1297:         vx=vy=vz=atoi(str);
[SZ2] 1298:       changeview(&win1);
[P01] 1299:       break;
[9D3] 1300:     case 4: /* neues Set */
[FF3] 1301:       while(mouse_press());
[CF4] 1302:       mouse_key(0);mouse_key(1);
[AK1] 1303:       if(!first)
[NL5] 1304:         first=actset=malloc(sizeof(set));
[ZR0] 1305:       else
[P70] 1306:       {
[9K5] 1307:         actset->next=malloc(sizeof(set));
[EK3] 1308:         actset->actset->next;
[U70] 1309:       }
[VL1] 1310:       if(actset)
[P70] 1311:       {
[VL2] 1312:         actset->next=0;
```

Listing 1. Mit dem Werkzeug »cad.c« konstruieren Sie Ihre Welten.


```
[2P2] 1313:      actset->first=0;
[6F1] 1314:      actobj=0;
[U70] 1315:      }
[P01] 1316:      break;
[0U2] 1317:      case 3:/* addpol */
[VL1] 1318:      if(actset)
[P70] 1319:      {
[5N2] 1320:          newobj=genpol();
[MK1] 1321:          if(actobj)
[LF3] 1322:              actobj->next=newobj;
[ZR0] 1323:          else
[7N3] 1324:              actset->first=newobj;
[EL1] 1325:              if(newobj)
[0F2] 1326:                  actobj=newobj;
[U70] 1327:          }
[P01] 1328:          break;
[M23] 1329:      case 2:/* adddspher */
[VL1] 1330:      if(actset)
[P70] 1331:      {
[RL2] 1332:          newobj=genccir();
[MK1] 1333:          if(actobj)
[LF3] 1334:              actobj->next=newobj;
[ZR0] 1335:          else
[7N3] 1336:              actset->first=newobj;
[EL1] 1337:              if(newobj)
[0F2] 1338:                  actobj=newobj;
[U70] 1339:          }
[P01] 1340:          break;
[8E1] 1341:      default:
[0S1] 1342:          sound(440);
[TA1] 1343:          delay(2);
[LR1] 1344:          nosound();
[P01] 1345:          break;
[U70] 1346:      }
[ZP6] 1347:      drawtangle(sch->left,sch->top,sch->right,
[FN4] 1348:          sch->bottom, sch->framecol);
[UY1] 1349:      setwin(act);
[BA2] 1350:      mouse show();
[4L2] 1351:      }while(!ready);
[M62] 1352:      mouse_hide();
[U70] 1353:      }
[NZ0] 1354:      #endif
```

Listing 1. Mit dem Werkzeug »cad.c« konstruieren Sie Ihre Welten.

```
[W60] 1: /*
[YL2] 2: Programm: calc.c
[0B7] 3: Funktion: enthält Ray-Tracing-Funktionen
[H84] 4: zur Berechnung der Welten
[9C5] 5: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 6: Autor: Matthias Oberdorfer
[HN4] 7: (c) 1992 DMV Widuch GmbH & Co. KG
[170] 8: */
[000] 9:
[KY1] 10: #ifndef CALC
[SY1] 11: #define CALC
[4L2] 12: #include"defs.h"
[3F2] 13: #include"vid.c"
[LK2] 14: #include"math.c"
[000] 15:
[B93] 16: /* Globale Variablen */
[UG1] 17: world w1;
[A85] 18: color background={0., 0.0, 0.5};
[KC2] 19: double amb=0.1;
[145] 20: color ambient={0.01, 0.01, 0.01};
[W54] 21: color black={0., 0., 0.};
[WT2] 22: word maxdepth=6;
[000] 23:
[000] 24:
[F57] 25: word hit_object(vector *point,vector *dir,
[413] 26:      object *obj,vector
[4C4] 27:      *int_point,vector *norma);
[HW6] 28: object *line_trace(vector *point, vector *
[7B4] 29:      dir, world *world, vector *
[0H4] 30:      int_point, vector *normal);
[KX5] 31: void colcpy(color *dest,color *src);
[6E6] 32: void add_color(color *dest, color *coll,
[AY1] 33:      color *col2);
[6E6] 34: void intens_col(color *dest, double k,
[6Y1] 35:      color *coll);
[GM4] 36: void Combine(color *colour,
[WA6] 37:      color *local_col,double local_weight,
[5P5] 38:      color *ref_col, double ref_weight,
[PL6] 39:      color *trans_col,double trans_weight);
[000] 40:
[6E7] 41: void trans_vector(object *o, vector*normal
[7M5] 42:      ,vector *dir,vector *trans_dir);
[357] 43: void ref_vector(vector *dir,vector*normal
[UN3] 44:      ,vector *ref_dir);
[U27] 45: void local_color(object *obj,vector *dir,
[BK5] 46:      vector *normal,vector *int_point,
[E95] 47:      world *world,color * local_col);
```

```
[6B6] 48: void surfaceCol(color *colour,object *
[K64] 49:      object,vector *intpoint);
[GF6] 50: void bckgrnd(color *colour,color *back,
[4T3] 51:      vector *dir,double am);
[XU6] 52: void raytrace(vector *point,vector *dir,
[KK6] 53:      word depth,world *world,color *colour);
[914] 54: void to_LUT(color *col);
[KD5] 55: word interpret_col(color *col);
[0P6] 56: word showpoint(word x,word y, world *w);
[000] 57:
[W60] 58: /*
[Z37] 59: word hit_object(point,dir,obj,int_point,
[381] 60:      norma);
[T96] 61: vector *point,*dir, *int_point,*norma;
[MN5] 62: überprüft ob ein eine Gerade ein
[144] 63: bestimmtes Objekt trifft
[170] 64: */
[000] 65:
[Z37] 66: word hit_object(point,dir,obj,int_point,
[V01] 67:      norma);
[T96] 68: vector *point,*dir, *int_point,*norma;
[4Y1] 69: object *obj;
[P70] 70: {
[BF6] 71:     polygon *polygon=&(obj->form.polygon);
[FS5] 72:     sphere *sphere=&(obj->form.sphere);
[7Y2] 73:     vector normal,a,b;
[792] 74:     double t,help;
[F11] 75:     int i;
[7E4] 76:     double l,m,n, orient,innen;
[000] 77:
[EU2] 78:     switch(obj->type)
[P70] 79:     {
[R82] 80:         case POLYGON:
[000] 81:
[FP4] 82:             /* Normale auf Ebene rechnen */
[295] 83:             vecsub(&a,&(polygon->vector[1]),
[FU3] 84:                 &(polygon->vector[0]));
[695] 85:             vecsub(&b,&(polygon->vector[2]),
[FU3] 86:                 &(polygon->vector[0]));
[1G3] 87:             vecprod(norma,&a,&b);
[A85] 88:             /* Normieren und Orientierung */
[SL4] 89:             help=-(dot(norma,point)-dot(
[WZ4] 90:                 &(polygon->vector[0]), norma));
[LG1] 91:             if(help<0)
[3G3] 92:                 vecmul(norma, -1/sqrt(
[V64] 93:                     dot(norma,norma)),norma);
[ZR0] 94:             else
[9C3] 95:                 vecmul(norma, 1/sqrt(
[V64] 96:                     dot(norma,norma)),norma);
[000] 97:
[TV4] 98:             /* Gerade und Ebene parallel ?*/
[000] 99:
[AV2] 100:             t=dot(norma,dir);
[000] 101:
[KAL] 102:             if(t==0.0)
[PU4] 103:                 return(0); /* no way;parallell */
[000] 104:
[ZY5] 105:             t=-(dot(norma,point)-dot(&(polygon->
[RK3] 106:                 vector[0]), norma))/t;
[000] 107:
[V61] 108:             if (t<=0)
[TL1] 109:                 return(0);
[XE3] 110:             /* Trifft aber falsch */
[000] 111:
[334] 112:             vecmul(int_point,t,dir);
[S16] 113:             vecadd(int_point,point,int_point);
[000] 114:
[R06] 115:             /* Liegt Schnittpunkt innerhalb von
[1E5] 116:             Polygon ? Die Summe aller winkel
[CF4] 117:             zwischen Schnittpunkt und
[R95] 118:             Eckpunkten muss 360 grad ergeben
[L42] 119:             t = 2 * PI sein
[170] 120:             */
[000] 121:
[UK0] 122:             t=0;
[3B5] 123:             for(i=0; i<polygon->num_vec; i++)
[P70] 124:             {
[KA5] 125:                 vecsub(&a,&(polygon->vector[i]),
[T12] 126:                     int_point);
[4D5] 127:                 vecsub(&b,&(polygon->vector[(i+1)%
[ED5] 128:                     polygon->num_vec],int_point);
[YR1] 129:                 norm(&a,&a);
[5S1] 130:                 norm(&b,&b);
[NG2] 131:                 innen=dot(&a,&b);
[NZ3] 132:                 if(innen>=1.0&&innen<=1.0)
[6N2] 133:                     t= t +acos(innen);
[ZR0] 134:                 else
[TL1] 135:                     return(0);
[U70] 136:             }
[000] 137:
[YW4] 138:             /* Nobody calculates perfect */
[000] 139:
[XY2] 140:             if(t>6.2830 && t<6.2833)
[UL1] 141:                 return(1);
[ZR0] 142:             else
[TL1] 143:                 return(0);
[000] 144:
[932] 145:             case SPHERE:
```



```
[LD2] 146: l=dot(dir,dir);
[MC6] 147: m=2*dir->x*(point->x-sphere->vector.x)
[FY5] 148: +2*dir->y*(point->y-sphere->vector.y)+
[106] 149: 2*dir->z*(point->z-sphere->vector.z);
[000] 150:
[HC5] 151: n=dot(&(amp;sphere->vector),&(sphere->
[CE5] 152: vector))+dot(point,point)-2*dot(&(
[F55] 153: sphere->vector),point)-sphere->
[NN3] 154: radius*sphere->radius;
[1V1] 155: t=m*m-4*1*n;
[000] 156:
[401] 157: if(t<0)
[924] 158: return(0); /* geht vorbei */
[000] 159:
[8C3] 160: if(t==0) /* tangiert */
[P70] 161: {
[7H1] 162: t=-m/(2*1);
[G31] 163: if(t<=0)
[TL1] 164: return(0);
[000] 165:
[334] 166: vecmul(int_point,t,dir);
[S16] 167: vecadd(int_point,point,int_point);
[R54] 168: vecsub(norma, int_point,
[663] 169: &(sphere->vector));
[000] 170:
[3W3] 171: /* raus aus der Kugel */
[000] 172:
[4W4] 173: vecmul(int_point,1.001,norma);
[8X4] 174: vecadd(int_point,int_point,
[663] 175: &(sphere->vector));
[000] 176:
[D23] 177: norm(norma,norma);
[UL1] 178: return(1);
[U70] 179: }
[P84] 180: else /* 2 Schnittpunkte */
[P70] 181: {
[1X1] 182: orient=1.00;
[SG3] 183: help=(-m+sqrt(t))/(2*1);
[0Z2] 184: t=(-m-sqrt(t))/(2*1);
[000] 185:
[YW4] 186: /* Nobody calculates perfect */
[000] 187:
[ZE1] 188: if(t<0.001)
[YU0] 189: t=0.0;
[DZ1] 190: if(help<0.001)
[7A1] 191: help=0.0;
[000] 192:
[801] 193: if(t>0)
[P70] 194: {
[LG1] 195: if(help>0)
[TY2] 196: t=(help<t)?help:t;
[ZR0] 197: else
[P70] 198: {
[9T2] 199: orient=-orient;
[9G2] 200: /* In der Kugel */
[U70] 201: }
[U70] 202: }
[ZR0] 203: else
[LG1] 204: if(help>0)
[P70] 205: {
[9T2] 206: orient=-orient;
[S91] 207: t=help;
[U70] 208: }
[ZR0] 209: else
[TG4] 210: return(0); /* kein treffer */
[0N2] 211: /* War wohl nix */
[000] 212:
[334] 213: vecmul(int_point,t,dir);
[S16] 214: vecadd(int_point,point,int_point);
[V34] 215: vecsub(norma,int_point,
[663] 216: &(sphere->vector));
[000] 217:
[3W3] 218: /* raus aus der Kugel */
[000] 219:
[D23] 220: norm(norma,norma);
[6T4] 221: vecmul(norma,orient,norma);
[UL1] 222: return(1);
[U70] 223: }
[U70] 224: }
[000] 225:
[U70] 226: }
[000] 227:
[W60] 228: /*
[KX5] 229: object *line_trace(vector *g1, vector
[6D5] 230: *g2,vector *treffpunkt,world *w);
[KK5] 231: schneidet gerade, die durch g1 und g2
[6F5] 232: geht; mit alle in Welt vorhandenen
[ME6] 233: Objecten und liefert den Schnittpunkt
[HZ6] 234: in treffpunkt zurück sowie einen Zeiger
[LE2] 235: auf das Objekt
[170] 236: */
[000] 237:
[AZ5] 238: object *line_trace(point, dir, world,
[023] 239: int_point, normal)
[RP6] 240: vector *point, *dir, *int_point, *normal;
[M42] 241: world *world;
[P70] 242: {
[8S1] 243: /*1.Objekt*/
```

```
[XN4] 244: object *next = world->object;
[PS5] 245: vector r,n; /*treffpunktspeicher */
[P66] 246: object *object=0; /* wird dargestellt */
[000] 247:
[X94] 248: /* Durchsuche alle Objekte */
[V62] 249: while(next) {
[DF6] 250: if(hit_object(point,dir,next,&r,&n)) {
[000] 251:
[2H4] 252: /* erstes Objekt getroffen */
[K12] 253: if(!object) {
[K52] 254: object=next;
[SL3] 255: veccpy(int_point,&r);
[HV2] 256: veccpy(normal,&n);
[U70] 257: }
[S26] 258: else /* nächstes Object aussuchen*/
[KK3] 259: if(distance(&r,point)<
[005] 260: distance(int_point,point)){
[K52] 261: object=next;
[SL3] 262: veccpy(int_point,&r);
[HV2] 263: veccpy(normal,&n);
[U70] 264: }
[U70] 265: }
[YV5] 266: next = next->next; /* und noch eins */
[U70] 267: }
[XK2] 268: return(object);
[U70] 269: }
[000] 270:
[W60] 271: /*
[104] 272: void colcpy(dest,src) :
[MA5] 273: kopiert Farbe von src nach dest
[170] 274: */
[000] 275: void colcpy(dest,src)
[TN3] 276: color *dest,*src;
[FT2] 277: {
[P70] 278: {
[3P2] 279: dest->r=src->r;
[PL2] 280: dest->g=src->g;
[LK2] 281: dest->b=src->b;
[U70] 282: }
[000] 283:
[W60] 284: /*
[S04] 285: addcolor(dest, coll, col2)
[AU3] 286: color *dest, *coll,*col2;
[0D3] 287: addiert zwei Farben
[170] 288: */
[000] 289: void add_color(dest, coll, col2)
[B45] 290: color *dest, *coll,*col2;
[AU3] 291: {
[P70] 292: {
[UX3] 293: dest->r = coll->r+col2->r;
[T03] 294: if(dest->r>MAXRED)
[1R2] 295: dest->r=MAXRED;
[2V3] 296: dest->g = coll->g+col2->g;
[GC3] 297: if(dest->g>MAXGREEN)
[LX2] 298: dest->g=MAXGREEN;
[UT3] 299: dest->b = coll->b+col2->b;
[A63] 300: if(dest->b>MAXBLUE)
[NT2] 301: dest->b=MAXBLUE;
[U70] 302: }
[000] 303:
[W60] 304: /*
[174] 305: intens_col(dest, k, coll)
[H03] 306: color *dest, *coll;
[LD1] 307: double k;
[5R6] 308: Dimmerfunktion zur Helligkeitsregelung
[621] 309: 0<=k<=1
[170] 310: */
[000] 311:
[W05] 312: void intens_col(dest, k, coll)
[H03] 313: color *dest, *coll;
[LD1] 314: double k;
[P70] 315: {
[S41] 316: if(k>1.0)
[GU0] 317: k=1.0;
[E83] 318: dest->r = k*coll->r;
[P53] 319: dest->g = k*coll->g;
[E43] 320: dest->b = k*coll->b;
[U70] 321: }
[000] 322:
[W60] 323: /*
[EM7] 324: void Combine(colour,local_col,local_weight,
[NB5] 325: ref_col, ref_weight,trans_col,
[P46] 326: trans_weight); kombiniert alle Farben
[KV2] 327: zur Gesamtfarbe
[170] 328: */
[000] 329:
[1H3] 330: void Combine(colour,
[T34] 331: local_col,local_weight,
[BK3] 332: ref_col, ref_weight,
[V74] 333: trans_col,trans_weight)
[TK5] 334: color *colour,*local_col,*ref_col,
[V42] 335: *trans_col;
[9M5] 336: double local_weight, ref_weight,
[5D2] 337: trans_weight;
[P70] 338: {
```

Listing 2. »calc.c« enthält Raytracing-Funktionen zur Berechnung der Welten.


```
[UM1] 339: color col;
[FH1] 340: col.r=1.0;
[FG1] 341: col.g=1.0;
[FG1] 342: col.b=1.0;
[OM5] 343: intens_col(colour,local_weight,
[1Z1] 344: local_col);
[SA6] 345: intens_col(&col,ref_weight,ref_col);
[V05] 346: add_color(colour,&col,colour);
[385] 347: intens_col(&col,trans_weight,
[D02] 348: trans_col);
[V05] 349: add_color(colour,&col,colour);
[U70] 350: }
[000] 351:
[W60] 352: /*
[PC6] 353: void ref_vector(normal, dir,ref_dir)
[1P6] 354: berechnet reflections Richtung auf eine
[KF2] 355: spiegelfläche
[170] 356: */
[000] 357:
[PC6] 358: void ref_vector(normal, dir,ref_dir)
[AY4] 359: vector *dir, *normal,*ref_dir;
[P70] 360: {
[ED1] 361: vector i;
[ND2] 362: vecncpy(&i,dir);
[PT1] 363: norm(&i,&i);
[173] 364: vecncpy(ref_dir, &i);
[YP2] 365: vecmul(&i,-1.0,&i);
[6H5] 366: vecmul(&i,dot(normal,&i)*2,normal);
[SH4] 367: vecadd(ref_dir,&i,ref_dir);
[U70] 368: }
[W60] 369: /*
[0Y5] 370: void trans_vector(obj, normal, dir,
[L02] 371: trans_dir);
[V86] 372: berechnet transmittierende Richtung
[U04] 373: durch eine durchsichtige Wand
[170] 374: */
[000] 375:
[5U5] 376: void trans_vector(obj, normal,dir,
[BT1] 377: trans_dir)
[865] 378: vector *trans_dir,*normal,*dir;
[4Y1] 379: object *obj;
[P70] 380: {
[ED1] 381: vector i;
[VN2] 382: double cosvieh;
[ND2] 383: vecncpy(&i,dir);
[PT1] 384: norm(&i,&i);
[PM4] 385: vecmul(trans_dir,1/obj->N,&i);
[YP2] 386: vecmul(&i,-1.0,&i);
[L04] 387: cosvieh=dot(normal,&i);
[Y16] 388: cosvieh = sqrt(1-(1-cosvieh*cosvieh)/
[BR2] 389: (obj->N*obj->N));
[FH6] 390: cosvieh = cosvieh-dot(normal,&i)/obj->N;
[394] 391: vecmul(&i,cosvieh,normal);
[TD5] 392: vecsub(trans_dir,trans_dir,&i);
[U70] 393: }
[000] 394:
[W60] 395: /*
[1W5] 396: void local_color(obj, dir , normal,
[GS4] 397: int_point, world, local_col)
[135] 398: berechnet den lokalen Farbwert
[X65] 399: des Schnittpunktes des Objekts
[181] 400: Input :
[024] 401: obj : Getroffenes Objekt
[B74] 402: dir : Richtung der Geraden
[3E4] 403: normal: Normale der Ebene
[2B6] 404: int_point: Koordinaten Schnittpunkt
[FB4] 405: world : dargestellte Welt
[XC1] 406: Output :
[4W3] 407: local_col: lokale Farbe
[170] 408: */
[000] 409:
[MM5] 410: void local_color(obj, dir, normal,
[GS4] 411: int_point, world, local_col)
[D95] 412: vector *normal,*int_point,*dir;
[6T2] 413: color *local_col;
[M42] 414: world *world;
[4Y1] 415: object *obj;
[P70] 416: {
[6Z3] 417: lamp *lamp=world->lamp;
[Z34] 418: vector L,V,dummy1,dummy2;
[342] 419: double licht;
[7T1] 420: color col;
[000] 421:
[TY2] 422: /* Erstmal dunkel */
[000] 423:
[R34] 424: colcpy(local_col,&black);
[000] 425:
[TZ3] 426: /* alle Lampen leuchten */
[D52] 427: while(lamp) {
[YN4] 428: /* Wie Hell wird der Punkt ? */
[B06] 429: /* abhängig von Winkel und Abstand */
[KH5] 430: /* Lampe auf der richtigen seite?*/
[VV5] 431: /* Schatten von anderen Objekten ? */
[000] 432:
[SZ2] 433: /* diffuses Licht */
[000] 434:
[D26] 435: vecsub(&L,int_point,&(lamp->vector));
[000] 436:
```

```
[972] 437: /* Schatten ? */
[H16] 438: if(obj==line_trace(&(lamp->vector),&L,
[SM3] 439: world,&dummy1,&dummy2))
[P70] 440: {
[000] 441:
[AR2] 442: vecmul(&L,-1.0,&L);
[BU1] 443: norm(&L,&L);
[000] 444:
[EV5] 445: /* lichtfarbe und stärke von Lampe*/
[000] 446:
[L16] 447: licht=dot(&L,normal)*obj->dif_const;
[000] 448:
[6F4] 449: /* Winkel Zwischen -90 und 90 */
[FP1] 450: if(licht>0)
[P70] 451: {
[8Z2] 452: /* Specular licht */
[1P2] 453: vecsub(&V,&L,dir);
[6G2] 454: vecmul(&V,.5,&V);
[HW1] 455: norm(&V,&V);
[Y95] 456: licht = licht + obj->spec_const*
[P83] 457: pow(dot(&V,normal),
[403] 458: obj->high_const);
[U70] 459: }
[ZR0] 460: else
[9B1] 461: licht = 0;
[000] 462:
[BV3] 463: /* so wirds beleuchtet */
[000] 464:
[FW3] 465: intens_col(&col,licht,
[1D2] 466: &(lamp->color));
[000] 467:
[1X2] 468: /* licht distance */
[000] 469:
[HB1] 470: if(licht)
[DS4] 471: intens_col(&col,exp(log(.5)/
[4N5] 472: lamp->intens*distance(int_point,
[ZF3] 473: &(lamp->vector))),&col);
[000] 474:
[B16] 475: add_color(local_col,local_col,&col);
[000] 476:
[U70] 477: }
[000] 478:
[SS2] 479: lamp=lamp->next;
[U70] 480: }
[Z03] 481: /* Umgebungslicht */
[8S6] 482: add_color(local_col,local_col,&ambient);
[U70] 483: }
[000] 484:
[W60] 485: /*
[E06] 486: surfaceCol(colour, object, intpoint)
[9A2] 487: color *colour;
[WH2] 488: object *object;
[GX2] 489: vector *intpoint;
[M45] 490: überzieht Objekte mit Texture
[170] 491: */
[000] 492:
[327] 493: void surfaceCol(colour, object, intpoint)
[9A2] 494: color *colour;
[WH2] 495: object *object;
[GX2] 496: vector *intpoint;
[P70] 497: {
[HA2] 498: vector a,b,c,d;
[0R3] 499: double winkel, x,y,r,r2;
[TD1] 500: word l,h;
[T05] 501: byte *co=object->text.map,col;
[2G2] 502: double ro,g,bl;
[000] 503:
[XA3] 504: if(object->text.map)
[P70] 505: {
[NK4] 506: /* Abstand von int und Anfang */
[3C3] 507: switch(object->type)
[P70] 508: {
[R82] 509: case POLYGON:
[PL5] 510: vecsub(&c,intpoint,&(object->form.
[PG3] 511: polygon.vector[0]));
[AB5] 512: vecsub(&a,&(object->form.polygon.
[TT5] 513: vector[1]),&(object->form.polygon.
[000] 514: vector[0]));
[2S1] 515: norm(&b,&a);
[GS1] 516: norm(&d,&c);
[000] 517:
[TR2] 518: winkel=dot(&b,&d);
[CZ4] 519: x=sqrt(dot(&c,&c))*winkel/object
[2X1] 520: ->text.big;
[264] 521: winkel=sin(acos(winkel));
[E24] 522: y=sqrt(dot(&c,&c))*winkel/object
[2X1] 523: ->text.big;
[X25] 524: h=((word)(x+.5))*object->text.x;
[D35] 525: l=((word)(y+.5))*object->text.y;
[000] 526:
[GR4] 527: col=(co+object->text.y*h+1);
[000] 528:
[4H4] 529: bl=((double)((int)(col/36)))/5;
[RV1] 530: col = col%36;
[144] 531: g=((double)((int)(col/6)))/5;
[KS1] 532: col = col%6;
[053] 533: ro=(double)(col)/5;
[000] 534:
```



```
[H24] 535: colour->r=ro *colour->r;
[TZ3] 536: colour->g=g *colour->g;
[EX3] 537: colour->b=bl *colour->b;
[P01] 538: break;
[000] 539:
[932] 540: case SPHERE:
[CS3] 541: x=intpoint->x-object->
[2N3] 542: form.sphere.vector.x;
[000] 543:
[7H5] 544: /* Wie wird texture aufgetragen */
[000] 545:
[64] 546: switch(object->text.typ)
[P70] 547: {
[N41] 548: case 1:
[KS3] 549: y=intpoint->z-object->
[4N3] 550: form.sphere.vector.z;
[P01] 551: break;
[S41] 552: case 2:
[GS3] 553: y=intpoint->y-object->
[3N3] 554: form.sphere.vector.y;
[P01] 555: break;
[U70] 556: }
[000] 557:
[8X4] 558: r=object->form.sphere.radius;
[1R5] 559: x=(asin(x/r)*r+r)/object->text.big;
[BR5] 560: y=(acos(y/r)*r+r)/object->text.big;
[X25] 561: h=((word)(x+.5))*object->text.x;
[D35] 562: l=((word)(y+.5))*object->text.y;
[000] 563:
[GR4] 564: col=(co+object->text.y*h+1);
[000] 565:
[4H4] 566: bl=((double)((int)(col/36)))/5;
[RV1] 567: col = col*36;
[144] 568: g=((double)((int)(col/6)))/5;
[KS1] 569: col = col*6;
[053] 570: ro=(double)(col)/5;
[000] 571:
[H24] 572: colour->r=ro *colour->r;
[TZ3] 573: colour->g=g *colour->g;
[EX3] 574: colour->b=bl *colour->b;
[U70] 575: }
[U70] 576: }
[ZR0] 577: else
[P70] 578: {
[WL4] 579: /* Keine Texture monochrom */
[966] 580: colour->r=object->color.r*colour->r;
[H36] 581: colour->g=object->color.g*colour->g;
[926] 582: colour->b=object->color.b*colour->b;
[U70] 583: }
[U70] 584: }
[000] 585:
[W60] 586: /*
[GF6] 587: void bckgrnd(color *colour,color *back,
[0T3] 588: vector *dir,double am):
[5N3] 589: erzeugt hintergrund
[170] 590: */
[000] 591:
[0E5] 592: void bckgrnd(colour,back,dir,am)
[U63] 593: color *colour, *back;
[YL1] 594: double am;
[ZZ1] 595: vector *dir;
[P70] 596: {
[NUL] 597: vector v,d;
[PD1] 598: double n;
[BD2] 599: veccpy(&d,dir);
[LS1] 600: norm(&d,&d);
[XY2] 601: v.x=0;v.y=1;v.z=0;
[SK3] 602: n=1.5707/acos(dot(&v,&d));
[XZ4] 603: colour->r= back->r*n+back->r*am;
[VX4] 604: colour->g= back->g*n+back->g*am;
[XW4] 605: colour->b= back->b*n+back->b*am;
[U70] 606: }
[000] 607:
[W60] 608: /*
[XU6] 609: void raytrace(vector *point,vector *dir,
[KK6] 610: word depth,world *world,color *colour);
[LT6] 611: liefert die Farbe und Intensität eines
[LZ3] 612: Bildpunktes einer Welt
[181] 613: Input :
[Z14] 614: point : Punkt auf Gerader
[9G4] 615: dir : Richtung der Geraden
[1Y3] 616: depth : recursive Tiefe
[ES4] 617: world : darzustellende Welt
[000] 618:
[XC1] 619: Output :
[ES4] 620: colour : Farbe und intensität
[170] 621: */
[000] 622:
[XL7] 623: void raytrace(point,dir,depth,world,colour)
[M63] 624: vector *point, *dir;
[302] 625: word depth;
[9A2] 626: color *colour;
[M42] 627: world *world;
[P70] 628: {
[A86] 629: color local_col, ref_col, trans_col;
[9F4] 630: vector ref_dir, trans_dir,
[P33] 631: int_point, normal;
[AR2] 632: object *object=0;
```

```
[000] 633:
[VN6] 634: /* finde nächste Object das vom Strahl
[5R2] 635: getroffen wird */
[000] 636:
[4C6] 637: object = line_trace(point, dir, world,
[YH3] 638: &int_point, &normal);
[000] 639:
[5Z4] 640: /* kein Object wird getroffen */
[WS1] 641: if(!object)
[6N3] 642: /* Hintergrund zeigen */
[NX5] 643: bckgrnd(colour,&background,dir,amb);
[ZR0] 644: else
[P70] 645: {
[DK4] 646: /* Farbe des Schnittpunkts */
[000] 647:
[LF5] 648: local_color(object,dir, &normal,
[G85] 649: &int_point, world, &local_col);
[000] 650:
[MW4] 651: /* Rekursionstiefe erreicht ? */
[000] 652:
[AE3] 653: if(depth == maxdepth)
[P70] 654: {
[NK3] 655: /* Alles ist schwarz */
[000] 656:
[GY3] 657: colcpy(&ref_col,&black);
[G84] 658: colcpy(&trans_col,&black);
[U70] 659: }
[ZR0] 660: else
[P70] 661: {
[KY3] 662: /* berechne reflektions
[SV2] 663: Strahlenvector */
[000] 664:
[E63] 665: if(object->mirror>0.0)
[P70] 666: {
[FY3] 667: ref_vector(&normal,dir,
[ZM1] 668: &ref_dir);
[065] 669: raytrace(&int_point, &ref_dir,
[154] 670: depth+1, world, &ref_col);
[U70] 671: }
[ZR0] 672: else
[TY3] 673: colcpy(&ref_col,&black);
[000] 674:
[V24] 675: /* berechne durchgehenden
[SV2] 676: Strahlenvector */
[000] 677:
[Z53] 678: if(object->trans>0.0)
[P70] 679: {
[6R5] 680: trans_vector(object,&normal,dir,
[D62] 681: &trans_dir);
[TM5] 682: raytrace(&int_point, &trans_dir,
[TL4] 683: depth+1, world, &trans_col);
[U70] 684: }
[ZR0] 685: else
[G84] 686: colcpy(&trans_col,&black);
[U70] 687: }
[3R2] 688: Combine(colour,
[RS4] 689: &local_col,object->material,
[G74] 690: &ref_col,object->mirror,
[FE4] 691: &trans_col,object->trans);
[000] 692:
[204] 693: /* Verbinde lamplight mit
[B92] 694: Objectfarbe */
[000] 695:
[PB6] 696: surfaceCol(colour, object,&int_point);
[000] 697: }
[U70] 698: }
[U70] 699: }
[000] 700:
[W60] 701: /*
[814] 702: void to_LUT(color *col):
[GV5] 703: übersetzt RGB-Farben in LUT-Werte
[M72] 704: Look Up Table
[170] 705: */
[000] 706:
[LX2] 707: void to_LUT(col)
[FU1] 708: color *col;
[P70] 709: {
[FY1] 710: double r,b,g;
[4V1] 711: r=col->r*2-1;
[MT1] 712: b=col->b*2-1;
[3U1] 713: g=col->g*2-1;
[000] 714:
[601] 715: if(r>0)
[P70] 716: {
[ZX1] 717: col->r = 1.0;
[US1] 718: col->b += r;
[9T1] 719: col->g += r;
[U70] 720: }
[ZR0] 721: else
[9M2] 722: col->r = 2*col->r;
[PZ0] 723: if(b>0)
[P70] 724: {
[FW1] 725: col->b = 1.0;
[AS1] 726: col->r += b;
[9R1] 727: col->g += b;
```

Listing 2. »calc.c« enthält Raytracing-Funktionen zur Berechnung der Welten.


```
[U70] 728: }
[ZR0] 729: else
[TH2] 730: col->b = 2*col->b;
[VZ0] 731: if(g>0)
[P70] 732: {
[YW1] 733: col->g = 1.0;
[YS1] 734: col->r += g;
[ER1] 735: col->b += g;
[U70] 736: }
[ZR0] 737: else
[KK2] 738: col->g = 2*col->g;
[GZ1] 739: if(col->r>1.0)
[ZT1] 740: col->r=1.0;
[6Y1] 741: if(col->b>1.0)
[FS1] 742: col->b=1.0;
[GY1] 743: if(col->g>1.0)
[YS1] 744: col->g=1.0;
[000] 745:
[U70] 746: }
[000] 747:
[W60] 748: /*
[A74] 749: word interpret_col(col):
[WX4] 750: interpretiert RGB-Wert in ein
[YX2] 751: Palettenregister
[170] 752: */
[000] 753:
[G54] 754: word interpret_col(col)
[FU1] 755: color *col;
[P70] 756: {
[0D5] 757: if(col->r<0||col->g<0||col->b<0)
[TL1] 758: return(0);
[215] 759: return((word)(col->r*5+.5)+(word)
[MC5] 760: (col->g*5+.5)*6+(word)(col->b*5+.5)*36);
[U70] 761: }
[000] 762:
[W60] 763: /*
[CL6] 764: word showpoint(word x, word y, world *w)
[K07] 765: verfolgt Strahl durch das Universum und
[EB7] 766: gibt Nummer des Palettenregisters zurück
[V06] 767: d.h. Umrechnung der von ray_trace()
[MZ5] 768: gelieferte Helligkeits und Farbinfos
[ZK1] 769: in Tabelle
[170] 770: */
[000] 771:
[YH3] 772: word showpoint(x,y,w)
[2N1] 773: word x, y;
[CC1] 774: world *w;
[P70] 775: {
[0Z1] 776: vector g1,g2;
[UM1] 777: color col;
[235] 778: word s=0; /* spiegeltiefe */
[YS3] 779: /* Gerade durch den Raum
[2P4] 780: durch zwei Punkte fixieren */
[M12] 781: g1.x = g1.y = 0;
[CL1] 782: g1.z=1;
[ZG4] 783: g2.x = -((double) (w->observer
[XD5] 784: ->p_width>>1)-x)/w->observer->dx;
[KB4] 785: g2.y = ((double) (w->observer
[WH5] 786: ->p_height>>1)-y)/w->observer->dy;
[E92] 787: g2.z = (double) 1;
[U34] 788: raytrace(&g1,&g2,s,w,&col);
[B82] 789: to_LUT(&col);
[9N4] 790: return(interpret_col(&col));
[U70] 791: }
[000] 792:
[W60] 793: /*
[SF4] 794: void show_world(world *w);
[1T5] 795: stellt Welt auf dem Bildschirm dar
[170] 796: */
[000] 797:
[883] 798: void show_world(w)
[CC1] 799: world *w;
[P70] 800: {
[EF1] 801: word x,y;
[PZ3] 802: for(y=0; y<w->observer->
[VP3] 803: p_height && !kbhit();y++)
[HZ3] 804: for(x=0; x<w->observer->
[UN3] 805: p_width && !kbhit(); x++)
[ZW5] 806: put_pixel(x, y, showpoint(x,y,w));
[KM1] 807: if(kbhit())
[R61] 808: getch();
[U70] 809: }
[000] 810:
[NZ0] 811: #endif
```

Listing 2. »calc.c« enthält Raytracing-Funktionen zur Berechnung der Welten.

```
[W60] 1: /*
[FN2] 2: Programm: defs.h
[1L6] 3: Funktion: Strukturen und Definitionen
[9C5] 4: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 5: Autor: Matthias Oberdorfer
```

```
[V04] 6: (c) 1992 Widuch GmbH & Co. KG
[170] 7: */
[000] 8:
[SY1] 9: #ifndef DEFS
[YY1] 10: #define DEFS
[000] 11:
[5F3] 12: #define POLYGON 0
[C83] 13: #define SPHERE 1
[RM3] 14: #define MAXVECTS 10
[DK3] 15: #define MAXRED 1.0
[1T3] 16: #define MAXGREEN 1.0
[KS3] 17: #define MAXBLUE 1.0
[000] 18:
[Y86] 19: typedef unsigned char byte; /* 1 Byte */
[V96] 20: typedef unsigned int word; /* 2 Byte */
[000] 21:
[VR5] 22: /* Vector oder Punkt im Universum */
[000] 23:
[0A4] 24: typedef struct vector {
[9X4] 25: double x,y,z; /* Position */
[XL1] 26: } vector;
[000] 27:
[5T3] 28: /* Polygondefinition */
[000] 29:
[884] 30: typedef struct polygon {
[0N5] 31: byte num_vec; /* Anzahl der Ecken */
[UD4] 32: vector vector[MAXVECTS];
[AV4] 33: /* ihre Eckpunkte */ } polygon;
[000] 34:
[W03] 35: /* Kugeldefinition */
[000] 36:
[F84] 37: typedef struct sphere {
[XS5] 38: vector vector; /* ihr Mittelpunkt */
[M85] 39: double radius; /* ihr Radius */
[SK1] 40: } sphere;
[000] 41:
[PG1] 42: /* Farbe */
[C04] 43: typedef struct color {
[TY1] 44: double r,g,b;
[XD1] 45: } color;
[000] 46:
[2V1] 47: /* Texture */
[3W2] 48: typedef struct {
[DU4] 49: word x,y; /* Maße der Map */
[0Z4] 50: word big; /* Streckung */
[706] 51: byte typ; /* wie auf die Oberfläche */
[RX2] 52: /* Nur für Kugeln */
[8N1] 53: byte *map;
[WV1] 54: } texture;
[000] 55:
[7N5] 56: /* sichtbares Object im Universum */
[000] 57:
[484] 58: typedef struct object {
[YU6] 59: struct object *next; /* nächstes Objekt */
[U45] 60: color color; /* Farbe im Spiel */
[TM5] 61: double material; /* Wieviel Prozent
[1L2] 62: des Materials */
[GF5] 63: double mirror; /* Wieviel Prozent
[2H3] 64: werden von Spiegelung
[6A2] 65: dargestellt */
[6X5] 66: double trans; /* Wieviel Prozent des
[Y74] 67: durchscheinenden Lichts
[SD3] 68: werden dargestellt */
[3V4] 69: double dif_const; /* 0<=x<=1 */
[XS4] 70: double spec_const; /* 0<=x<=1 */
[665] 71: double high_const; /* 0<=x<=100 */
[5R5] 72: double N; /* Velocitätsfaktor */
[E16] 73: texture text; /* Oberflächenfarbe */
[HV4] 74: byte type; /* Objecttyp */
[FF1] 75: union {
[DZ2] 76: polygon polygon;
[UA2] 77: sphere sphere;
[S91] 78: } form;
[HG1] 79: } object;
[000] 80:
[W60] 81: /*
[BA4] 82: Lichtquelle im Universum
[170] 83: */
[BT6] 84: typedef struct lamp { /* Lichtquellen */
[UD6] 85: struct lamp *next; /* nächste Lampe */
[HB4] 86: vector vector; /* Position */
[CH5] 87: color color; /* Farbe des Lichts */
[CC4] 88: double intens; /* Stärke */
[1A1] 89: } lamp;
[000] 90:
[W60] 91: /*
[824] 92: Beobachter im Universum
[170] 93: */
[N36] 94: typedef struct { /* wir / Bildschirm */
[WL5] 95: int p_width; /* Pixelbreite */
[2H5] 96: int p_height; /* Pixelhöhe */
[7P5] 97: int dx,dy; /* Abstand vom Screen */
[R22] 98: } observer;
[000] 99:
[W60] 100: /*
[RH2] 101: Neues Universum
[170] 102: */
[000] 103:
```



```
[PD6] 104: typedef struct { /* unsere neue Welt */
[PK4] 105:     lamp *lamp; /* erste Lampe */
[GG5] 106:     object *object; /* erstes Objekt */
[2G4] 107:     observer *observer; /* wir */
[SE1] 108: } world;
[000] 109:
[NZ0] 110: #endif
```

Listing 3. Strukturen und Definitionen beinhaltet »defs.h«.

```
[W60] 1: /*
[9W2] 2: Programm: graph.c
[M66] 3: Funktion: einfache Zeichenfunktionen
[9C5] 4: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 5: Autor: Matthias Oberdorfer
[V04] 6: (c) 1992 Widuch GmbH & Co. KG
[170] 7: */
[000] 8:
[Y32] 9: #ifndef GRAPH
[442] 10: #define GRAPH
[000] 11:
[3F2] 12: #include "vid.c"
[AL2] 13: #include "maus.c"
[000] 14:
[416] 15: void drawline(word x0, word y0, word x1,
[HL2] 16: word y1, word c);
[Y16] 17: void drawtangle(word left, word top,
[HB5] 18: word right, word bottom, word c);
[B65] 19: void fill(word x, word y, word c);
[000] 20:
[W60] 21: /*
[K56] 22: void drawline(x0, y0, x1, y1, c): zeichnet
[6P5] 23: Linie von x0, y0 nach x1, y1 in Farbe c
[170] 24: */
[000] 25:
[2F4] 26: void drawline(x0, y0, x1, y1, c)
[D13] 27: word x0, y0, x1, y1, c;
[P70] 28: {
[095] 29: word ax, ay, dx, dy, ix, ix, of, ct;
[391] 30: ax=ay=0;
[2C1] 31: if(x1>=x0)
[P70] 32: {
[KH2] 33: dx = x1-x0; ix = 1;
[U70] 34: }
[ZR0] 35: else
[P70] 36: {
[RR2] 37: dx = x0-x1; ix = -1;
[U70] 38: }
[1D1] 39: if(y1>=y0)
[P70] 40: {
[WH2] 41: dy = y1-y0; iy = 1;
[U70] 42: }
[ZR0] 43: else
[P70] 44: {
[1S2] 45: dy = y0-y1; iy = -1;
[U70] 46: }
[ZM1] 47: if(dx < dy)
[P70] 48: {
[NS2] 49: h=dx; dx=dy; dy=h;
[WT3] 50: ay=ix; ax=iy; ix=0; iy=0;
[U70] 51: }
[DP1] 52: of= dx >> 1;
[XS0] 53: ct=1;
[AA3] 54: put_pixel(x0, y0, c);
[522] 55: while(dx>=ct)
[P70] 56: {
[PR2] 57: x0 += ix; y0 += ay;
[342] 58: ct ++; of += dy;
[LA1] 59: if(of>dx)
[P70] 60: {
[EF2] 61: of -= dx; x0 += ay;
[U51] 62: y0 += iy;
[U70] 63: }
[AA3] 64: put_pixel(x0, y0, c);
[U70] 65: }
[U70] 66: }
[000] 67:
[W60] 68: /*
[B57] 69: void drawtangle(left, top, right, bottom, c):
[SM4] 70: zeichnet Rechteck in Farbe c;
[170] 71: */
[000] 72:
[XZ6] 73: void drawtangle(left, top, right, bottom, c)
[V05] 74: word left, top, right, bottom, c;
[P70] 75: {
[E91] 76: word i;
[0A2] 77: if(left>right)
[905] 78: {i=left; left=right; right=i;}
[EB2] 79: if(top>bottom)
[H35] 80: {i=top; top=bottom; bottom=i;}
[000] 81:
[XZ3] 82: for(i=left; i<=right; i++)
[P70] 83: {
```

```
[WD3] 84: put_pixel(i, top, c);
[YU3] 85: put_pixel(i, bottom, c);
[U70] 86: }
[Y14] 87: for(i=top; i<=bottom; i++)
[P70] 88: {
[MB3] 89: put_pixel(left, i, c);
[KN3] 90: put_pixel(right, i, c);
[U70] 91: }
[U70] 92: }
[000] 93:
[W60] 94: /*
[B96] 95: fill(x, y, co): füllt Rechteck mit Farbe
[TM0] 96: auf
[170] 97: */
[FY3] 98: fillhelp(x, y, co, c, dx, dy)
[6A3] 99: int x, y, co, c, dx, dy;
[P70] 100: {
[3A1] 101: int h, b;
[554] 102: while(get_pixel(x, y)==c)
[P70] 103: {
[DD1] 104: h=x, b=y;
[914] 105: while(get_pixel(h, b)==c)
[993] 106: put_pixel(h++, b, co);
[1P1] 107: h=x; b=y-1;
[914] 108: while(get_pixel(h, b)==c)
[783] 109: put_pixel(h, b--, co);
[FM1] 110: h=x-1; b=y;
[914] 111: while(get_pixel(h, b)==c)
[F93] 112: put_pixel(h--, b, co);
[ZN1] 113: h=x; b=y+1;
[914] 114: while(get_pixel(h, b)==c)
[T73] 115: put_pixel(h, b++, co);
[ZC2] 116: x=x+dx; y=y+dy;
[U70] 117: }
[U70] 118: }
[2S2] 119: void fill(x, y, co)
[G02] 120: word x, y, co;
[P70] 121: {
[VS3] 122: word c=get_pixel(x, y);
[591] 123: if(c==co)
[KC1] 124: return;
[FL4] 125: if(x<0 || y<0 || y>319 || y>199)
[KC1] 126: return;
[FL3] 127: fillhelp(x, y, co, c, 1, 1);
[E44] 128: fillhelp(x+1, y-1, co, c, 1, -1);
[184] 129: fillhelp(x-1, y-1, co, c, -1, -1);
[144] 130: fillhelp(x-1, y+1, co, c, -1, 1);
[U70] 131: }
[NZ0] 132: #endif
```

Listing 4. »graph.c« stellt einfache Zeichenfunktionen bereit.

```
[W60] 1: /*
[XM2] 2: Programm: load.c
[ZM6] 3: Funktion: enthält Speicherfunktionen
[R12] 4: für Welten
[9C5] 5: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 6: Autor: Matthias Oberdorfer
[V04] 7: (c) 1992 Widuch GmbH & Co. KG
[170] 8: */
[000] 9:
[3Z1] 10: #ifndef LOAD
[9Z1] 11: #define LOAD
[000] 12:
[8W2] 13: #include <stdio.h>
[2T2] 14: #include <alloc.h>
[4L2] 15: #include "defs.h"
[3H2] 16: #include "calc.c"
[000] 17:
[B06] 18: int saveworld(world *w, char *name);
[KY5] 19: int loadworld(world *w, char *name);
[000] 20:
[W60] 21: /*
[B06] 22: int saveworld(world *w, char *name);
[BX5] 23: speichert ein Welt mit Objekten und
[4F3] 24: Lampen unter Namen ab.
[GK3] 25: Output 0 : Kein Fehler
[ZY3] 26: 1 : Fehler beim Speichern
[170] 27: */
[000] 28:
[LP3] 29: int saveworld(w, name)
[CC1] 30: world *w;
[0U1] 31: char *name;
[P70] 32: {
[SB1] 33: FILE *fp;
[AU2] 34: lamp *l= w->lamp;
[M73] 35: object *o= w->object;
[F11] 36: int i;
[8W2] 37: /* Datei öffnen */
[7Z2] 38: fp=fopen(name, "wb");
[PV1] 39: if(fp!=NULL)
[P70] 40: {
[V03] 41: /* Welt speichern */
```

Listing 5. »load.c« enthält die Speicherfunktionen.


```
[BY4] 42: fwrite(w,sizeof(world), 1, fp);
[4K0] 43: i=0;
[HA1] 44: while(1)
[P70] 45: {
[UF0] 46:     i++;
[RP1] 47:     l=l->next;
[U70] 48: }
[T85] 49: /* Anzahl der Lampen und Lampen */
[V24] 50: fwrite(&i,sizeof(i),1,fp);
[UP1] 51: l=w->lamp;
[HA1] 52: while(1)
[P70] 53: {
[1N4] 54:     fwrite(l,sizeof(lamp),1, fp);
[RP1] 55:     l=l->next;
[U70] 56: }
[4K0] 57: i=0;
[XA1] 58: while(o)
[P70] 59: {
[UF0] 60:     i++;
[9R1] 61:     o=o->next;
[U70] 62: }
[VE5] 63: /* Anzahl der Objekte und Objekte */
[V24] 64: fwrite(&i,sizeof(i),1,fp);
[R32] 65: o=w->object;
[XA1] 66: while(o)
[P70] 67: {
[424] 68:     fwrite(o,sizeof(object),1, fp);
[000] 69: }
[3P3] 70: /* texture vorhanden ? */
[3E2] 71: if(o->text.map)
[685] 72:     fwrite(o->text.map,sizeof(byte),
[T04] 73:         o->text.x*o->text.y,fp);
[9R1] 74:     o=o->next;
[U70] 75: }
[YR2] 76: /* Und uns selbst */
[056] 77: fwrite(w->observer,sizeof(observer),
[HU0] 78:     1,fp);
[000] 79: }
[4C4] 80: /* Noch ein paar variablen */
[XD6] 81: fwrite(&background,sizeof(color),1,fp);
[SG5] 82: fwrite(&black,sizeof(color),1,fp);
[VX5] 83: fwrite(&ambient,sizeof(color),1,fp);
[026] 84: fwrite(&maxdepth,sizeof(word),1,fp);
[YA5] 85: fwrite(&amb,sizeof(double),1,fp);
[000] 86: }
[PT1] 87: fclose(fp);
[124] 88: return(0); /* Kein Fehler */
[U70] 89: }
[GD3] 90: return(-1); /* Fehler */
[U70] 91: }
[000] 92: }
[W60] 93: /*
[KY5] 94: int loadworld(world *w,char *name);
[4D5] 95: lädt eine Welt mit Objekten und
[3N2] 96: Lampen mit Namen.
[YM3] 97: Output 0 : Kein Fehler
[1K3] 98: !=0 : Fehler beim Laden
[170] 99: */
[000] 100:
[UM3] 101: int loadworld(w,name)
[CC1] 102: world *w;
[0U1] 103: char *name;
[P70] 104: {
[SB1] 105:     FILE *fp;
[X63] 106:     lamp *l=w->lamp,*ll;
[5X3] 107:     object *o=w->object,*ol;
[F11] 108:     int i;
[000] 109:
[HA1] 110:     while(1)
[P70] 111:     {
[1R0] 112:         ll=l;
[RP1] 113:         l=l->next;
[KB1] 114:         free(ll);
[U70] 115:     }
[XA1] 116:     while(o)
[P70] 117:     {
[AR0] 118:         ol=o;
[S15] 119:         if(o->text.map&o->text.x!=0&o->
[EH1] 120:             text.y!=0)
[FY2] 121:             free(o->text.map);
[9R1] 122:         o=o->next;
[VB1] 123:         free(ol);
[U70] 124:     }
[000] 125:
[4H2] 126:     if(w->observer)
[613] 127:         free(w->observer);
[000] 128:
[GL1] 129:     w->lamp=0;
[FW1] 130:     w->object=0;
[GB2] 131:     w->observer=0;
[8W2] 132:     /* Datei öffnen */
[Z22] 133:     fp=fopen(name,"rb");
[PV1] 134:     if(fp==NULL)
[P70] 135:     {
[V03] 136:         /* Welt speichern */
[6N4] 137:         fread(w,sizeof(world), 1, fp);
[T85] 138:         /* Anzahl der Lampen und Lampen */
[NT3] 139:         fread(&i,sizeof(i),1,fp);
[5P0] 140:         ll=0;
```

```
[YGL] 141:     while(i--)
[P70] 142:     {
[NY3] 143:         l=malloc(sizeof(lamp));
[ST0] 144:         if(!l)
[P70] 145:         {
[PT1] 146:             fclose(fp);
[VL1] 147:             return(2);
[U70] 148:         }
[DT0] 149:         if(ll)
[NV1] 150:             ll->next = l;
[ZR0] 151:         else
[VV1] 152:             w->lamp = l;
[5D4] 153:         fread(l,sizeof(lamp),1, fp);
[1R0] 154:         ll=l;
[U70] 155:     }
[WT1] 156:     l->next = 0;
[6G3] 157:     /* Anzahl der Objekte */
[NT3] 158:     fread(&i,sizeof(i),1,fp);
[BP0] 159:     ol=0;
[YGL] 160:     while(i--)
[P70] 161:     {
[T64] 162:         o=malloc(sizeof(object));
[YT0] 163:         if(!o)
[P70] 164:         {
[PT1] 165:             fclose(fp);
[VL1] 166:             return(2);
[U70] 167:         }
[GT0] 168:         if(ol)
[522] 169:             ol->next = o;
[ZR0] 170:         else
[052] 171:             w->object = o;
[000] 172:
[CN4] 173:         fread(o,sizeof(object),1, fp);
[000] 174:
[S15] 175:         if(o->text.map&o->text.x!=0&o->
[EH1] 176:             text.y!=0)
[P70] 177:         {
[Z95] 178:             o->text.map=malloc(o->text.x*o->
[FD3] 179:                 text.y*sizeof(byte));
[CL2] 180:             if(!o->text.map)
[P70] 181:             {
[TS3] 182:                 o->text.x=o->text.y=0;
[CB2] 183:                 o->text.map=0;
[Z93] 184:                 w->observer=malloc(
[Z23] 185:                     sizeof(observer));
[PT1] 186:                 fclose(fp);
[VL1] 187:                 return(2);
[U70] 188:             }
[H35] 189:             fread(o->text.map, sizeof(byte),
[F54] 190:                 o->text.x*o->text.y, fp);
[U70] 191:         }
[AR0] 192:         ol=o;
[U70] 193:     }
[YR2] 194:     /* Und uns selbst */
[D76] 195:     w->observer=malloc(sizeof(observer));
[GL2] 196:     if(!w->observer)
[P70] 197:     {
[PT1] 198:         fclose(fp);
[VL1] 199:         return(2);
[U70] 200:     }
[746] 201:     fread(w->observer,sizeof(observer),
[HU0] 202:         1,fp);
[4C4] 203:     /* Noch ein paar variablen */
[Y46] 204:     fread(&background,sizeof(color),1,fp);
[475] 205:     fread(&black,sizeof(color),1,fp);
[MK5] 206:     fread(&ambient,sizeof(color),1,fp);
[DP5] 207:     fread(&maxdepth,sizeof(word),1,fp);
[ZV4] 208:     fread(&amb,sizeof(double),1,fp);
[000] 209:
[PT1] 210:     fclose(fp);
[124] 211:     return(0); /* Kein Fehler */
[U70] 212: }
[D76] 213: w->observer=malloc(sizeof(observer));
[GD3] 214: return(-1); /* Fehler */
[U70] 215: }
[NZ0] 216: #endif
```

Listing 5. »load.c« enthält die Speicherfunktionen.

```
[W60] 1: /*
[5P2] 2: Programm: math.c
[US5] 3: Funktion: enthält mathematische
[MS3] 4: Funktionen zur Vektor-
[1F1] 5: rechnung
[9C5] 6: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 7: Autor: Matthias Oberdorfer
[V04] 8: (c) 1992 Widuch GmbH & Co. KG
[170] 9: */
[000] 10:
[B02] 11: #ifndef MATH
[H02] 12: #define MATH
[000] 13:
[4L2] 14: #include"defs.h"
[RN2] 15: #include<math.h>
```



```
[000] 16:
[6W6] 17: void vecsub(vector *r, vector *a, vector *b);
[F35] 18: double dot(vector *a, vector *b);
[X96] 19: double distance(vector *p1, vector *p2);
[7X6] 20: void vecsub(vector *r, vector *a, vector *b);
[SU6] 21: void vecadd(vector *r, vector *a, vector *b);
[RV6] 22: void vecmul(vector *r, double t, vector *a);
[4B6] 23: vecprod(vector *a, vector *b, vector *c);
[GV5] 24: void norm(vector *dest, vector *src);
[NH6] 25: void veccpy(vector *dest, vector *src);
[000] 26:
[000] 27:
[000] 28: /*
[W60] 29: double dot(double x, double y, double z);
[KE6] 30: rechnet Skalarprodukt zweier
[Y05] 31: Vektoren aus.
[R72] 32: */
[170] 33:
[000] 34: double dot(a, b)
[LB2] 35: vector *a, *b;
[ZV1] 36: {
[P70] 37:     return((a->x*b->x+a->y*b->y+a->z*b->z));
[YX5] 38: }
[U70] 39:
[000] 40: /*
[W60] 41: double distance(vector *p1, vector *p2);
[Y76] 42: rechnet betrag oder abstand
[0R4] 43: zweier Vektoren aus.
[WH3] 44: */
[170] 45:
[000] 46: double distance(p1, p2)
[5G3] 47: vector *p1, *p2;
[3B2] 48: {
[P70] 49:     vector t;
[TD1] 50:     vecsub(&t, p1, p2);
[WK2] 51:     return(sqrt(dot(&t, &t)));
[U04] 52: }
[U70] 53:
[000] 54: /*
[W60] 55: void vecsub(vector *r, vector *a,
[BA5] 56:     vector *b);
[4S1] 57: Vector Subtraktion r= a-b
[T14] 58: */
[170] 59:
[000] 60: void vecsub(r, a, b)
[6Y2] 61: vector *r, *a, *b;
[3A2] 62: {
[P70] 63:     r->x = a->x - b->x;
[6T2] 64:     r->y = a->y - b->y;
[ET2] 65:     r->z = a->z - b->z;
[PT2] 66: }
[U70] 67:
[000] 68: /*
[W60] 69: void vecadd(vector *r, vector *a,
[W75] 70:     vector *b);
[4S1] 71: Vector Addition r= a+b
[BG3] 72: */
[170] 73:
[000] 74: void vecadd(r, a, b)
[RV2] 75: vector *r, *a, *b;
[3A2] 76: {
[P70] 77:     r->x = a->x + b->x;
[2T2] 78:     r->y = a->y + b->y;
[AT2] 79:     r->z = a->z + b->z;
[KT2] 80: }
[U70] 81:
[000] 82: /*
[W60] 83: void vecmul(vector *r, double t, vector *a);
[597] 84: Multipliziert Vector mit konstanter r= t*a
[170] 85: */
[C13] 86: void vecmul(r, t, a)
[0X1] 87: vector *r, *a;
[WD1] 88: double t;
[P70] 89: {
[GX1] 90:     r->x = t*a->x;
[LX1] 91:     r->y = t*a->y;
[PK1] 92:     r->z = t*a->z;
[U70] 93: }
[000] 94:
[W60] 95: /*
[4B6] 96: vecprod(vector *c, vector *b, vector *a);
[X25] 97: bildet das Vektorprodukt c = a*b
[170] 98: */
[000] 99:
[T62] 100: vecprod(c, a, b)
[692] 101: vector *a, *b, *c;
[P70] 102: {
[GB4] 103:     c->x = a->y * b->z - a->z * b->y;
[YC4] 104:     c->y = a->x * b->z + a->z * b->x;
[8B4] 105:     c->z = a->x * b->y - a->y * b->x;
[U70] 106: }
[000] 107:
[W60] 108: /*
[NH6] 109: void veccpy(vector *dest, vector *src);
[GH3] 110: kopiert Vektor src in
[M22] 111: Vektor dest.
[170] 112: */
```

```
[000] 113:
[1P3] 114: void veccpy(dest, src)
[W03] 115: vector *dest, *src;
[P70] 116: {
[DR2] 117:     dest->x=src->x;
[MR2] 118:     dest->y=src->y;
[VR2] 119:     dest->z=src->z;
[U70] 120: }
[000] 121:
[W60] 122: /*
[P63] 123: void norm(dest, src)
[WL3] 124: normiert einen Vector
[170] 125: */
[000] 126:
[P63] 127: void norm(dest, src)
[VX2] 128: vector *dest, *src;
[P70] 129: {
[RZ4] 130:     double d=1/sqrt(dot(src, src));
[W63] 131:     vecmul(dest, d, src);
[U70] 132: }
[NZ0] 133: #endif
```

Listing 6. In »math.c« finden Sie die Funktionen zur Vektorrechnung.

```
[W60] 1: /*
[8R2] 2: Programm: maus.c
[2Z5] 3: Funktion: enthält Mausfunktionen
[9C5] 4: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 5: Autor: Matthias Oberdorfer
[V04] 6: (c) 1992 Widuch GmbH & Co. KG
[170] 7: */
[000] 8:
[U02] 9: #ifndef MAUS
[012] 10: #define MAUS
[000] 11:
[7M2] 12: #include<dos.h>
[4L2] 13: #include"defs.h"
[KF3] 14: #define MOUSEINT 0x33
[000] 15:
[1V4] 16: unsigned mouse_check(void);
[204] 17: void mouse_show(void);
[GW3] 18: void mouse_hide(void);
[3L5] 19: void mouse_move(word x, word y);
[Y56] 20: void mouse_def(byte *bitfield, word x,
[HA1] 21: word y);
[HD5] 22: void mouse_pos(word *x, word *y);
[S74] 23: word mouse_press(void);
[W64] 24: word mouse_key(word nr);
[446] 25: void mouse_area(word left, word top,
[S44] 26: word right, word bottom);
[000] 27:
[EA3] 28: byte pointer[64] = {
[C04] 29: 127, 255, 127, 255, 127, 255, 127, 255,
[S04] 30: 127, 255, 255, 255, 192, 129, 255, 255,
[C04] 31: 127, 255, 127, 255, 127, 255, 127, 255,
[N04] 32: 127, 255, 255, 255, 255, 255, 255, 255,
[WV2] 33: 128, 0, 128, 0, 128, 0, 128, 0,
[CE2] 34: 128, 0, 0, 0, 63, 126, 0, 0,
[WV2] 35: 128, 0, 128, 0, 128, 0, 128, 0,
[S22] 36: 128, 0, 0, 0, 0, 0, 0, 0,
[BD0] 37: };
[000] 38:
[W60] 39: /*
[9V6] 40: unsigned mouse_check(void): überprüft
[TH6] 41: ob ein Maustreiber installiert ist oder
[XZ0] 42: nicht.
[C83] 43: 0 : Kein Maustreiber
[8F3] 44: !=0 Anzahl der Knöpfe
[170] 45: */
[000] 46:
[6V3] 47: unsigned mouse_check()
[P70] 48: {
[R73] 49:     struct REGPACK reg;
[WD2] 50:     reg.r_ax = 0x00;
[DE3] 51:     intr(MOUSEINT, &reg);
[D16] 52:     return(reg.r_ax!=0xffff?0:reg.r_bx);
[U70] 53: }
[000] 54:
[W60] 55: /*
[066] 56: void mouse_show(): macht Mauszeiger
[TC1] 57: sichtbar
[170] 58: */
[000] 59:
[G23] 60: void mouse_show()
[P70] 61: {
[R73] 62:     struct REGPACK reg;
[0E2] 63:     reg.r_ax = 0x01;
[DE3] 64:     intr(MOUSEINT, &reg);
[U70] 65: }
[000] 66:
```

Listing 7. »maus.c« enthält die Mausfunktionen.


```
[W60] 67: /*
[E26] 68: void mouse_hide(): macht Mauszeiger
[HW1] 69:     unsichtbar
[170] 70: */
[000] 71:
[YY2] 72: void mouse_hide()
[P70] 73: {
[R73] 74:     struct REGPACK reg;
[4E2] 75:     reg.r_ax = 0x02;
[DE3] 76:     intr(MOUSEINT, &reg);
[U70] 77: }
[000] 78:
[W60] 79: /*
[BD5] 80: void mouse_def(): definiert das
[074] 81: Aussehen des Mauszeigers
[170] 82: */
[000] 83:
[6W4] 84: void mouse_def(bitfield,x,y)
[6L2] 85: byte *bitfield;
[EF1] 86: word x,y;
[P70] 87: {
[R73] 88:     struct REGPACK reg;
[0F2] 89:     reg.r_ax = 0x09;
[312] 90:     reg.r_bx = x;
[812] 91:     reg.r_cx = y;
[DL4] 92:     reg.r_es = FP_SEG(bitfield);
[XK4] 93:     reg.r_dx = FP_OFF(bitfield);
[DE3] 94:     intr(MOUSEINT, &reg);
[U70] 95: }
[000] 96:
[W60] 97: /*
[YF5] 98: void mouse_pos(x,y):liefert die
[4W2] 99: Mausposition x,y
[170] 100: */
[000] 101:
[GC3] 102: void mouse_pos(x,y)
[WX1] 103: word *x,*y;
[P70] 104: {
[R73] 105:     struct REGPACK reg;
[8E2] 106:     reg.r_ax = 0x03;
[DE3] 107:     intr(MOUSEINT, &reg);
[D12] 108:     *x=reg.r_cx;
[L12] 109:     *y=reg.r_dx;
[U70] 110: }
[000] 111:
[W60] 112: /*
[E27] 113: word mouse_press():liefert die Maustasten
[XV5] 114: status: bit 0= linker, bit 1 = rechter
[VT2] 115: bit 2=mittlerer
[170] 116: */
[000] 117:
[X73] 118: word mouse_press()
[P70] 119: {
[R73] 120:     struct REGPACK reg;
[8E2] 121:     reg.r_ax = 0x03;
[DE3] 122:     intr(MOUSEINT, &reg);
[4W2] 123:     return(reg.r_bx);
[U70] 124: }
[000] 125:
[W60] 126: /*
[D96] 127: word mouse_key(nr):liefert Anzahl wie
[BU5] 128: oft eine Maustaste gedrückt wurde
[L83] 129: 0 = linke Maustaste
[K83] 130: 1 = rechte Maustaste
[NP3] 131: 2 = mittlere Maustaste
[170] 132: */
[000] 133:
[X53] 134: word mouse_key(nr)
[ME1] 135: word nr;
[P70] 136: {
[R73] 137:     struct REGPACK reg;
[ME2] 138:     reg.r_ax = 0x06;
[S72] 139:     reg.r_bx = nr;
[DE3] 140:     intr(MOUSEINT, &reg);
[4W2] 141:     return(reg.r_bx);
[U70] 142: }
[000] 143:
[W60] 144: /*
[GG5] 145: void mouse_move(x,y): bewegt den
[1F4] 146: Mauscursor an Position x,y
[170] 147: */
[000] 148:
[MM3] 149: void mouse_move(x,y)
[EF1] 150: word x,y;
[P70] 151: {
[R73] 152:     struct REGPACK reg;
[CE2] 153:     reg.r_ax = 0x04;
[712] 154:     reg.r_cx = x;
[C12] 155:     reg.r_dx = y;
[DE3] 156:     intr(MOUSEINT, &reg);
[U70] 157: }
[000] 158:
[W60] 159: /*
[AX6] 160: void mouse_area(left,top,right,bottom):
[SY5] 161: legt Area fest, in der der Mauscursor
[D22] 162: bewegt werden darf
[170] 163: */
[000] 164:
[XK6] 165: void mouse_area(left,top ,right,bottom)
```

```
[LR4] 166: word left,top, right,bottom;
[P70] 167: {
[R73] 168:     struct REGPACK reg;
[SE2] 169:     reg.r_ax = 0x07;
[CM2] 170:     reg.r_cx = left;
[0S2] 171:     reg.r_dx = right;
[DE3] 172:     intr(MOUSEINT, &reg);
[WE2] 173:     reg.r_ax = 0x08;
[7H2] 174:     reg.r_cx = top;
[313] 175:     reg.r_dx = bottom;
[DE3] 176:     intr(MOUSEINT, &reg);
[U70] 177: }
[000] 178:
[NZ0] 179: #endif
```

Listing 7. »maus.c« enthält die Mausfunktionen.

```
[W60] 1: /*
[6N2] 2: Programm: ray.c
[S96] 3: Funktion: Verbindet CAD-, Textur- und
[YU2] 4: Rechenprogramme
[9C5] 5: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 6: Autor: Matthias Oberdorfer
[V04] 7: (c) 1992 Widuch GmbH & Co. KG
[170] 8: */
[000] 9:
[1Z1] 10: #ifndef RAY
[7Z1] 11: #define RAY
[000] 12:
[043] 13: #include<stdlib.h>
[8W2] 14: #include<stdio.h>
[RN2] 15: #include<math.h>
[2T2] 16: #include<alloc.h>
[B53] 17: #include<string.h>
[000] 18:
[3F2] 19: #include"vid.c"
[AL2] 20: #include"maus.c"
[1R2] 21: #include"graph.c"
[GB3] 22: #include"schalter.c"
[1L2] 23: #include"save.c"
[N83] 24: #include"texture.c"
[3H2] 25: #include"calc.c"
[DL2] 26: #include"load.c"
[AC2] 27: #include"cad.c"
[000] 28:
[000] 29:
[6W4] 30: word countlamp=0,countobj=0;
[000] 31:
[000] 32:
[XS2] 33: schalter ne2={
[470] 34: 0,
[1Z1] 35: 203,180,304,195,
[B11] 36: 56,0,215,
[870] 37: 2,
[U31] 38: "EXIT",
[BD0] 39: };
[000] 40:
[DU2] 41: schalter ex1={
[MP0] 42: &ne2,
[NZ1] 43: 203,165,304,178,
[B11] 44: 56,0,215,
[870] 45: 2,
[002] 46: "CALC WORLD",
[BD0] 47: };
[000] 48:
[3U2] 49: schalter cw1={
[4S0] 50: &ex1,
[0Z1] 51: 203,148,304,162,
[B11] 52: 56,0,215,
[870] 53: 2,
[A22] 54: "LOAD WORLD",
[BD0] 55: };
[000] 56:
[NS2] 57: schalter mdl={
[UR0] 58: &cw1,
[FY1] 59: 203,133,304,145,
[B11] 60: 56,0,215,
[870] 61: 2,
[Y12] 62: "SAVE WORLD",
[BD0] 63: };
[000] 64:
[NU2] 65: schalter mtl={
[CP0] 66: &mdl,
[1Z1] 67: 102,165,201,178,
[B11] 68: 56,0,215,
[870] 69: 2,
[EUI] 70: "EDIT OBSV",
[BD0] 71: };
[000] 72:
[8M2] 73: schalter eobl={
[CS0] 74: &mtl,
[CY1] 75: 102,148,201,162,
[B11] 76: 56,0,215,
[870] 77: 2,
```



```
[3T1] 78: "EDIT LAMP",
[BD0] 79: };
[000] 80:
[XS2] 81: schalter ell = {
[ZX0] 82:   &eobl,
[VK1] 83:   102,133,201,145,
[B11] 84:   56,0,215,
[870] 85:   2,
[WM1] 86:   "EDIT OBJ",
[BD0] 87:   };
[000] 88:
[9T2] 89: schalter eol = {
[MP0] 90:   &ell,
[NM1] 91:   1,165,100,178,
[B11] 92:   56,0,215,
[870] 93:   2,
[TU1] 94:   "MAKE TEXT",
[BD0] 95:   };
[000] 96:
[LV2] 97: schalter swl = {
[OR0] 98:   &eol,
[CM1] 99:   1,148,100,162,
[B11] 100:  56,0,215,
[870] 101:  2,
[KN1] 102:  "LOAD PIC",
[BD0] 103:  };
[000] 104:
[000] 105:
[KS2] 106: schalter ldl = {
[AT0] 107:   &swl,
[5M1] 108:   1,133,100,145,
[B11] 109:   56,0,215,
[870] 110:   2,
[GV1] 111:   "MAKE 3DATA",
[BD0] 112:   };
[000] 113:
[N63] 114: int count(void **f);
[2P4] 115: int loadtexture(object *o);
[US2] 116: void clssc(void);
[DH3] 117: void getcol(color *c);
[9C3] 118: void edobj(world *w);
[PK3] 119: void edlamp(world *w);
[4D3] 120: void edobs(world *w);
[935] 121: int save(int x,int y,world *w);
[000] 122:
[000] 123: /*
[000] 124:   count(f): zählt objektliste
[170] 125: */
[000] 126:
[232] 127: int count(f)
[391] 128: void **f;
[P70] 129: {
[G91] 130:   int i=0;
[T91] 131:   while(f)
[P70] 132:   {
[UF0] 133:     i++;
[7N0] 134:     f=f*f;
[U70] 135:   }
[KM1] 136:   return(i);
[U70] 137: }
[000] 138:
[W60] 139: /*
[PV5] 140: loadtexture(o) : lädt texture für
[PG5] 141: ein object o in den reservierten
[S12] 142: Speicher ein
[ZA3] 143: 1 falls erfolgreich
[170] 144: */
[000] 145:
[T63] 146: int loadtexture(o)
[CK1] 147: object *o;
[P70] 148: {
[SB1] 149:   FILE *fp;
[5B1] 150:   int i,j;
[BH1] 151:   byte col;
[991] 152:   byte *c;
[H92] 153:   char name[30];
[PV1] 154:   gotoxy(1,3);
[TM3] 155:   print("Dateiname: ",215);
[852] 156:   getstr(name);
[PV1] 157:   gotoxy(1,3);
[WT3] 158:   print(" ",215);
[222] 159:   fp=fopen(name,"rb");
[PH2] 160:   if(!fp)return(0);
[9W5] 161:   fread(&(o->text.x),sizeof(word),1,fp);
[BW5] 162:   fread(&(o->text.y),sizeof(word),1,fp);
[3E2] 163:   if(o->text.map)
[FY2] 164:     free(o->text.map);
[000] 165:
[ZA6] 166:   o->text.map=c=malloc(sizeof(byte)*o->
[FZ2] 167:     text.x*o->text.y);
[CL2] 168:   if(!o->text.map)
[P70] 169:   {
[PV1] 170:     gotoxy(1,3);
[Z85] 171:     print("Nicht genug Speicher",215);
[PT1] 172:     fclose(fp);
[TL1] 173:     return(0);
[U70] 174:   }
[NV3] 175:   for(i=0;i<o->text.x;i++)
```

```
[ZV3] 176:   for(j=0;j<o->text.y;j++)
[P70] 177:   {
[PC4] 178:     fread(&col,sizeof(col),1,fp);
[0A5] 179:     *(c+o->text.y*i+j)=(unsigned)col;
[U70] 180:   }
[PT1] 181:   fclose(fp);
[PV1] 182:   gotoxy(1,3);
[YR3] 183:   print("Streckung: ",215);
[852] 184:   getstr(name);
[SX3] 185:   o->text.big=atoi(name);
[263] 186:   if(o->type==SPHERE)
[P70] 187:   {
[VZ1] 188:     gotoxy(13,3);
[SL3] 189:     print(" ",215);
[PV1] 190:     gotoxy(1,3);
[LR6] 191:     print("Um die Kugel Vertikal Horizontal\
[2R1] 192: (V/H): ",215);
[852] 193:     getstr(name);
[964] 194:     if(*name=='V'||*name=='v')
[4E2] 195:       o->text.typ=1;
[ZR0] 196:     else
[6E2] 197:       o->text.typ=2;
[U70] 198:   }
[UL1] 199:   return(1);
[U70] 200: }
[000] 201:
[W60] 202: /*
[BH7] 203: clssc(): löscht gesamten Graphikbildschirm
[170] 204: */
[000] 205:
[5Y1] 206: void clssc()
[P70] 207: {
[ZD1] 208:   int x,y;
[CU2] 209:   for(x=1;x<=80;x++)
[NU2] 210:     for(y=1;y<=14;y++)
[P70] 211:     {
[F02] 212:       gotoxy(x,y);
[232] 213:       print(" ",215);
[U70] 214:     }
[U70] 215: }
[000] 216:
[W60] 217: /*
[0D6] 218: getcol(c): holt Farbwerte von Benutzer
[170] 219: */
[000] 220:
[AA2] 221: void getcol(c)
[RA1] 222: color *c;
[P70] 223: {
[E32] 224:   char str[20];
[312] 225:   double help;
[VZ1] 226:   gotoxy(13,3);
[Z23] 227:   print(" ",215);
[PV1] 228:   gotoxy(1,3);
[Y94] 229:   print("Rotanteil (0.0-1.0): ",215);
[E02] 230:   getstr(str);
[AL2] 231:   help=atof(str);
[526] 232:   if(help>=0.0&&help<=1.0&&strlen(str)>0)
[2F2] 233:     c->r=atof(str);
[VZ1] 234:   gotoxy(13,3);
[Z23] 235:   print(" ",215);
[PV1] 236:   gotoxy(1,3);
[T55] 237:   print("Grünanteil (0.0-1.0): ",215);
[E02] 238:   getstr(str);
[AL2] 239:   help=atof(str);
[526] 240:   if(help>=0.0&&help<=1.0&&strlen(str)>0)
[RE2] 241:     c->g=atof(str);
[VZ1] 242:   gotoxy(13,3);
[E04] 243:   print(" ",215);
[PV1] 244:   gotoxy(1,3);
[2X4] 245:   print("Blauanteil (0.0-1.0): ",215);
[E02] 246:   getstr(str);
[AL2] 247:   help=atof(str);
[526] 248:   if(help>=0.0&&help<=1.0&&strlen(str)>0)
[KE2] 249:     c->b=atof(str);
[U70] 250: }
[000] 251:
[W60] 252: /*
[PK6] 253: edobj(w): Benutzereditor für Objekte
[170] 254: */
[000] 255:
[U02] 256: void edobj(w)
[CC1] 257: world *w;
[P70] 258: {
[H32] 259:   char str[30];
[8S1] 260:   word nr,i;
[CK1] 261:   object *o;
[312] 262:   double help;
[671] 263:   clssc();
[GV1] 264:   gotoxy(1,1);
[YC4] 265:   print("Nr des Objekts: ",215);
[E02] 266:   getstr(str);
[H92] 267:   nr=atoi(str);
[6X3] 268:   if(nr>countobj||nr<=0)
[KC1] 269:     return;
[R32] 270:   o=w->object;
```

Listing 8. Das Hauptprogramm »ray.c« verbindet CAD-, Textur- und Rechenprogramme.


```

[VL2] 271: for(i=1;i<nr;i++)
[9R1] 272: o=o->next;
[263] 273: if(o->type==SPHERE)
[P70] 274: {
[YT4] 275:     print(" Kugel mit Radius ",215);
[2C6] 276:     printnum((int)o->form.sphere.radius);
[U70] 277: }
[ZR0] 278: else
[P70] 279: {
[K44] 280:     print(" Flaeche mit ",215);
[KW6] 281:     printnum((int)o->form.polygon.num_vec);
[1W2] 282:     print(" Pktn",215);
[U70] 283: }
[LV1] 284: gotoxy(1,2);
[B16] 285: print("Texture oder einfarbig (t/e): ",
[AL0] 286:     215);
[E02] 287: getstr(str);
[004] 288: if(*str=='t'||*str=='T')
[TP2] 289:     loadtexture(o);
[ZR0] 290: else
[P70] 291: {
[3Z1] 292:     o->text.x=0;
[5Z1] 293:     o->text.y=0;
[CB2] 294:     o->text.map=0;
[R23] 295:     getcol(&(o->color));
[U70] 296: }
[TV1] 297: gotoxy(1,4);
[L45] 298: print("Materialdefinition ",215);
[WW1] 299: gotoxy(1,5);
[KW5] 300: print("Undurchsichtigkeit (0.0-1.0): ",
[AL0] 301:     215);
[E02] 302: getstr(str);
[AL2] 303: help=atoi(str);
[526] 304: if(help>0.0&&help<=1.0&&strlen(str)>0)
[5V2] 305:     o->material=help;
[ZV1] 306: gotoxy(1,6);
[AX4] 307: print("Spiegelung (0.0-1.0): ",215);
[E02] 308: getstr(str);
[AL2] 309: help=atoi(str);
[526] 310: if(help>0.0&&help<=1.0&&strlen(str)>0)
[XN2] 311:     o->mirror=help;
[2W1] 312: gotoxy(1,7);
[P65] 313: print("Transparenz (0.0-1.0): ",215);
[E02] 314: getstr(str);
[AL2] 315: help=atoi(str);
[526] 316: if(help>0.0&&help<=1.0&&strlen(str)>0)
[6B2] 317:     o->trans=help;
[5W1] 318: gotoxy(1,8);
[YV5] 319: print("Matte Reflexion (0.0-1.0): ",215);
[E02] 320: getstr(str);
[AL2] 321: help=atoi(str);
[RE3] 322: if(help>0.0&&help<=1.0)
[F53] 323:     o->diff_const=help;
[8W1] 324: gotoxy(1,9);
[C16] 325: print("Spiegelnde Reflexion (0.0-1.0): ",
[AR0] 326:     215);
[E02] 327: getstr(str);
[AL2] 328: help=atoi(str);
[526] 329: if(help>0.0&&help<=1.0&&strlen(str)>0)
[0C3] 330:     o->spec_const=help;
[GZ1] 331: gotoxy(1,10);
[AZ5] 332: print("Konzentr.spie.Refl.(0.0-100.0)\
[KZ0] 333:     ",215);
[E02] 334: getstr(str);
[AL2] 335: help=atoi(str);
[ZZ5] 336: if(help>0.0&&help<=100.0&&strlen(str)>0)
[1C3] 337:     o->high_const=help;
[MZ1] 338: gotoxy(1,11);
[CM4] 339: print("Velozitaet (>0.0): ",215);
[E02] 340: getstr(str);
[AL2] 341: help=atoi(str);
[BS1] 342: if(help>0.0)
[EL1] 343:     o->N=help;
[ZR0] 344: else
[P61] 345:     o->N=1.0;
[671] 346:     clssc();
[U70] 347: }
[000] 348:
[W60] 349: /*
[PD6] 350: edlamp(w):Benutzereditor für Lampen
[170] 351: */
[000] 352:
[6B2] 353: void edlamp(w)
[CC1] 354: world *w;
[P70] 355: {
[H32] 356:     char str[30];
[8S1] 357:     word nr,i;
[881] 358:     lamp *l;
[312] 359:     double help;
[671] 360:     clssc();
[GV1] 361:     gotoxy(1,1);
[C04] 362:     print("Nr der Lampe: ",215);
[E02] 363:     getstr(str);
[H92] 364:     nr=atoi(str);
[M34] 365:     if(nr>countlamp||nr<=0)
[KC1] 366:         return;
[UP1] 367:     l=w->lamp;
[VL2] 368:     for(i=1;i<nr;i++)

```

```

[RP1] 369:     l=l->next;
[LV1] 370:     gotoxy(1,2);
[N26] 371:     print("Intensitaet wie weit >0.0: ",215);
[E02] 372:     getstr(str);
[AL2] 373:     help=atoi(str);
[KA4] 374:     if(help>0.0&&strlen(str)>0)
[GM2] 375:         l->intens=help;
[E23] 376:     getcol(&(l->color));
[671] 377:     clssc();
[U70] 378: }
[000] 379:
[W60] 380: /*
[KR6] 381: edobs(w):Benutzereditor für observer
[170] 382: */
[VW1] 383: int auf =0;
[000] 384:
[N12] 385: void edobs(w)
[CC1] 386: world *w;
[P70] 387: {
[H32] 388:     char str[30];
[8S1] 389:     word nr,i;
[N44] 390:     observer *o=w->observer;
[1K1] 391:     int help;
[GD1] 392:     double h;
[671] 393:     clssc();
[GV1] 394:     gotoxy(1,1);
[UF5] 395:     print("Aufloesung (320/640/800/1024):",
[AL0] 396:         215);
[E02] 397:     getstr(str);
[GV1] 398:     gotoxy(1,1);
[8W3] 399:     print("
[400] 400:         ",215);
[AL2] 401:     help=atoi(str);
[BC4] 402:     if(help>320&&strlen(str)>0)
[9Z0] 403:         auf=1;
[RC4] 404:     if(help>640&&strlen(str)>0)
[BZ0] 405:         auf=2;
[FC4] 406:     if(help>800&&strlen(str)>0)
[DZ0] 407:         auf=3;
[GV1] 408:     gotoxy(1,1);
[ND5] 409:     print("Bildbreite in Pixel: ",215);
[E02] 410:     getstr(str);
[GL2] 411:     help=atoi(str);
[EX3] 412:     if(help>0 &&strlen(str)>0)
[XT2] 413:         o->p_width=help;
[ZR0] 414:     else
[LA2] 415:         o->p_width=60;
[LV1] 416:     gotoxy(1,2);
[R75] 417:     print("Bildhoehe in Pixel: ",215);
[E02] 418:     getstr(str);
[GL2] 419:     help=atoi(str);
[EX3] 420:     if(help>0 &&strlen(str)>0)
[8X2] 421:         o->p_height=help;
[ZR0] 422:     else
[RH2] 423:         o->p_height=40;
[4B4] 424:     o->dx=(int)(o->p_width*1.14);
[ES3] 425:     o->dy=(int)(o->p_width);
[LV1] 426:     gotoxy(1,2);
[MC5] 427:     print("Hintergrundfarbe
[AL0] 428:         ",215);
[K93] 429:     getcol(&background);
[9S4] 430:     background.r =background.r/2;
[XP4] 431:     background.g =background.g/2;
[9P4] 432:     background.b =background.b/2;
[LV1] 433:     gotoxy(1,2);
[D85] 434:     print("Grundhelligkeit
[AL0] 435:         ",215);
[NU2] 436:     getcol(&ambient);
[PV1] 437:     gotoxy(1,3);
[G56] 438:     print("Stärke der Grundhelligkeit \
[VX1] 439:     (0.0-1.0):",215);
[E02] 440:     getstr(str);
[ZV1] 441:     h=atoi(str);
[TR4] 442:     if(h>0&&h<=1.0&&strlen(str)>0)
[FZ0] 443:         amb=h;
[ZR0] 444:     else
[L41] 445:         amb=0.0;
[TV1] 446:     gotoxy(1,4);
[AH5] 447:     print("Berechnungstiefe >=0: ",215);
[E02] 448:     getstr(str);
[GL2] 449:     help=atoi(str);
[SZ3] 450:     if(help>0&&strlen(str)>0)
[PG2] 451:         maxdepth=help;
[ZR0] 452:     else
[V02] 453:         maxdepth=6;
[671] 454:     clssc();
[U70] 455: }
[000] 456:
[W60] 457: /*
[0F6] 458: save(x,y,w) : sichert die Daten der Welt
[170] 459: */
[000] 460:
[MK2] 461: int save(x,y,w)
[ZD1] 462: int x,y;
[CC1] 463: world *w;
[P70] 464: {
[H92] 465:     char name[30];
[F02] 466:     gotoxy(x,y);

```



```
[H54] 467: print("Name:
[KW0] 468: ",215);
[E62] 469: gotoxy(x+5,y);
[852] 470: getstr(name);
[S43] 471: saveworld(w,name);
[U70] 472: }
[000] 473:
[UW0] 474: main()
[P70] 475: {
[LN3] 476: word x,y,ready=0,help;
[NL4] 477: word vidmod = setmod(GR320);
[0P3] 478: char name[30],str[30];
[923] 479: schalter *sch=&ld1;
[000] 480:
[WH1] 481: w1.lamp=0;
[VU1] 482: w1.object=0;
[W92] 483: w1.observer=0;
[000] 484:
[HA1] 485: delay(0);
[9X1] 486: init_pal();
[P72] 487: showsch(sch);
[AY3] 488: mouse_def(pointer,17,6);
[BA2] 489: mouse_show();
[F62] 490: mouse_key(0);
[000] 491:
[YV1] 492: gotoxy(3,5);
[B16] 493: print(" Welcome to the world of",35);
[4W1] 494: gotoxy(3,7);
[8M4] 495: print(" RAY TRACING",30);
[000] 496:
[AW1] 497: gotoxy(3,9);
[795] 498: print(" (c) 1992 Widuch GmbH & Co. \
[R61] 499: KG ",35);
[000] 500:
[VU1] 501: w1.object=0;
[WH1] 502: w1.lamp=0;
[W92] 503: w1.observer=0;
[000] 504:
[LV0] 505: do{
[CF3] 506: while(!mouse_press())
[P70] 507: {
[ZT2] 508: mouse_pos(&x,&y);
[081] 509: x=x>>1;
[U70] 510: }
[M62] 511: mouse_hide();
[4R4] 512: switch(testsch(x,y,&ld1,&sch))
[P70] 513: {
[751] 514: case 10:
[391] 515: ready=1;
[P01] 516: break;
[N41] 517: case 1:
[4W0] 518: cad();
[V23] 519: w1.lamp=firstlamp;
[D14] 520: w1.object=first->first;
[WH2] 521: if(!w1.observer)
[P70] 522: {
[D74] 523: w1.observer=malloc(sizeof
[LV1] 524: (observer));
[8N4] 525: memset(w1.observer,0,sizeof
[LV1] 526: (observer));
[U70] 527: }
[ZD3] 528: mouse_area(0,0,640,200);
[A02] 529: textattr(0);
[0E1] 530: clrscr();
[W92] 531: showsch(&ld1);
[AY3] 532: mouse_def(pointer,17,6);
[BA2] 533: mouse_show();
[F62] 534: mouse_key(0);
[R26] 535: countlamp=count((void **)firstlamp);
[UA6] 536: countobj=count((void **)first->first);
[802] 537: gotoxy(1,16);
[HE3] 538: print("Objekte: ",215);
[PF3] 539: printnum(countobj);
[4T3] 540: print(" - Lampen: ",215);
[RK3] 541: printnum(countlamp);
[P01] 542: break;
[S41] 543: case 2:
[802] 544: gotoxy(1,16);
[944] 545: print("Name:
[KW0] 546: ",215);
[D02] 547: gotoxy(6,16);
[852] 548: getstr(name);
[802] 549: gotoxy(1,16);
[WF3] 550: print(" ",215);
[GV1] 551: gotoxy(1,1);
[895] 552: print("Aufloesung (320/640/800/1024)\
[WW0] 553: :",215);
[E02] 554: getstr(str);
[AL2] 555: help=atoi(str);
[7Z0] 556: auf=0;
[BC4] 557: if(help>320&&strlen(str)>0)
[9Z0] 558: auf=1;
[RC4] 559: if(help>640&&strlen(str)>0)
[BZ0] 560: auf=2;
[FC4] 561: if(help>800&&strlen(str)>0)
[DZ0] 562: auf=3;
[000] 563:
[A02] 564: textattr(0);
```

```
[0E1] 565: clrscr();
[000] 566:
[2W1] 567: switch(auf)
[P70] 568: {
[N41] 569: case 1:
[R34] 570: /*****
[E96] 571: /* Hier die Super-VGA-Modi eintragen: */
[R34] 572: /*****
[922] 573: setmod(0x2f);
[P01] 574: break;
[S41] 575: case 2:
[MZ1] 576: setmod(0x30);
[P01] 577: break;
[V41] 578: case 3:
[M02] 579: setmod(0x38);
[P01] 580: break;
[U70] 581: }
[531] 582: if(auf)
[9X1] 583: init_pal();
[RA2] 584: loadpic(name);
[BA2] 585: mouse_show();
[5X4] 586: while(!mouse_press()&&!kbhit());
[531] 587: if(auf)
[P70] 588: {
[V42] 589: setmod(GR320);
[9X1] 590: init_pal();
[U70] 591: }
[AV2] 592: if(kbhit())getch();
[M62] 593: mouse_hide();
[A02] 594: textattr(0);
[0E1] 595: clrscr();
[W92] 596: showsch(&ld1);
[AY3] 597: mouse_def(pointer,17,6);
[BA2] 598: mouse_show();
[F62] 599: mouse_key(0);
[802] 600: gotoxy(1,16);
[HE3] 601: print("Objekte: ",215);
[PF3] 602: printnum(countobj);
[4T3] 603: print(" - Lampen: ",215);
[RK3] 604: printnum(countlamp);
[P01] 605: break;
[V41] 606: case 3:
[SE2] 607: maketexture();
[A02] 608: textattr(0);
[0E1] 609: clrscr();
[W92] 610: showsch(&ld1);
[AY3] 611: mouse_def(pointer,17,6);
[BA2] 612: mouse_show();
[F62] 613: mouse_key(0);
[802] 614: gotoxy(1,16);
[HE3] 615: print("Objekte: ",215);
[PF3] 616: printnum(countobj);
[4T3] 617: print(" - Lampen: ",215);
[RK3] 618: printnum(countlamp);
[P01] 619: break;
[DM3] 620: case 4: /* Edit Object */
[SM1] 621: edobj(&w1);
[P01] 622: break;
[EC3] 623: case 5: /* Edit Lamp */
[RT1] 624: edlamp(&w1);
[P01] 625: break;
[L54] 626: case 6: /* Edit Observer */
[AN1] 627: edobs(&w1);
[P01] 628: break;
[WG3] 629: case 7: /* Save World */
[K32] 630: save(1,16,&w1);
[P01] 631: break;
[HF3] 632: case 8: /* load World */
[802] 633: gotoxy(1,16);
[KK3] 634: print("
[KW0] 635: ",215);
[E32] 636: loadall(1,16);
[V23] 637: w1.lamp=firstlamp;
[D14] 638: w1.object=first->first;
[R26] 639: countlamp=count((void **)firstlamp);
[UA6] 640: countobj=count((void **)first->first);
[802] 641: gotoxy(1,16);
[HE3] 642: print("Objekte: ",215);
[PF3] 643: printnum(countobj);
[4T3] 644: print(" - Lampen: ",215);
[RK3] 645: printnum(countlamp);
[P01] 646: break;
[CF3] 647: case 9: /* calc World */
[A02] 648: textattr(0);
[0E1] 649: clrscr();
[GV1] 650: gotoxy(1,1);
[XC5] 651: print("Zum Speichern des Bildes \
[AY1] 652: rechte ",215);
[LV1] 653: gotoxy(1,2);
[RT5] 654: print("Maustaste druecken; sonst \
[8N1] 655: linke",215);
[PV1] 656: gotoxy(1,3);
[LF3] 657: print("Name: ",215);
[VV1] 658: gotoxy(6,3);
[852] 659: getstr(name);
```

Listing 8. Das Hauptprogramm »ray.c« verbindet CAD-, Textur- und Rechenprogramme.


```
[PV1] 660: gotoxy(1,3);
[WF3] 661: print(" ",215);
[A02] 662: textattr(0);
[0E1] 663: clrscr();
[R34] 664: /******
[E96] 665: /* Hier die Super-VGA-Modi eintragen: */
[R34] 666: /******
[2W1] 667: switch(auf)
[P70] 668: {
[N41] 669: case 1:
[922] 670: setmod(0x2f);
[P01] 671: break;
[S41] 672: case 2:
[MZ1] 673: setmod(0x30);
[P01] 674: break;
[V41] 675: case 3:
[M02] 676: setmod(0x38);
[P01] 677: break;
[U70] 678: }
[531] 679: if(auf)
[9X1] 680: init_pal();
[CN2] 681: show_world(&wl);
[BA2] 682: mouse_show();
[5X4] 683: while(!mouse_press() && !kbhit());
[AV2] 684: if(kbhit()) getch();
[M62] 685: mouse_hide();
[FF3] 686: while(mouse_press());
[VH2] 687: if(mouse_key(1))
[W04] 688: savepic(0,0,wl.observer->
[9P3] 689: p_width,wl.observer->
[1M2] 690: p_height,name);
[531] 691: if(auf)
[P70] 692: {
[V42] 693: setmod(0x32);
[9X1] 694: init_pal();
[U70] 695: }
[A02] 696: textattr(0);
[0E1] 697: clrscr();
[W92] 698: showsch(&ld1);
[AY3] 699: mouse_def(pointer,17,6);
[BA2] 700: mouse_show();
[F62] 701: mouse_key(0);
[802] 702: gotoxy(1,16);
[HE3] 703: print("Objekte: ",215);
[PF3] 704: printnum(countobj);
[4T3] 705: print(" - Lampen: ",215);
[RK3] 706: printnum(countlamp);
[P01] 707: break;
[8E1] 708: default:
[0S1] 709: sound(440);
[TA1] 710: delay(2);
[LR1] 711: nosound();
[P01] 712: break;
[U70] 713: }
[U55] 714: drawtangle(sch->left,sch->top,sch->
[UG5] 715: right,sch->bottom, sch->framecol);
[BA2] 716: mouse_show();
[4L2] 717: }while(!ready);
[M62] 718: mouse_hide();
[2K2] 719: setmod(vidmod);
[U70] 720: }
[NZ0] 721: #endif
```

Listing 8. Das Hauptprogramm »ray.c« verbindet CAD-, Textur- und Rechenprogramme.

```
[W60] 1: /*
[SP2] 2: Programm: save.c
[UF5] 3: Funktion: enthält Routinen zur
[MR3] 4: Speicherung von Daten
[ZT2] 5: auf Festplatten
[9C5] 6: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 7: Autor: Matthias Oberdorfer
[V04] 8: (c) 1992 Widuch GmbH & Co. KG
[170] 9: */
[000] 10:
[W02] 11: #ifndef SAVE
[212] 12: #define SAVE
[000] 13:
[3F2] 14: #include "vid.c"
[1R2] 15: #include "graph.c"
[043] 16: #include <stdlib.h>
[8W2] 17: #include <stdio.h>
[000] 18:
[566] 19: int savepic(word x1,word y1,word x2,
[T43] 20: word y2,char *name);
[VW3] 21: int loadpic(char *name);
[000] 22:
[W60] 23: /*
[CU5] 24: savepic(x1,y1,x2,y2,name): Speichert
[AV4] 25: Bildausschnitt unter name ab;
[DP5] 26: width, height: Bildbreite, Bildhoehe
[BZ2] 27: 1 wenn erfolgreich
[170] 28: */
```

```
[000] 29:
[6R4] 30: int savepic(x1,y1,x2,y2,name)
[0N2] 31: word x1,y1,x2,y2;
[0U1] 32: char *name;
[P70] 33: {
[SB1] 34: FILE *fp;
[T81] 35: byte c;
[2A1] 36: int i, j;
[2H5] 37: word width=x2-x1+1,height=y2-y1+1;
[000] 38:
[8W2] 39: /* Datei öffnen */
[7Z2] 40: fp=fopen(name,"wb");
[521] 41: if(!fp)
[TL1] 42: return(0);
[SK5] 43: fwrite(&width,sizeof(width),1,fp);
[0T5] 44: fwrite(&height,sizeof(height),1,fp);
[000] 45:
[R13] 46: for(i=x1;i<=x2;i++)
[323] 47: for(j=y1;j<=y2;j++)
[P70] 48: {
[0X2] 49: c=get_pixel(i,j);
[X14] 50: fwrite(&c,sizeof(c),1,fp);
[U70] 51: }
[PT1] 52: fclose(fp);
[UL1] 53: return(1);
[U70] 54: }
[000] 55:
[W60] 56: /*
[826] 57: loadpic(name): lädt Bild mit name ein
[WS4] 58: liefert 1 falls erfolgreich
[170] 59: */
[000] 60:
[KW2] 61: int loadpic(name)
[0U1] 62: char *name;
[P70] 63: {
[SB1] 64: FILE *fp;
[2A1] 65: int i, j;
[T81] 66: byte c;
[E33] 67: word width,height;
[000] 68:
[8W2] 69: /* Datei öffnen */
[2Z2] 70: fp=fopen(name,"rb");
[521] 71: if(!fp)
[TL1] 72: return(0);
[495] 73: fread(&width,sizeof(width),1,fp);
[KM5] 74: fread(&height,sizeof(height),1,fp);
[000] 75:
[Z63] 76: for(i=0;i<width;i++)
[M93] 77: for(j=0;j<height;j++)
[P70] 78: {
[3T3] 79: fread(&c,sizeof(c),1,fp);
[HW2] 80: put_pixel(i,j,c);
[U70] 81: }
[PT1] 82: fclose(fp);
[UL1] 83: return(1);
[U70] 84: }
[000] 85:
[NZ0] 86: #endif
```

Listing 9. »save.c« stellt Routinen zur Datenspeicherung bereit.

```
[W60] 1: /*
[PD3] 2: Programm: schalter.c
[225] 3: Funktion: enthält Mausfunktionen
[9C5] 4: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 5: Autor: Matthias Oberdorfer
[V04] 6: (c) 1992 Widuch GmbH & Co. KG
[170] 7: */
[000] 8:
[0R2] 9: #ifndef SCHALTER
[6R2] 10: #define SCHALTER
[000] 11:
[3F2] 12: #include "vid.c"
[AL2] 13: #include "maus.c"
[1R2] 14: #include "graph.c"
[043] 15: #include <stdlib.h>
[8W2] 16: #include <stdio.h>
[000] 17:
[864] 18: typedef struct schalter
[P70] 19: {
[FM6] 20: struct schalter *next; /*next Schalter*/
[KC5] 21: word left,top; /* wie Groß */
[FT4] 22: word right,bottom; /* und wo */
[X84] 23: word framecol,col,ilucol;
[WN5] 24: word backcol; /* bleibt erleuchtet
[X11] 25: bei 2 */
[Z42] 26: /* blinkt bei 1
[3X3] 27: tut sich nichts bei 0 */
[SH1] 28: char *msg;
[C12] 29: } schalter;
[000] 30:
[H07] 31: word testsch(word x,word y,schalter *sch,
[ND2] 32: schalter **sha);
```



```
[S05] 33: void print(char *str, word co);
[D23] 34: void printnum(word nr);
[DX3] 35: void getstr(char *str);
[SL4] 36: void showsch(schalter *sch);
[000] 37:
[W60] 38: /*
[Y46] 39: testsch(x,y,sch,sha): überprüft ob
[G86] 40: an Stelle x,y ein Schalter ist und gibt
[0L5] 41: ihn gegebenenfalls in sha zurück
[170] 42: */
[000] 43:
[964] 44: word testsch(x,y,sch,sha)
[EF1] 45: word x,y;
[U53] 46: schalter *sch,**sha;
[P70] 47: {
[VP2] 48: schalter *s=sch;
[M91] 49: int i=1;
[DB1] 50: while(s)
[P70] 51: {
[TM4] 52: if(x>=s->left && x<=s->right &&
[H24] 53: y>=s->top && y<=s->bottom)
[P70] 54: {
[942] 55: if(s->backcol)
[P70] 56: {
[M62] 57: mouse_hide();
[AR5] 58: drawtangle(s->left,s->top,s->right
[CS3] 59: ,s->bottom, s->ilucol);
[WH2] 60: if(s->backcol==1)
[P70] 61: {
[DN1] 62: delay(100);
[884] 63: drawtangle(s->left,s->top,
[H65] 64: s->right,s->bottom,s->framecol);
[U70] 65: }
[BA2] 66: mouse_show();
[U70] 67: }
[DB1] 68: *sha=s;
[KM1] 69: return(i);
[U70] 70: }
[UF0] 71: i++;
[1S1] 72: s=s->next;
[U70] 73: }
[TL1] 74: return(0);
[U70] 75: }
[000] 76:
[W60] 77: /*
[DR6] 78: print(str, co):gibt String im Grafikmodus
[B53] 79: in der Farbe co aus.
[170] 80: */
[000] 81:
[RZ2] 82: void print(str,co)
[7L1] 83: char *str;
[7D1] 84: word co;
[P70] 85: {
[R73] 86: struct REGPACK reg;
[G91] 87: int i=0;
[7H2] 88: while(*(str + i))
[P70] 89: {
[7C4] 90: reg.r_ax = 0x0900!*(str+i);
[443] 91: reg.r_bx = co&0x00ff;
[002] 92: reg.r_cx = 1;
[VC2] 93: intr(0x10,&reg);
[GE4] 94: gotoxy(wherex()+1,wherey());
[UF0] 95: i++;
[U70] 96: }
[U70] 97: }
[000] 98:
[W60] 99: /*
[XB6] 100: getstr(str): holt String vom Benutzer
[SR2] 101: im Graphikmodus
[170] 102: */
[000] 103:
[3X2] 104: void getstr(str)
[7L1] 105: char *str;
[P70] 106: {
[ZW1] 107: char ch[2];
[7B1] 108: ch[1]=0;
[1V0] 109: do{
[8C2] 110: ch[0]=getch();
[M32] 111: switch(ch[0])
[P70] 112: {
[DK1] 113: case '\r':
[9K1] 114: case '\n':
[P01] 115: break;
[A51] 116: case 8:
[211] 117: str--;
[KE4] 118: gotoxy(wherex()-1,wherey());
[232] 119: print(" ",215);
[KE4] 120: gotoxy(wherex()-1,wherey());
[P01] 121: break;
[8E1] 122: default:
[632] 123: print(ch,215);
[ZC2] 124: *str ++ = ch[0];
[P01] 125: break;
[U70] 126: }
[U70] 127: }
[D43] 128: while(ch[0]!='\r');
[C81] 129: *str = 0;
```

```
[U70] 130: }
[000] 131:
[W60] 132: /*
[V46] 133: printnum(i):gibt Zahl im Grafikmodus
[VR0] 134: aus.
[170] 135: */
[6U2] 136: void printnum(i)
[E91] 137: word i;
[P70] 138: {
[B32] 139: char str[10];
[ZB2] 140: itoa(i,str,10);
[UD2] 141: print(str,215);
[U70] 142: }
[000] 143:
[W60] 144: /*
[M46] 145: showsch(sch) : Stellt Schalter auf dem
[VT5] 146: Bildschirm nach structdefinitionen
[TM0] 147: auf
[170] 148: */
[000] 149:
[LW2] 150: void showsch(sch)
[SA2] 151: schalter *sch;
[P70] 152: {
[VP2] 153: schalter *s=sch;
[DB1] 154: while(s)
[P70] 155: {
[UW5] 156: drawtangle(s->left,s->top,s->right,
[GT3] 157: s->bottom, s->framecol);
[AV4] 158: fill(s->left+1,s->top+1,s->col);
[VF1] 159: if(s->msg)
[P70] 160: {
[KV4] 161: gotoxy(s->left/8+2,s->top/8+2);
[1W2] 162: print(s->msg,215);
[U70] 163: }
[1S1] 164: s=s->next;
[U70] 165: }
[U70] 166: }
[000] 167:
[NZ0] 168: #endif
```

Listing 10. »schalter.c« enthält die Bedienungselemente für die Maus.

```
[W60] 1: /*
[WE3] 2: Programm: texture.c
[YM6] 3: Funktion: Editor für Strukturen auf
[AF4] 4: dreidimensionalen Körpern
[8M4] 5: Malprogramm für Texturen
[9C5] 6: Sprache: Turbo C 2.0/Turbo C++ 1.0
[KR4] 7: Autor: Matthias Oberdorfer
[V04] 8: (c) 1992 Widuch GmbH & Co. KG
[170] 9: */
[000] 10:
[NR2] 11: #ifndef TEXTURE
[VR2] 12: #define TEXTURE
[000] 13:
[3F2] 14: #include"vid.c"
[AL2] 15: #include"maus.c"
[1R2] 16: #include"graph.c"
[GB3] 17: #include"schalter.c"
[1L2] 18: #include"save.c"
[043] 19: #include<stdlib.h>
[8W2] 20: #include<stdio.h>
[000] 21:
[LK2] 22: void cls(void);
[334] 23: void maketexture(void);
[000] 24:
[V13] 25: schalter schall = {
[470] 26: 0,
[5Y1] 27: 242,142,300,154,
[B11] 28: 56,0,215,
[870] 29: 2,
[A31] 30: "TEXT",
[BD0] 31: };
[000] 32:
[R13] 33: schalter schal0 = {
[Y41] 34: &schall,
[EY1] 35: 170,142,240,154,
[B11] 36: 56,0,215,
[870] 37: 2,
[E51] 38: "CLEAR",
[BD0] 39: };
[000] 40:
[RT2] 41: schalter scha9 = {
[U41] 42: &schal0,
[W21] 43: 170,157,240,169,
[B11] 44: 56,0,215,
[870] 45: 2,
[U31] 46: "EXIT",
[BD0] 47: };
```

Listing 11. »texture.c« ist ein Editor für Texturen.


```

[000] 48:
[000] 49:
[MT2] 50: schalter scha8={
[P31] 51:  &scha9,
[D02] 52:  138,142,147,186,
[B11] 53:  56,0,215,
[470] 54:  0,
[470] 55:  0,
[BD0] 56:  };
[000] 57:
[CD2] 58: /* Zeichenfeld */
[ZS2] 59: schalter schal={
[L31] 60:  &scha8,
[3E1] 61:  0,0,319,140,
[MN0] 62:  0,0,0,
[470] 63:  0,
[470] 64:  0,
[BD0] 65:  };
[000] 66:
[2T2] 67: schalter scha2={
[Y21] 68:  &scha1,
[VG1] 69:  0,142,64,154,
[B11] 70:  56,0,215,
[870] 71:  2,
[X21] 72:  "DRAW",
[BD0] 73:  };
[000] 74:
[5T2] 75: schalter scha3={
[L31] 76:  &scha2,
[HV1] 77:  66,142,128,154,
[B11] 78:  56,0,215,
[870] 79:  2,
[U11] 80:  "LINE",
[BD0] 81:  };
[000] 82:
[4F0] 83: /* */
[8T2] 84: schalter scha4={
[431] 85:  &scha3,
[FH1] 86:  0,157,64,169,
[B11] 87:  56,0,215,
[870] 88:  2,
[321] 89:  "LOAD",
[BD0] 90:  };
[000] 91:
[BT2] 92: schalter scha5={
[731] 93:  &scha4,
[LW1] 94:  66,157,128,169,
[B11] 95:  56,0,215,
[870] 96:  2,
[R11] 97:  "SAVE",
[BD0] 98:  };
[000] 99:
[ET2] 100: schalter scha6={
[A31] 101:  &scha5,
[AH1] 102:  0,172,64,184,
[B11] 103:  56,0,215,
[870] 104:  2,
[M11] 105:  "FILL",
[BD0] 106:  };
[000] 107:
[HT2] 108: schalter scha7={
[D31] 109:  &scha6,
[XV1] 110:  66,172,128,184,
[B11] 111:  56,0,215,
[870] 112:  2,
[6V0] 113:  "BOX",
[BD0] 114:  };
[000] 115:
[6C3] 116: #define FIRST  &scha7
[000] 117:
[W60] 118: /*
[T95] 119:  cls(); löscht den Bildschirm des
[4B2] 120:  Malprogrammes
[170] 121:  */
[000] 122:
[SK1] 123: void cls()
[P70] 124: {
[EF1] 125:  word x,y;
[US2] 126:  for(x=0;x<320;x++)
[YS2] 127:  for(y=0;y<141;y++)
[ZW2] 128:  put_pixel(x,y,0);
[U70] 129: }
[000] 130:
[W60] 131: /*
[876] 132:  maketexture(void): editor für Muster
[V45] 133:  und dreidimensionalen Körpern
[170] 134:  */
[000] 135:
[833] 136: void maketexture()
[P70] 137: {
[YU1] 138:  word b,g,r;
[RW1] 139:  word x,y,i;
[S32] 140:  char help[2];
[Z23] 141:  word xold,yold, farbe=0;
[DW3] 142:  word ready=0, drawnr=1;
[SH4] 143:  schalter *sch=&scha2,*dummy;
[H92] 144:  char name[30];
[A02] 145:  textattr(0);

```

```

[0E1] 146:  clrscr();
[FW3] 147:  /* Farbpalette zeigen */
[000] 148:
[4A2] 149:  for(b=0;b<6;b++)
[2B2] 150:  for(g=0;g<6;g++)
[4D2] 151:  for(r=0;r<6;r++)
[CK5] 152:  put_pixel(g+140,r+7*b+144,r+g*6+b*36);
[TP2] 153:  showsch(FIRST);
[M62] 154:  mouse_hide();
[GA5] 155:  drawtangle(scha2.left,scha2.top,
[B64] 156:  scha2.right,scha2.bottom,
[C92] 157:  scha2.ilucol);
[MA4] 158:  drawtangle(150,142,160,152,215);
[BV2] 159:  fill(151,143,farbe);
[C42] 160:  gotoxy(32,21);
[CX2] 161:  print("by M.O.",215);
[BA2] 162:  mouse_show();
[000] 163:
[AY3] 164:  mouse_def(pointer,17,6);
[BA2] 165:  mouse_show();
[F62] 166:  mouse_key(0);
[000] 167:
[1V0] 168:  do{
[CF3] 169:  while(!mouse_press())
[P70] 170:  {
[ZT2] 171:  mouse_pos(&x,&y);
[081] 172:  x=x>>1;
[C42] 173:  gotoxy(21,24);
[HT2] 174:  print("POS: ",215);
[K32] 175:  printnum(x);
[Z32] 176:  print(" ",215);
[N32] 177:  printnum(y);
[2R2] 178:  print("      ",215);
[U70] 179:  }
[000] 180:
[XD4] 181:  testsch(x,y,FIRST,&dummy);
[000] 182:
[Y44] 183:  if(dummy!=&scha1&&dummy!=&scha8)
[P70] 184:  {
[M62] 185:  mouse_hide();
[UU5] 186:  drawtangle(sch->left,sch->top,sch->
[UG5] 187:  right,sch->bottom, sch->framecol);
[BA2] 188:  mouse_show();
[U70] 189:  }
[M62] 190:  mouse_hide();
[U11] 191:  dummy=sch;
[U35] 192:  switch(testsch(x,y,FIRST,&sch))
[P70] 193:  {
[N41] 194:  case 1:
[9G1] 195:  drawnr=4;
[P01] 196:  break;
[S41] 197:  case 2:
[8G1] 198:  drawnr=3;
[P01] 199:  break;
[Y41] 200:  case 4:
[B42] 201:  gotoxy(21,23);
[TM3] 202:  print("Dateiname: ",215);
[852] 203:  getstr(name);
[B42] 204:  gotoxy(21,23);
[WF3] 205:  print("      ",215);
[1Z0] 206:  cls();
[RA2] 207:  loadpic(name);
[P01] 208:  break;
[V41] 209:  case 3:
[B42] 210:  gotoxy(21,23);
[TM3] 211:  print("Dateiname: ",215);
[852] 212:  getstr(name);
[B42] 213:  gotoxy(21,23);
[P45] 214:  print("links oben mit Maus ",215);
[FF3] 215:  while(mouse_press());
[F62] 216:  mouse_key(0);
[BA2] 217:  mouse_show();
[XM3] 218:  while(!mouse_press());
[B42] 219:  gotoxy(21,23);
[LX4] 220:  print("Maus rechts unten",215);
[ZT2] 221:  mouse_pos(&x,&y);
[081] 222:  x=x>>1;
[AG2] 223:  xold=x;yold=y;
[FF3] 224:  while(mouse_press());
[B42] 225:  gotoxy(21,23);
[EC3] 226:  print("      ",215);
[M62] 227:  mouse_hide();
[ZT2] 228:  mouse_pos(&x,&y);
[081] 229:  x=x>>1;
[BL1] 230:  if(x>xold)
[3M3] 231:  {i=x;x=xold;xold=i;}
[FL1] 232:  if(y>yold)
[AM3] 233:  {i=y;y=yold;yold=i;}
[2U4] 234:  savepic(x,y,xold,yold,name);
[8U4] 235:  drawtangle(x,y,xold,yold,215);
[P01] 236:  break;
[151] 237:  case 5:
[7G1] 238:  drawnr=2;
[P01] 239:  break;
[451] 240:  case 6:
[6G1] 241:  drawnr=1;
[P01] 242:  break;
[751] 243:  case 7:

```



```
[ZT2] 244: mouse_pos(&x,&y);
[Ø81] 245: x=x>>1;
[AG2] 246: xold=x;yold=y;
[NC2] 247: switch(drawnr)
[P7Ø] 248: {
[N41] 249: case 1:
[GP2] 250: if(!mouse_key(Ø))
[XA4] 251: /* Noch nicht losgelassen */
[TB3] 252: while(mouse_press())
[P7Ø] 253: {
[AG2] 254: xold=x;yold=y;
[ZT2] 255: mouse_pos(&x,&y);
[Ø81] 256: x=x>>1;
[Z85] 257: if(x>=sch->left && x<=sch->right &&
[CR4] 258: y>=sch->top && y<=sch->bottom)
[NZ4] 259: drawline(xold,yold,x,y,farbe);
[U7Ø] 260: }
[PØ1] 261: break;
[S41] 262: case 2:
[BA2] 263: mouse_show();
[FF3] 264: while(mouse_press());
[M62] 265: mouse_hide();
[ZT2] 266: mouse_pos(&x,&y);
[Ø81] 267: x=x>>1;
[LZ4] 268: drawline(x,y,xold,yold,farbe);
[PØ1] 269: break;
[V41] 270: case 3:
[PF2] 271: fill(x,y,farbe);
[PØ1] 272: break;
[Y41] 273: case 4:
[BA2] 274: mouse_show();
[FF3] 275: while(mouse_press());
[M62] 276: mouse_hide();
[ZT2] 277: mouse_pos(&x,&y);
[Ø81] 278: x=x>>1;
[G85] 279: drawtangle(x,y,xold,yold,farbe);
[PØ1] 280: break;
[151] 281: case 5:
[AR1] 282: help[1]=Ø;
[1VØ] 283: do{
[BA2] 284: mouse_show();
[A82] 285: while(!kbhit())
[P7Ø] 286: {
[MV2] 287: if(mouse_press())
[P7Ø] 288: {
[ZT2] 289: mouse_pos(&x,&y);
[Ø81] 290: x=x>>1;
[FY2] 291: gotoxy(x/8+1,y/8+1);
[U7Ø] 292: }
[U7Ø] 293: }
[DN2] 294: help[Ø]=getch();
[GN2] 295: if(help[Ø]!='\r')
[P7Ø] 296: {
[M62] 297: mouse_hide();
[LY2] 298: print(help,farbe);
[U7Ø] 299: }
[U7Ø] 300: }
[2D3] 301: while(help[Ø]!='\r');
[712] 302: sch=dummy;
[PØ1] 303: break;
[U7Ø] 304: }
[RT1] 305: sch=dummy;
[PØ1] 306: break;
[A51] 307: case 8:
[GP2] 308: if(!mouse_key(Ø))
[XA4] 309: /* Noch nicht losgelassen */
[TB3] 310: while(mouse_press())
[P7Ø] 311: {
[RP3] 312: farbe=get_pixel(x,y);
[6E4] 313: if(get_pixel(15Ø,143)==farbe)
[2U3] 314: drawtangle(15Ø,142,16Ø,152,
[AV2] 315: farbe>=215?214:215);
[BV2] 316: fill(151,143,farbe);
[U7Ø] 317: }
[RT1] 318: sch=dummy;
[PØ1] 319: break;
[D51] 320: case 9:
[391] 321: ready=1;
[PØ1] 322: break;
[751] 323: case 10:
[1ZØ] 324: cls();
[PØ1] 325: break;
[B51] 326: case 11:
[AG1] 327: drawnr=5;
[PØ1] 328: break;
[8E1] 329: default:
[ØS1] 330: sound(44Ø);
[TA1] 331: delay(2);
[LR1] 332: nosound();
[PØ1] 333: break;
[U7Ø] 334: }
[BA2] 335: mouse_show();
[ØØØ] 336: }while(!ready);
[4L2] 337: }
[ØØØ] 338: }
[M62] 339: mouse_hide();
[U7Ø] 340: }
[NZØ] 341: #endif
```

Listing 11. »texture.c« ist ein Editor für Texturen.

```
[W6Ø] 1: /*
[MM2] 2: Programm: vid.c
[YX5] 3: Funktion: enthält Funktionen zur
[K24] 4: Videokartenansteuerung
[9C4] 5: Sprache: Turbo C 2.Ø/Turbo C++ 1.Ø
[KR4] 6: Autor: Matthias Oberdorfer
[VØ4] 7: (c) 1992 Widuch GmbH & Co. KG
[17Ø] 8: */
[ØØØ] 9:
[1Ø] 10: #ifndef VID
[DX1] 11: #define VID
[ØØØ] 12:
[7M2] 13: #include<dos.h>
[4L2] 14: #include"defs.h"
[ØØØ] 15:
[624] 16: word setmod(word mode);
[D76] 17: void put_pixel(word x,word y, word c);
[RE3] 18: void init_pal(void);
[ØØØ] 19:
[W6Ø] 20: /*
[TD1] 21: setmod():
[L76] 22: schaltet in den neuen Videomodus um
[HW5] 23: liefert alten Videomodus zurück.
[17Ø] 24: */
[ØØØ] 25:
[ØØØ] 26:
[XU2] 27: #define GR32Ø ØxØØ13
[ØØØ] 28:
[NX2] 29: word setmod(mode)
[9S1] 30: word mode;
[P7Ø] 31: {
[R73] 32: struct REGPACK reg;
[YH1] 33: word old;
[MD6] 34: reg.r_ax = ØxØfØØ; /*alten MODUS merken*/
[VC2] 35: intr(Øx1Ø,&reg);
[1X2] 36: old=reg.r_ax&Øxf;
[9E6] 37: reg.r_ax = mode; /*neu MODUS festlegen*/
[VC2] 38: intr(Øx1Ø,&reg);
[SV1] 39: return(old);
[U7Ø] 40: }
[ØØØ] 41:
[W6Ø] 42: /*
[W16] 43: void put_pixel(word x,word y, word c)
[B56] 44: setzt Farbwert an einen Grafikpunkt
[17Ø] 45: */
[ØØØ] 46:
[HG3] 47: void put_pixel(x,y,c)
[5W1] 48: word x,y,c;
[P7Ø] 49: {
[R73] 50: struct REGPACK reg;
[XZ2] 51: reg.r_ax=ØxØcØØ[c];
[D92] 52: if(x<64Ø && y<64Ø)
[P7Ø] 53: {
[G22] 54: reg.r_cx=x;
[R22] 55: reg.r_dx=y;
[7K2] 56: intr(Øx1Ø, &reg);
[U7Ø] 57: }
[U7Ø] 58: }
[ØØØ] 59:
[W6Ø] 60: /*
[3K5] 61: unsigned get_pixel(word x,word y)
[GR5] 62: holt Farbwert an einen Grafikpunkt
[17Ø] 63: */
[ØØØ] 64:
[DØ4] 65: unsigned get_pixel(x,y)
[EF1] 66: word x,y;
[P7Ø] 67: {
[R73] 68: struct REGPACK reg;
[7M2] 69: reg.r_ax=ØxØdØØ;
[G22] 70: reg.r_cx=x;
[R22] 71: reg.r_dx=y;
[7K2] 72: intr(Øx1Ø, &reg);
[XW3] 73: return(reg.r_ax&ØxØØff);
[U7Ø] 74: }
[ØØØ] 75:
[W6Ø] 76: /*
[3C3] 77: void init_pal (void)
[A75] 78: initialisiert RGB-Farbpalette
[17Ø] 79: */
[ØØØ] 80:
[RM2] 81: void init_pal()
[P7Ø] 82: {
[HM1] 83: int r,g,b;
[R73] 84: struct REGPACK reg;
[4A2] 85: for(b=Ø;b<6;b++)
[2B2] 86: for(g=Ø;g<6;g++)
[4D2] 87: for(r=Ø;r<6;r++)
[P7Ø] 88: {
[6P2] 89: reg.r_ax = Øx1Ø1Ø;
[463] 90: reg.r_bx = r*g*6+b*36;
[9N5] 91: reg.r_cx = (word)((g*1Ø.5)<<8|(word)
[2C1] 92: ((b)*1Ø.5);
[LG4] 93: reg.r_dx = (word)((r)*1Ø.5)<<8;
[VC2] 94: intr(Øx1Ø,&reg);
[U7Ø] 95: }
[U7Ø] 96: }
[ØØØ] 97:
[NZØ] 98: #endif
```

Listing 12. »vid.c« stellt Funktionen zur Ansteuerung der Grafikkarte bereit.

EGA/VGA-Interna

Durch die direkte Programmierung der Grafikkarten-Hardware steigern Sie die Geschwindigkeit Ihrer grafischen Applikationen erheblich. Dazu benötigen Sie Grundwissen über die Hardware-Organisation Ihrer EGA- beziehungsweise VGA-Karte. Dieser Beitrag zeigt Ihnen einige Details zu Grafikkarten auf.

Die beiden weitverbreiteten Grafikstandards EGA (Enhanced Graphics Adapter) und VGA (Video Graphics Array) verfügen über mehrere Hardware-Teilkomponenten. Hierzu gehören die Controller CRTC (Cathode Ray Tube Controller), TS (Timer Sequencer), GDC (Graphics Data Controller) und ATC (Attribute Controller). Diese Teilkomponenten des Grafikkarten-Chipsatzes halten für Sie eine große Anzahl von Registern bereit, mit denen Sie die Grafikkarten-Hardware manipulieren können. Darüber hinaus existieren einige allgemeine Register, die sich keinem Controller zuweisen lassen. Die VGA-Karte besitzt den weiteren externen Palettenpeicher (External Palette RAM). Im folgenden steht die allgemeine Bezeichnung »EGA-Karte« für beide Grafikkarten-Standards. Alle erweiterten Funktionen der VGA-Karte erhalten einen speziellen Hinweis.

Die EGA-Karte stellt Ihnen mehrere Text- und Grafikmodi zur Verfügung, die Sie Tabelle 1 entnehmen. Alle mit einem Stern »*« gekennzeichneten Modi stellen neue Grafikmodi der VGA-Karte dar. Die Programmierung der Textmodi ist im Gegensatz zur Programmierung der Grafikmodi relativ einfach. Sie bedienen sich der Funktion 0 des BIOS-Interrupts 10hex, um einen der Textmodi zu aktivieren. Hierzu übergeben Sie im AH-Register die Funktionsnummer 0. In das Register »AL« schreiben Sie zusätzlich die Modusnummer des gewünschten Textmodus gemäß Tabelle 1. Die benötigten Videoinformationen schreiben Sie danach entweder direkt in den Bildschirmspeicher oder überlassen diese Tätigkeit wiederum dem Grafikkarten-BIOS.

● **Grafikmodi:** Im folgenden geht es ausschließlich um die Grafikmodi der EGA- beziehungsweise VGA-Karte. Die Menge der Grafikmodi läßt sich in zwei Klassen unterteilen: in die Grafikmodi mit 16 Farben und den VGA-Standard-Modus 13hex mit 256 Farben. Allen Modi ist gemein, daß das Bildschirmspeichersegment an der physikalischen Adresse A000hex liegt. Der Bildschirmspeicher ist in allen Grafikmodi 64 KByte groß und endet mit der Segmentadresse AFFFhex. Durch den unterschiedlichen Speicherbedarf der Grafikmodi realisieren Sie eine oder mehrere Grafikseiten. Tabelle 1 informiert Sie zusätzlich über den Speicherbedarf einer Bildschirmseite in den verschiedenen Bildschirmmodi.

Die am häufigsten verwendeten Grafikmodi stellen die 16 Farbenmodi dar. Hierzu gehören die Grafikmodi 320x200 (0Dhex), 640x200 (0Ehex), 640x350 (10hex) und bei der VGA-Karte 640x480 (12hex). Wenn Sie an Ihrer EGA-Karte einen monochromen Monitor betreiben, steht Ihnen der monochrome Modus 640x350 (0Fhex) zur Verfügung.

● **Bitplanes:** Alle 16 Farbenmodi verwenden das Vier-Plane-Speichermodell. Eine »Plane« (Ebene) ist eine 64 KByte große Speicherseite der Grafikkarte. Gebräuchlich ist hierbei

auch der Ausdruck »Bitplane«, da eine Ebene bitweise organisiert ist (Bild 1). Sie zeigt Ihnen, wie Ihre Grafikkarte die Videoinformationen im Grafikmodus verschlüsselt. Ein Bildpunkt entspricht einem Bit in einer Ebene. Die vier Ebenen der EGA-Karte blenden Sie einzeln oder mehrere gleichzeitig an die Segmentadresse A000hex zur Manipulation des Bildschirmspeichers ein. Zur Darstellung eines Bildpunkts liest die EGA-Karte aus jeder Ebene ein Bit und setzt diese zu einem Nibble (4 Bit) zusammen. So entstehen zwei hoch vier, also 16 Farben. Das Auslesen der vier Bits eines Punkts erfolgt an immer der gleichen Adresse aus den vier Ebenen. Die vier Planes sind von 0 bis 3 durchnummeriert. Ein Bildpunkt (Pixel = Bit) in der Ebene 0 liefert so auch die Bitposition 0 der 4 Bit breiten Farbinformation. Diese Bildschirmspeicherorganisation ist auf den ersten Blick etwas verwirrend, da Ihre Grafikkarte die vier Bitplanes nicht hintereinander, sondern »übereinander« organisiert.

Der GDC verwaltet die Grafikfunktionen

● **Registerzugriff:** Der wichtigste Teilbaustein des Grafikkarten-Chipsatzes im Hinblick auf die Grafikmodi ist der GDC. Tabelle 2 zeigt Ihnen die neun Register des GDC auf. Der GDC verfügt über ein Index- und ein Datenregister. Das Indexregister liegt an der I/O-Portadresse 3CEhex an. Bevor Sie ein GDC-Register manipulieren, zeigen Sie dem GDC an, für welches Register Sie die nachfolgenden Daten bestimmen. Hierzu indizieren Sie durch einen Schreibzugriff auf die Portadresse 3CEhex eines der neun Register, indem Sie eine Out-Anweisung vornehmen:

```
mov dx,03ceh
mov al,<Index>
out dx,al
```

Das Datenregister des GDC erreichen Sie über die Portadresse 3CFhex. Nachdem Sie ein Register selektiert haben, schreiben Sie das Datenbyte wiederum durch eine Out-Anweisung in das Datenregister:

```
inc dx
mov al,<Datum>
out dx,al
```

Die Programmsequenz geht davon aus, daß Sie das GDC-Register in der oben genannten Weise indiziert haben und sich die Adresse 3CEhex noch im CPU-Register »DX« befindet. Die Prozessoren der 80x86-Reihe verfügen darüber hinaus über die Anweisung »out dx,ax«, mit deren Hilfe Sie die Selektierung und die Datenübertragung in einem Durchgang erledigen:

```
mov dx,03ceh
```

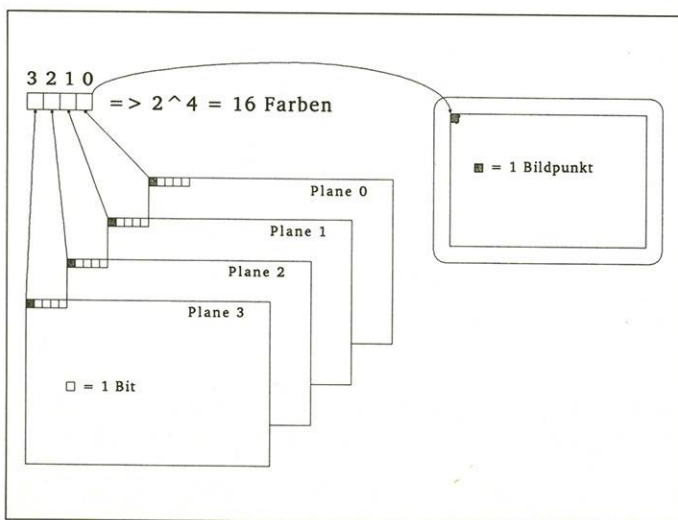


Bild 1. Das Vier-Ebenen-Speichermodell der EGA-Modi

Modus(hex)	Art	Auflösung	Farben	Videosegment(hex)	Seitengröße(Bytes)
00	Text	40x25	16	B800	2000
01	Text	40x25	16	B800	2000
02	Text	80x25	16	B800	4000
03	Text	80x25	16	B800	4000
04	Grafik	320x200	4	B800	16000
05	Grafik	320x200	4	B800	16000
06	Grafik	640x200	2	B800	16000
07	Text	80x25	2	B000	4000
0D	Grafik	320x200	16	A000	4x8000
0E	Grafik	640x200	16	A000	4x16000
0F	Grafik	640x350	2	A000	28000
10	Grafik	640x350	16	A000	4x28000
11*	Grafik	640x480	2	A000	38400
12*	Grafik	640x480	16	A000	4x38400
13*	Grafik	320x200	256	A000	64000

Tabelle 1. Die verschiedenen Grafikmodi der EGA/VGA-Karten

Schreibmodus 0 ist der vielfältigste der drei (EGA) beziehungsweise vier (VGA) Schreibmodi. Im Schreibmodus 0 beachtet der GDC den Inhalt beinahe aller anderen GDC-Register. Hierzu gehören die Register 0, 1, 3 und 8.

● **Schreibmodus 0:** Um einen oder mehrere Pixel auf dem Bildschirm einzufärben, führen Sie im Schreibmodus 0 folgende Schritte durch. Sie berechnen zunächst die Adresse des Bildpunkts, dessen Farbe Sie verändern möchten. Hierzu verwenden Sie folgende Standardformel:

$$\text{adr} = y * \text{breite} + x \text{ div } 8$$

Die Variable »breite« geben Sie in Bytes an. Im Modus 320x200 ist der Bildschirm 40 Byte breit (320 – 8 Pixel pro Byte). In den Modi 640x200, 640x350 und 640x480 umfaßt die Breite des Bildschirms 80 Byte. Die Formel ist so ausgerichtet, daß die Variable »y« im Bereich zwischen 0 und »Höhe des Bildschirms – 1« liegt. Die Variable »x« besitzt den Wertebereich zwischen 0 und 319 beziehungsweise 0 und 639. Der Koordinatenursprung liegt in der linken oberen Ecke des Bildschirms. Danach lesen Sie durch einen Prozessorbefehl ein Byte aus der so errechneten Bildschirmspeicheradresse. So füllen Sie die vier Latch-Register der EGA-Karte mit Daten.

Da ein Pixel einem Bit entspricht, wenden Sie sich nun dem Maskenregister zu. Die Bitmaske des Pixels ermitteln Sie durch folgende Formel:

$$\text{mask} = 80h \text{ shr } (x \text{ mod } 8)$$

Diese Bitmaske tragen Sie danach in das Maskenregister (Index 8) ein. Der GDC modifiziert so nur den Bildpunkt an der Koordinate x,y. Sie können natürlich auch mehrere Bildpunkte durch einen Schreibzugriff setzen. Dazu setzen Sie einfach die entsprechenden Bits im Maskenregister. Den Farbwert (0 bis 15), den Sie für den Bildpunkt vorsehen, schreiben Sie in das Register 0. Damit der GDC auch alle Bitplanes berücksichtigt, tragen Sie in das Register 1 den Wert »0Fhex« (Bits 0 bis 3 gesetzt) ein. Durch einen Schreibzugriff auf das Register 3 (»Rotieren/Verknüpfung«) wählen Sie – soweit gewünscht – eine bestimmte Anzahl von Bitrotationen sowie eine der vier Bitverknüpfungsoperationen. Abschließend führen Sie durch einen CPU-Schreibzugriff auf die zuvor berechnete Bildschirmspeicheradresse das Setzen eines oder mehrerer Bildpunkte aus. Den Wert, den Sie bei diesem Schreibvorgang spezifizieren, ist unerheblich, da Sie alle Informationen für den GDC bereits in die GDC-Register geschrieben haben. Der CPU-Schreibvorgang löst im GDC mehrere Vorgänge, beispielsweise die Durchführung der gewünschten Bitmanipulationen (Register 3), aus. Diese Vorgänge verlaufen zunächst in den vier Latch-Registern. Erst nachdem der GDC diese Arbeiten abgeschlossen hat, schreibt er die Inhalte der Latch-Register in die vier Ebenen.

● **Schreibmodus 1:** Der Schreibmodus 1 ist weit leichter zu handhaben als Schreibmodus 0. Dies liegt unter anderem

an den völlig anderen Einsatzgebieten dieses Schreibmodus. Sie verwenden den Schreibmodus 1, wenn Sie eine schnelle Speicherkopieroutine innerhalb des Bildschirmspeichers benötigen. Durch eine CPU-Leseoperation laden Sie die vier Latch-Register mit dem Inhalt der vier Ebenen an einer bestimmten Bildschirmspeicheradresse. Danach führen Sie eine CPU-Schreiboperation durch und übertragen so den Inhalt der vier Latch-Register an eine andere Bildschirmspeicherposition. Der GDC ignoriert im Schreibmodus 1 den Inhalt der GDC-Register 0, 1, 3 und 8, so daß keine Bitmanipulationen und -verknüpfungen stattfinden.

● **Schreibmodus 2:** Der Schreibmodus 2 stimmt in seiner Funktionsweise mit dem Schreibmodus 0 überein. Abweichungen von der oben genannten Programmsequenz entstehen lediglich bei der Farbauswahl. Die Farbe für zu setzende Bildpunkte im Schreibmodus 0 legen Sie durch Schreibzugriffe auf die beiden Register 0 und 1 fest. Die abschließende CPU-Schreiboperation überführt die durch das Maskenregister selektierten Bildpunkte in den neuen Farbzustand. Welches Datenbyte Sie bei dieser Schreiboperation verwenden, ist unerheblich. Im Schreibmodus 2 entfällt die Farblektierung durch die beiden genannten GDC-Register. Statt dessen spezifizieren Sie die gewünschte Farbe bei der abschließenden CPU-Schreiboperation. Der GDC beachtet hierbei nur die unteren vier Bits, die für die Auswahl der Farbe (0 bis 15) ausreichen. Der Schreibmodus 2 eignet sich daher hervorragend für das Setzen von Bildpunkten.

● **Schreibmodus 3:** Den Schreibmodus 3 stellt Ihnen nur die VGA-Karte zur Verfügung. In diesem Schreibmodus liest der GDC zunächst den Inhalt des Registers 0 aus. Wie Sie bereits erfahren haben, hält dieses Register für jede Ebene ein Bit bereit. Der GDC erweitert jedes Bit zu einem Byte und weist die erhaltenen vier Bytes den vier Ebenen zu. Ein Nullbit erweitert er zu dem Wert 0, ein gesetztes Bit zu dem Wert FFhex. Auch im Schreibmodus 3 führen Sie eine abschließende CPU-Schreiboperation durch. Den Wert verknüpft der GDC mit dem Inhalt des Maskenregisters (Index 8) durch ein logisches »UND«. Das Ergebnis verwendet er dann als Bitmaske zum Schreiben der vier Latch-Register in den Bildschirmspeicher. Der GDC beachtet im Schreibmodus 3 zusätzlich den Inhalt des GDC-Registers 3.

Lesen statt Schreiben, die umgekehrte Richtung

● **Lesemodus 0:** Der Lesemodus 0 ist der Standard-Lesemodus des GDC. Sie selektieren durch einen Schreibzugriff auf das Plane-Leseregister (Index 4) eine der vier Ebenen. Danach führen Sie eine CPU-Leseoperation durch und erhalten in dem verwendeten CPU-Register acht nebeneinander-

liegende Bildpunkte (1 Byte). Wenn Sie die Farbe eines Pixels benötigen, führen Sie diesen Lesevorgang für alle vier Planes hintereinander durch. Anschließend maskieren Sie das zu untersuchende Bit aus (in jedem gelesenen Byte) und setzen diese zu einem Nibble zusammen.

● **Lesemodus 1:** Der Lesemodus 1 ist der kompliziertere der beiden Lesemodi. In diesem Lesemodus selektieren Sie keine Plane, um aus dieser zu lesen. Statt dessen schreiben Sie in das Register »Farbvergleich« (Index 2) einen Farbwert zwischen 0 und 15. Danach führen Sie an einer bestimmten Bildschirmspeicheradresse eine CPU-Leseoperation durch. Der GDC liefert Ihnen in dem verwendeten CPU-Register Informationen über acht nebeneinanderliegende Bildpunkte. Ein gesetztes Bit zeigt Ihnen an, daß der Bildpunkt mit der Farbe des »Farbvergleich«-Registers übereinstimmt. Nullbits bedeuten das Gegenteil. Der GDC berücksichtigt im Lesemodus 1 den Inhalt des »Farbausnahme«-Registers (Index 7). Durch dieses Register bestimmen Sie, welche der vier Bitplanes der GDC in den Farbvergleich einbezieht und welche er von dem Farbvergleich ausnimmt.

● **Der Timer Sequencer:** Ein weiterer Teilbaustein, den Sie unter Umständen für die Programmierung der Grafikmodi benötigen, ist der TS (Timer Sequencer). Er besitzt ebenfalls ein Index- und ein Datenregister. Für die Programmierung des TS gelten die gleichen Programmierkonventionen, die Sie bei der Besprechung des GDC kennengelernt haben. Das Indexregister besitzt die Portadresse 3C4hex, das Datenregister liegt am I/O-Port 3C5hex an. Wie auch beim GDC, erlaubt nur die VGA-Karte Lesezugriffe auf die TS-Register. Das wichtigste Register des TS ist das Plane-Schreibregister (Index 2). Die unteren vier Bits repräsentieren wiederum die vier Bitplanes. Mit Hilfe dieses TS-Registers legen Sie fest, ob der TS eine Plane für Schreibzugriffe des Prozessors freischaltet oder sperrt. Sie benötigen dieses Register zum Beispiel bei der Darstellung von Spielfiguren (»Sprites«). Der GDC und der TS arbeiten eng zusammen. Alle Betriebsmodi, die Sie durch das Modusregister des GDC selektieren, finden durch den TS ebenfalls Beachtung.

Der TS und der GDC arbeiten eng zusammen

Das Zusammenspiel des GDC und des TS läßt sich am besten durch ein konstruiertes Beispiel veranschaulichen. Sie möchten ein Spiel schreiben, in dem Sie Sprites über einen Hintergrund bewegen, den Sie dabei nicht zerstören möchten. Zunächst sichern Sie den Hintergrund der Spielfigur, indem Sie den Lesemodus 0 des GDC aktivieren und nacheinander aus den vier Ebenen die zu sichernden Videoinformationen in einen Speicherbereich kopieren. Daraufhin

Index	Register
00	Aktivieren/Deaktivieren
01	Erlaube Aktivieren/Deaktivieren
02	Farbvergleich
03	Rotieren/Verknüpfung
04	Plane-Leseregister
05	GDC-Modus
06	Gemischtes Register
07	Farbausnahme
08	Maskenregister

Tabelle 2. Der GDC verfügt über neun Register.

Bits 3,4	Verknüpfung
00b	Schreiben
01b	Logisches UND
10b	Logisches ODER
11b	Logisches Exklusiv-ODER

Tabelle 3. Die EGA-Karte besitzt vier logische Verknüpfungen.

Bit	Bedeutung
0,1	Schreibmodus 0-2 (EGA) Schreibmodus 0-3 (VGA)
2	Testbedingung (immer 0)
3	Lesemodus 0-1
4	Odd/Even
5-7	Reserviert (0)

Tabelle 4. Über das GDC-Modusregister bestimmen Sie Schreib- und Lesemodus.

erfolgt die Darstellung der Spielfigur. Es ist vorteilhaft, die Spritedaten des Objekts ebenfalls in vier Ebenen aufzuteilen und diese hintereinander im normalen Hauptspeicher bereitzuhalten. Sie selektieren zur Darstellung der Spielfigur den Schreibmodus 2 des GDC. Dieser eignet sich für das intendierte Vorhaben am besten. Da eine Spielfigur nie rechteckig ist, legen Sie sich zusätzlich im Hauptspeicher eine Tabelle mit Bitmasken an. Diese erzeugen Sie, indem Sie die Spritedaten aller vier Ebenen durch ein logisches »ODER« miteinander verknüpfen. Bei der Darstellung des Sprites arbeiten nun GDC und TS zusammen. Die Bitmasken Ihres Sprites schreiben Sie in das Maskenregister (Index 8) des GDC. Über das Plane-Schreibregister des TS (Index 2) aktivieren Sie nun die vier Ebenen hintereinander und übertragen die Spritedaten aus dem Hauptspeicher in den Bildschirmspeicher.

Der CRTC spielt bei der Grafikprogrammierung eine untergeordnete Rolle. Sein Indexregister belegt die Portadresse 3x4hex, sein Datenregister die Portadresse 3x5hex. Das »x« in den Portadressen ist variabel und steht bei Verwendung eines monochromen Monitors für ein hexadezimalen »B«, bei Farbmonitoren für ein »D«.

Der ATC ist der Dritte im Bunde

Von Interesse sind hier nur die Register 0Chex und 0Dhex (Adreßregister). Dieses 16-Bit-Register enthält die Bildschirmspeicheradresse, ab der die Grafikkarte die Videoinformationen zur Anzeige bringt. Nach der Aktivierung eines Bildschirmmodus enthält dieses Register immer den Wert 0. Wenn Sie den Inhalt des Adreßregisters in 40- bis 80-Byte-Schritten (je nach Modus) erhöhen, stellen Sie fest, daß der Bildschirm nach oben rollt (»scrolling«). Das BIOS verwendet das Adreßregister zum Beispiel zur Realisierung mehrerer Bildschirmseiten. Wenn Sie sich beispielsweise im Grafikmodus 10hex befinden (640 × 350), benötigt die Grafikkarte 80 × 350 oder 28 000 Byte in jeder Plane. Die PC-Systemarchitektur sieht für den gesamten Bildschirmspeicher einen 64 KByte großen Bereich (von A000hex bis AFFFhex) vor. In diesen physikalischen Adreßbereich passen also zirka 2,3 Bildschirmseiten. Das BIOS rundet den Speicherbedarf einer Bildschirmseite so auf, daß eine numerisch gerade Anzahl von Bildschirmseiten entsteht. In diesem Fall sieht das BIOS 32 768 Byte (32 KByte) für eine Bildschirmseite vor. In diesem Grafikmodus können Sie also die Adressen 0 beziehungsweise 8000hex in das Adreßregister schreiben. Sie lösen damit einen visuellen Wechsel der Bildschirmseiten aus.

● **Der Attribut Controller:** Den ATC (Attribute Controller) Ihrer EGA-Karte benötigen Sie für die Wahl der 16 Bildschirmfarben. Die EGA-Karte erlaubt Ihnen die Auswahl der Bildschirmfarben aus einer Palette von 64 Farben. Es geht im folgenden nur um die 16 Farbbregister der EGA-Karte, obwohl Ihnen der ATC noch weitere Register zur Verfügung stellt. Die Programmierung des ATC gestaltet sich im Gegensatz zur Programmierung der anderen Teilkomponenten erheblich schwieriger. Der ATC besitzt zwar ebenfalls ein Index- und ein Datenregister, doch teilen sich diese eine Portadresse (3C0hex). Wenn Sie auf den ATC zugreifen, schreiben Sie zunächst den Registerindex in den Port

3C0hex. Hierbei schaltet der ATC ein Flipflop um, so daß Sie nun über die Portadresse 3C0hex das Datenregister erreichen. Problematisch an dieser Technik ist nur, daß Sie vor einem ATC-Zugriff nicht wissen, ob gerade das Index- oder das Datenregister an dieser Portadresse anliegt. Sie können jedoch das Flipflop des ATC in den Zustand »Index« überführen, indem Sie einen Lesezugriff auf den Port 3xAhex durchführen. Das »x« in der Portadresse besitzt die gleiche Bedeutung wie in den Portadressen des CRTC. An der Portadresse 3xAhex liegt das »Input Status Register 1« an, dessen Belegung Sie Tabelle 5 entnehmen. Das Resultat des Lesezugriffs ist in diesem Zusammenhang bedeutungslos. Die VGA-Karte erlaubt es Ihnen hier, alle ATC-Register auszulesen. Dies erledigen Sie nach der Indizierung eines ATC-Registers durch einen Lesezugriff auf die Portadresse 3C1hex.

Die 16 Farbregister (Palettenregister) besitzen die Registerindizes 0 bis 0Fhex. Nachdem Sie einen neuen Farbwert in eines der Palettenregister eingetragen haben, schreiben Sie den Wert 20hex in den Port 03C0hex. Mit diesem »magischen« Wert hat es folgende Bewandtnis: Das Indexregister des ATC ist nicht so einfach wie die Indexregister der anderen Teilkomponenten aufgebaut. Das Bit Nummer fünf im ATC-Indexregister entscheidet darüber, ob die CPU (Bit gelöscht) oder der ATC (Bit gesetzt) Zugriff auf die ATC-Register besitzt. Wenn Sie ein ATC-Register selektieren, löschen Sie implizit das Bit Nummer fünf, da Sie die Palettenregister über die Indizes 0 bis 0Fhex erreichen. Sie erlauben somit der CPU den Registerzugriff für eine folgende Schreibinstruktion. Es liegt in Ihrer Verantwortung als Programmierer, dem ATC nach der Datenübertragung den Zugriff auf die ATC-Register uneingeschränkt zu erlauben. Sie kommen dieser Aufgabe nach, indem Sie den Wert 20hex in das ATC-Indexregister schreiben und damit das Bit Nummer fünf setzen.

Die nächste Generation: 256 Farben

● **Der Grafikmodus 13hex:** Der VGA-Standardmodus 13hex mit seiner Auflösung von 320 x 200 Punkten bei 256 Farben ist weit einfacher zu programmieren als die 16-Farben-Grafikmodi. Nachdem Sie diesen Bildschirmmodus durch die Funktion 0 des Interrupts 10hex aktiviert haben, steht Ihnen ein linearer Grafikspeicher zur Verfügung. In dem 256-Farben-Modus der VGA-Karte repräsentiert ein Byte des Bildschirmspeichers einen Bildpunkt, der die Farbwerte 0 bis 255 annimmt. Zum Programmieren dieses Grafikmodus benötigen Sie keines der GDC- oder TS-Register. Sie lesen und schreiben einen Bildpunkt durch eine CPU-Lese-/Schreiboperation. Logische Verknüpfungen (»UND«, »ODER«, »Exklusiv ODER« und »NICHT«) wickeln Sie ebenfalls über die CPU ab. Die VGA-Karte trennt sich dennoch nicht völlig von dem Vier-Ebenen-Speichermode. Bild 2 zeigt Ihnen, wie die VGA-Karte die Videoinformationen im VGA-Modus 13hex in den vier Ebenen unterbringt. Allerdings fügt die VGA-Karte die Planes zu einem linearen Grafikspeicher zusammen.

● **Die DAC-Register:** Eine genauere Beschreibung verlan-

gen jedoch die DAC-Register (Digital Analog Converter) der VGA-Karte. Sie spezifizieren durch die DAC-Register die RGB-Anteile der 256 Farbregister. Der DAC konvertiert einen digitalen Farbwert vor der Ausgabe zum Monitor in einen analogen Wert. Tabelle 5 listet die DAC-Register mit den zugehörigen Portadressen auf. Das DAC-Maskenregister beinhaltet standardmäßig den Wert »FFhex«. Sie können ihn modifizieren, um bestimmte Farbwerte auszufiltern. Die Portadresse 3C7hex besitzt bei einem Lesezugriff eine andere Bedeutung als bei einem Schreibzugriff. Wenn Sie diesen I/O-Port beschreiben, legen Sie die Nummer eines Farbregisters für einen Lesezugriff fest. Eine Leseopera-

Portadresse(hex)	Zugriff	Register
3C6	Lesen/Schreiben	Maskenregister
3C7	Schreiben	Leseadresse
3C7	Lesen	Statusregister
3C8	Lesen/Schreiben	Schreibadresse
3C9	Lesen/Schreiben	Datenregister

Tabelle 6. Der DAC besteht aus fünf Registern.

tion liefert Ihnen hingegen den Status des DAC. Die Bits 0 und 1 informieren Sie darüber, ob ein Lesezugriff (beide Bits gelöscht) oder ein Schreibzugriff (beide Bits gesetzt) der aktuellen DAC-Operation vorausging. Über den Port 3C8hex spezifizieren Sie die Nummer eines Palettenregisters für einen Schreibzugriff. Die eigentliche Datenübertragung erfolgt in beiden Richtungen (Lesen und Schreiben) über die Portadresse 3C9hex. Wenn Sie ein Palettenregister mit einer neuen RGB-Farbkombination versehen möchten, so schreiben Sie ein Byte für jeden Farbanteil hintereinander in diesen Port. Der Lesezugriff erfolgt analog zum Schreibzugriff. Die RGB-Werte liegen in dem numerischen Bereich zwischen 0 und 63. So entstehen 64 hoch 3 oder 262 144 Farbwerte. Auf dem VGA-Markt gibt es seit einiger Zeit auch VGA-Karten, die über eine Farbpalette von 16,7 Millionen Farben verfügen. Wenn Sie eine solche VGA-Karte besitzen, übergeben Sie die RGB-Werte im Bereich zwischen 0 und 255.

»egavga.asm« (Listing 1) stellt Ihnen mehrere Assembler-Routinen zu den besprochenen Teilgebieten zur Verfügung. Der Prolog beziehungsweise Epilog der Routinen ist so ausgelegt, daß Sie die Routinen in die Programmiersprache Turbo Pascal implementieren können. Davon macht »demo.pas« (Listing 2) Gebrauch.

(Ingo Kubbilun/bm)

Bit	Bedeutung
0	Vertikaler/horizontaler Rücklauf
1,2	Reserviert (0)
3	Vertikaler Strahlrücklauf
4,5	Diagnose
6	Reserviert (0)
7	Vertikaler Strahlrücklauf (Komplement Bit 3)

Tabelle 5. Das Eingabestatusregister 1 liefert Ihnen mehrere Informationen.

[XL3]	1: ; Programm: egavga.asm
[R27]	2: ; Funktion: Funktionen zur Programmierung
[234]	3: ; der EGA/VGA-Karten
[T54]	4: ; Sprache : MASM oder TASM
[KZ3]	5: ; Autor : Ingo Kubbilun
[7A4]	6: ; (c) 1992 Widuch GmbH & Co. KG
[000]	7:
[XE4]	8: CODE segment byte public
[000]	9:
[3X3]	10: assume cs:CODE,ds:CODE
[GX4]	11: assume es:NOTHING,ss:NOTHING
[000]	12:
[962]	13: delay macro
[EA2]	14: jmp short \$+2
[EA2]	15: jmp short \$+2
[EA2]	16: jmp short \$+2
[HP0]	17: endm
[000]	18:
[WA2]	19: public detect
[KD3]	20: public activatemode
[MB3]	21: public restoremode
[WP2]	22: public waitfly1
[XP2]	23: public waitfly2
[TC3]	24: public setlinearadr
[HB3]	25: public setegacolor
[TA3]	26: public getegacolor
[2C3]	27: public setvgacolor
[AB3]	28: public getvgacolor


```

[7T2] 29:      public  putpixel
[5R2] 30:      public  getpixel
[DP2] 31:      public  copyrect
[6E3] 32:      public  colorcompare
[000] 33:
[DE2] 34: actmode  db      ?
[4E2] 35: oldmode  db      ?
[H72] 36: egavga   db      ?
[WC2] 37: crtadr   dw      ?
[TC2] 38: muladr   dw      ?
[662] 39: ofsy     dw      ?
[ED4] 40: logtab   dw      offset vmove
[0T2] 41:         dw      offset vand
[SH2] 42:         dw      offset vor
[9W2] 43:         dw      offset vxor
[4C4] 44: palette0  db      00h,01h,02h,03h
[M13] 45:         db      04h,05h,06h,07h
[C03] 46:         db      10h,11h,12h,13h
[W13] 47:         db      14h,15h,16h,17h
[9C4] 48: palette1  db      00h,08h,00h,00h
[613] 49:         db      18h,18h,00h,00h
[A03] 50:         db      00h,08h,00h,00h
[C03] 51:         db      00h,18h,00h,00h
[6C4] 52: palette2  db      00h,01h,02h,03h
[M13] 53:         db      04h,05h,06h,07h
[653] 54:         db      38h,39h,3ah,3bh
[083] 55:         db      3ch,3dh,3eh,3fh
[0T3] 56: palette   db      10h dup (?)
[000] 57:
[Y93] 58: detect    proc    near
[000] 59:
[7F1] 60:         push   bp
[PUI] 61:         mov    bp,sp
[462] 62:         mov    ax,1a00h
[YH1] 63:         int     10h
[0T1] 64:         xor     ah,ah
[EX1] 65:         cmp     al,lah
[EX1] 66:         jnz     isega
[NW1] 67:         cmp     bl,04h
[AR1] 68:         jz      isega
[RW1] 69:         cmp     bl,05h
[AR1] 70:         jz      isega
[2A1] 71:         inc     ah
[D43] 72: isega:      xchg    ah,al
[0T1] 73:         xor     ah,ah
[H13] 74:         mov     cs:egavga,al
[1D1] 75:         pop     bp
[3P0] 76:         ret
[V82] 77: detect      endp
[000] 78:
[MY3] 79: activatemode proc    near
[000] 80:
[7F1] 81:         push   bp
[PUI] 82:         mov    bp,sp
[7F1] 83:         push   bp
[XY1] 84:         mov    ah,0fh
[YH1] 85:         int     10h
[1D1] 86:         pop     bp
[W53] 87:         mov     cs:oldmode,al
[SY2] 88:         mov     ax,[bp+04h]
[XF1] 89:         push    ax
[0T1] 90:         xor     ah,ah
[YH1] 91:         int     10h
[RD1] 92:         pop     ax
[C63] 93:         mov     cs:actmode,al
[CH3] 94:         mov     cs:crtadr,03b4h
[RX1] 95:         cmp     al,0fh
[XR1] 96:         jz      adrok
[EW1] 97:         cmp     al,11h
[XR1] 98:         jz      adrok
[GH3] 99:         mov     cs:crtadr,03d4h
[S26] 100: adrok:      mov     cs:muladr,offset mul40
[S25] 101:         mov     cs:ofsy,40
[LX1] 102:         cmp     al,0dh
[GT1] 103:         jz      mulok
[7T4] 104:         mov     cs:muladr,offset mul80
[AS2] 105:         mov     cs:ofsy,80
[GW1] 106:         cmp     al,12h
[ WV1] 107:         jbe     mulok
[3Z4] 108:         mov     cs:muladr,offset mul320
[WC5] 109: mulok:      mov     si,offset palette0
[NX1] 110:         cmp     al,0eh
[6U1] 111:         jbe     palok
[Z24] 112:         mov     si,offset palettell
[RX1] 113:         cmp     al,0fh
[UR1] 114:         jz      palok
[Z34] 115:         mov     si,offset palette2
[R75] 116: palok:      mov     di,offset palette
[UL0] 117:         cld
[MF1] 118:         push    ds
[2V1] 119:         mov     ax,cs
[1V1] 120:         mov     ds,ax
[3V1] 121:         mov     es,ax
[NZ1] 122:         mov     cx,08h
[LZ1] 123:         rep     movsw
[ED1] 124:         pop     ds
[1D1] 125:         pop     bp
[741] 126:         ret     2
[FZ2] 127: activatemode endp

```

```

[000] 128:
[YX3] 129: restoremode proc    near
[000] 130:
[N91] 131:         push    bp
[C43] 132:         mov     al,cs:oldmode
[6P1] 133:         xor     ah,ah
[NB1] 134:         int     10h
[F71] 135:         pop     bp
[3P0] 136:         ret
[UV2] 137: restoremode endp
[000] 138:
[YG3] 139: waitfly1  proc    near
[000] 140:
[VR5] 141: ; Input Status Register 1 auslesen
[RN5] 142: ; Bit 3 ist 1, wenn Vertical Retrace
[081] 143: ; aktiv
[000] 144:
[123] 145:         mov     dx,cs:crtadr
[KV1] 146:         add     dx,06h
[A23] 147: retrx1:    in     al,dx
[AV1] 148:         and     al,08h
[UY1] 149:         jz      retrx1
[C23] 150: retry1:    in     al,dx
[AV1] 151:         and     al,08h
[052] 152:         jnz     retry1
[3P0] 153:         ret
[VF2] 154: waitfly1  endp
[000] 155:
[ZG3] 156: waitfly2  proc    near
[000] 157:
[0D5] 158: ; alternatives Warten auf Strahl-
[CS1] 159: ; rücklauf
[000] 160:
[123] 161:         mov     dx,cs:crtadr
[KV1] 162:         add     dx,06h
[D23] 163: retrx2:    in     al,dx
[AV1] 164:         and     al,08h
[XY1] 165:         jz      retrx2
[F23] 166: retry2:    in     al,dx
[WU1] 167:         and     al,01h
[352] 168:         jnz     retry2
[3P0] 169:         ret
[WF2] 170: waitfly2  endp
[000] 171:
[VX3] 172: setlinearadr proc    near
[000] 173:
[2A6] 174: ; Register »Linear Starting Address«
[D06] 175: ; des CRTC (Index 0Ch/0Dh) modifizieren
[000] 176:
[7F1] 177:         push    bp
[PUI] 178:         mov     bp,sp
[EM0] 179:         cli
[WY2] 180:         mov     cx,[bp+04h]
[123] 181:         mov     dx,cs:crtadr
[3Z1] 182:         mov     al,0ch
[6V1] 183:         out     dx,al
[SB1] 184:         inc     dx
[UV0] 185:         delay
[8T1] 186:         mov     al,ch
[6V1] 187:         out     dx,al
[LA1] 188:         dec     dx
[5Z1] 189:         mov     al,0dh
[6V1] 190:         out     dx,al
[SB1] 191:         inc     dx
[UV0] 192:         delay
[GT1] 193:         mov     al,cl
[6V1] 194:         out     dx,al
[6P0] 195:         sti
[1D1] 196:         pop     bp
[741] 197:         ret     2
[PY2] 198: setlinearadr endp
[000] 199:
[VX3] 200: setegacolor proc    near
[000] 201:
[RT5] 202: ; ATC-Palettenregister beschreiben
[000] 203:
[7F1] 204:         push    bp
[PUI] 205:         mov     bp,sp
[YY2] 206:         mov     bx,[bp+06h]
[CX1] 207:         and     bx,0fh
[WY2] 208:         mov     cx,[bp+04h]
[HX1] 209:         and     cx,3fh
[EM0] 210:         cli
[123] 211:         mov     dx,cs:crtadr
[KV1] 212:         add     dx,06h
[6L1] 213:         in     al,dx
[672] 214:         mov     dx,03c0h
[FT1] 215:         mov     al,bl
[6V1] 216:         out     dx,al
[UV0] 217:         delay
[GT1] 218:         mov     al,cl
[6V1] 219:         out     dx,al
[UV0] 220:         delay
[ZX1] 221:         mov     al,20h
[6V1] 222:         out     dx,al
[6P0] 223:         sti

```

Listing 1. Das Assembler-Programm »egavga.asm« assemblieren Sie mit dem MASM oder dem TASM.


```

[F13] 224:      cmp     cs:egavga,00h
[G21] 225:      jnz     setok
[V14] 226:      mov     cs:palette[bx],cl
[SM2] 227: setok:  pop     bp
[B41] 228:      ret     4
[RV2] 229: setegacolor endp
[000] 230:
[3X3] 231: getegacolor proc  near
[000] 232:
[RE5] 233: ; ATC-Palettenregister auslesen
[000] 234:
[N91] 235:      push    bp
[1T1] 236:      mov     bp,sp
[ZR2] 237:      mov     bx,[bp+04h]
[8P1] 238:      and     bx,0fh
[GX2] 239:      cmp     cs:egavga,00h
[G21] 240:      jnz     getcol
[3Z3] 241:      mov     al,cs:palette[bx]
[F71] 242:      pop     bp
[N11] 243:      ret     2
[8A2] 244: getcol:  cli
[533] 245:      mov     dx,cs:crtadr
[W11] 246:      add     dx,06h
[7A1] 247:      push    dx
[1G1] 248:      in      al,dx
[062] 249:      mov     dx,03c0h
[GP1] 250:      mov     al,bl
[NR1] 251:      out     dx,al
[UV0] 252:      delay
[W51] 253:      inc     dx
[1G1] 254:      in      al,dx
[181] 255:      pop     dx
[4A1] 256:      push    ax
[1G1] 257:      in      al,dx
[062] 258:      mov     dx,03c0h
[4S1] 259:      mov     al,20h
[NR1] 260:      out     dx,al
[Y71] 261:      pop     ax
[6P0] 262:      sti
[F71] 263:      pop     bp
[N11] 264:      ret     2
[ZU2] 265: getegacolor endp
[000] 266:
[CY3] 267: setvgacolor proc  near
[000] 268:
[GT5] 269: ; DAC-Palettenregister beschreiben
[000] 270:
[N91] 271:      push    bp
[1T1] 272:      mov     bp,sp
[EM0] 273:      cli
[S62] 274:      mov     dx,03c8h
[LR2] 275:      mov     al,[bp+0ah]
[NR1] 276:      out     dx,al
[W51] 277:      inc     dx
[UV0] 278:      delay
[AR2] 279:      mov     al,[bp+08h]
[NR1] 280:      out     dx,al
[UV0] 281:      delay
[8R2] 282:      mov     al,[bp+06h]
[NR1] 283:      out     dx,al
[UV0] 284:      delay
[6R2] 285:      mov     al,[bp+04h]
[NR1] 286:      out     dx,al
[6P0] 287:      sti
[F71] 288:      pop     bp
[V11] 289:      ret     8
[8W2] 290: setvgacolor endp
[000] 291:
[MX3] 292: getvgacolor proc  near
[000] 293:
[GE5] 294: ; DAC-Palettenregister auslesen
[000] 295:
[N91] 296:      push    bp
[1T1] 297:      mov     bp,sp
[EM0] 298:      cli
[N62] 299:      mov     dx,03c7h
[6R2] 300:      mov     al,[bp+10h]
[NR1] 301:      out     dx,al
[SM1] 302:      add     dx,02h
[UV0] 303:      delay
[1G1] 304:      in      al,dx
[5P1] 305:      mov     bh,al
[UV0] 306:      delay
[1G1] 307:      in      al,dx
[DP1] 308:      mov     bl,al
[UV0] 309:      delay
[1G1] 310:      in      al,dx
[6P1] 311:      mov     ch,al
[6P0] 312:      sti
[6P2] 313:      les     di,[bp+0ch]
[LM2] 314:      mov     es:[di],bh
[VM2] 315:      les     di,[bp+08h]
[VM2] 316:      mov     es:[di],bl
[RN2] 317:      les     di,[bp+04h]
[MM2] 318:      mov     es:[di],ch
[F71] 319:      pop     bp
[W41] 320:      ret     14
[GV2] 321: getvgacolor endp

```

```

[000] 322:
[9L3] 323: putpixel  proc  near
[000] 324:
[0D5] 325: ; Pixel im Schreibmodus 2 setzen
[000] 326:
[7F1] 327:      push    bp
[PU1] 328:      mov     bp,sp
[V92] 329:      mov     ax,0a000h
[3V1] 330:      mov     es,ax
[PZ2] 331:      mov     cx,[bp+0ah]
[6Z2] 332:      mov     dx,[bp+08h]
[LH2] 333:      call    cs:muladr
[G93] 334:      cmp     cs:actmode,13h
[YA2] 335:      jz      specput
[3V2] 336:      mov     al,es:[bx]
[TX1] 337:      mov     ah,80h
[BS1] 338:      shr     ah,cl
[DY1] 339:      mov     al,08h
[U92] 340:      mov     dx,03ceh
[VY1] 341:      out     dx,ax
[T52] 342:      mov     ax,0205h
[VY1] 343:      out     dx,ax
[8X2] 344:      mov     ah,[bp+04h]
[MU1] 345:      and     ah,03h
[9K1] 346:      shl     ah,1
[9K1] 347:      shl     ah,1
[9K1] 348:      shl     ah,1
[3Y1] 349:      mov     al,03h
[VY1] 350:      out     dx,ax
[SX2] 351:      mov     al,[bp+06h]
[RW2] 352:      mov     es:[bx],al
[1D1] 353:      pop     bp
[L41] 354:      ret     8
[7D4] 355: specput:  mov     di,[bp+04h]
[XU1] 356:      and     di,03h
[KK1] 357:      shl     di,1
[SX2] 358:      mov     al,[bp+06h]
[4B3] 359:      call    cs:logtab[di]
[1D1] 360:      pop     bp
[L41] 361:      ret     8
[6K2] 362: putpixel  endp
[000] 363:
[7H3] 364: getpixel  proc  near
[000] 365:
[V45] 366: ; Pixel im Lesemodus 0 einlesen
[000] 367:
[7F1] 368:      push    bp
[PU1] 369:      mov     bp,sp
[V92] 370:      mov     ax,0a000h
[3V1] 371:      mov     es,ax
[0Z2] 372:      mov     cx,[bp+06h]
[YY2] 373:      mov     dx,[bp+04h]
[LH2] 374:      call    cs:muladr
[G93] 375:      cmp     cs:actmode,13h
[W82] 376:      jz      specget
[N52] 377:      mov     ax,0005h
[U92] 378:      mov     dx,03ceh
[VY1] 379:      out     dx,ax
[6T1] 380:      xor     ch,ch
[KA1] 381:      inc     cl
[RX1] 382:      mov     ah,03h
[5Y1] 383:      mov     al,04h
[6V1] 384:      out     dx,al
[SB1] 385:      inc     dx
[133] 386: gloop:   mov     al,ah
[6V1] 387:      out     dx,al
[3V2] 388:      mov     al,es:[bx]
[ZR1] 389:      shl     al,cl
[WH1] 390:      rcl     ch,1
[X81] 391:      dec     ah
[7Y1] 392:      jns     gloop
[8T1] 393:      mov     al,ch
[1D1] 394:      pop     bp
[B41] 395:      ret     4
[484] 396: specget:  mov     al,es:[bx]
[1D1] 397:      pop     bp
[B41] 398:      ret     4
[4G2] 399: getpixel  endp
[000] 400:
[FG3] 401: copyrect  proc  near
[000] 402:
[TC5] 403: ; Bildschirmrechteck im Schreib-
[YY2] 404: ; modus 1 kopieren
[000] 405:
[G93] 406:      cmp     cs:actmode,13h
[9U1] 407:      jnz     crok
[V81] 408:      ret     12
[CP2] 409: crok:    push    bp
[PU1] 410:      mov     bp,sp
[MF1] 411:      push    ds
[YZ2] 412:      mov     cx,[bp+0eh]
[WZ2] 413:      mov     dx,[bp+0ch]
[LH2] 414:      call    cs:muladr
[2V1] 415:      mov     si,bx
[PZ2] 416:      mov     cx,[bp+0ah]
[6Z2] 417:      mov     dx,[bp+08h]
[LH2] 418:      call    cs:muladr
[4U1] 419:      mov     di,bx

```



```

[022] 420:      mov     cx,[bp+06h]
[UY2] 421:      mov     bx,[bp+04h]
[NM1] 422:      shr     cx,1
[NM1] 423:      shr     cx,1
[NM1] 424:      shr     cx,1
[U92] 425:      mov     dx,03ceh
[R52] 426:      mov     ax,0105h
[YV1] 427:      out     dx,ax
[V92] 428:      mov     ax,0a000h
[1V1] 429:      mov     ds,ax
[3V1] 430:      mov     es,ax
[UL0] 431:      cld
[R53] 432: copyloop: push    cx
[MF1] 433:      push    si
[PE1] 434:      push    di
[9Y1] 435:      rep     movsb
[GC1] 436:      pop     di
[ED1] 437:      pop     si
[VD1] 438:      pop     cx
[TN2] 439:      add     si,cs:ofsy
[VM2] 440:      add     di,cs:ofsy
[FA1] 441:      dec     bx
[3L2] 442:      jnz     copyloop
[ED1] 443:      pop     ds
[1D1] 444:      pop     bp
[V81] 445:      ret     12
[CF2] 446: copyrect  endp
[000] 447:
[8Z3] 448: colorcompare proc  near
[000] 449:
[VL5] 450: ; Lesemodus 1 aktivieren und Farb-
[X55] 451: ; vergleich über Color Compare
[3A2] 452: ; durchführen
[000] 453:
[G93] 454:      cmp     cs:actmode,13h
[WS1] 455:      jnz     ccok
[F41] 456:      ret     6
[ZM2] 457: ccok:      push    bp
[PU1] 458:      mov     bp,sp
[V92] 459:      mov     ax,0a000h
[3V1] 460:      mov     es,ax
[4Z2] 461:      mov     cx,[bp+08h]
[Z22] 462:      mov     dx,[bp+06h]
[LH2] 463:      call    cs:muladr
[562] 464:      mov     ax,0805h
[U92] 465:      mov     dx,03ceh
[YV1] 466:      out     dx,ax
[8X2] 467:      mov     ah,[bp+04h]
[UV1] 468:      and     ah,0fh
[1Y1] 469:      mov     al,02h
[YV1] 470:      out     dx,ax
[3V2] 471:      mov     al,es:[bx]
[1D1] 472:      pop     bp
[F41] 473:      ret     6
[303] 474: colorcompare endp
[LU3] 475: ; lokale Hilfsroutinen
[M33] 476: mul40      label  near
[FV1] 477:      mov     bx,dx
[VL1] 478:      shl     bx,1
[VL1] 479:      shl     bx,1
[VL1] 480:      shl     bx,1
[ZF1] 481:      push    bx
[VL1] 482:      shl     bx,1
[VL1] 483:      shl     bx,1
[RD1] 484:      pop     ax
[BRI] 485:      add     bx,ax
[CV1] 486:      mov     ax,cx
[HM1] 487:      shr     ax,1
[HM1] 488:      shr     ax,1
[HM1] 489:      shr     ax,1
[BRI] 490:      add     bx,ax
[CV1] 491:      and     cl,07h
[3P0] 492:      ret
[S33] 493: mul80      label  near
[FV1] 494:      mov     bx,dx
[VL1] 495:      shl     bx,1
[VL1] 496:      shl     bx,1
[VL1] 497:      shl     bx,1
[VL1] 498:      shl     bx,1
[ZF1] 499:      push    bx
[VL1] 500:      shl     bx,1
[VL1] 501:      shl     bx,1
[RD1] 502:      pop     ax
[BRI] 503:      add     bx,ax
[CV1] 504:      mov     ax,cx
[HM1] 505:      shr     ax,1
[HM1] 506:      shr     ax,1
[HM1] 507:      shr     ax,1
[BRI] 508:      add     bx,ax
[CV1] 509:      and     cl,07h
[3P0] 510:      ret
[753] 511: mul320     label  near
[7T1] 512:      mov     bh,d1
[RT1] 513:      xor     bl,bl
[ZL1] 514:      shl     dx,1
[ZL1] 515:      shl     dx,1
[ZL1] 516:      shl     dx,1
[ZL1] 517:      shl     dx,1

```

```

[ZL1] 518:      shl     dx,1
[ZL1] 519:      shl     dx,1
[ER1] 520:      add     bx,dx
[DR1] 521:      add     bx,cx
[3P0] 522:      ret
[663] 523: vmove      label  near
[RW2] 524:      mov     es:[bx],al
[3P0] 525:      ret
[223] 526: vand       label  near
[MT2] 527:      and     es:[bx],al
[3P0] 528:      ret
[S23] 529: vor        label  near
[CP2] 530:      or      es:[bx],al
[3P0] 531:      ret
[T43] 532: vxor       label  near
[XW2] 533:      xor     es:[bx],al
[3P0] 534:      ret
[112] 535: CODE      ends
[4M0] 536:      end

```

Listing 1. Das Assembler-Programm »egavga.asm« assemblieren Sie mit dem MASM oder dem TASM.

```

[AF3] 1: { Programm: demo.pas
[HV6] 2:      Funktion: Demonstration der Funktionen
[743] 3:      aus »egavga.asm«
[5V4] 4:      Sprache : Turbo Pascal ab V4.0
[0R3] 5:      Autor   : Ingo Kubbilun
[FL4] 6:      (c) 1992 Widuch GmbH & Co. KG }
[000] 7:
[6M3] 8: program egavga_demo;
[000] 9:
[A72] 10: uses crt,dos;
[000] 11:
[WM2] 12: const vmove = 0;
[7G1] 13:      vand = 1;
[0H1] 14:      vor = 2;
[4L1] 15:      vxor = 3;
[WW3] 16: var regs : Registers;
[142] 17:      karte : Byte;
[GP1] 18:      ch : Char;
[T22] 19:      i,x,y : Word;
[Y62] 20:      s : string;
[000] 21:
[TS2] 22: {$L egavga.obj}
[000] 23:
[HM5] 24: function detect : Byte; external;
[076] 25: procedure activatemode( mode : Byte );
[VL1] 26: external;
[5M5] 27: procedure restoremode; external;
[P25] 28: procedure waitfly1; external;
[T25] 29: procedure waitfly2; external;
[R36] 30: procedure setlinearadr ( adr : Word );
[VL1] 31: external;
[RR6] 32: procedure setegacolor ( reg, col : Byte );
[VL1] 33: external;
[UT6] 34: function getegacolor ( reg : Byte ): Byte;
[VL1] 35: external;
[7E6] 36: procedure setvgacolor ( reg, red, blue,
[724] 37: green : Byte ); external;
[0V5] 38: procedure getvgacolor ( reg : Byte;
[H76] 39: var red, blue, green : Byte ); external;
[WY6] 40: procedure putpixel ( x,y : Word; col, mask:
[6V2] 41: Byte ); external;
[CB6] 42: function getpixel ( x,y : Word ) : Byte;
[VL1] 43: external;
[GD6] 44: procedure copyrect ( x1,y1,x2,y2,width,
[F54] 45: height : Word ); external;
[BV5] 46: function colorcompare ( x,y : Word )
[LT2] 47: : Byte; external;
[Z15] 48: procedure print( x, y, c : Byte;
[HY1] 49: s : string );
[000] 50:
[2P2] 51: var i : Integer;
[000] 52:
[MU0] 53: begin
[F33] 54:     portw[$3ce]:=$0005;
[673] 55:     portw[$3ce]:=$ff08;
[933] 56:     portw[$3ce]:=$0003;
[333] 57:     portw[$3ce]:=$0001;
[9X1] 58:     regs.ah:=$02;
[AX1] 59:     regs.bh:=$00;
[NY1] 60:     regs.dh:=y;
[Z21] 61:     regs.dl:=x;
[4C2] 62:     intr($10,regs);
[8U3] 63:     for i:=1 to Length(s) do
[MU0] 64:         begin
[WX1] 65:             regs.ah:=$0e;
[XB3] 66:             regs.al:=Ord(s[i]);
[XW1] 67:             regs.bl:=c;
[4C2] 68:             intr($10,regs);

```

Listing 2. »demo.pas« demonstriert Ihnen einige Fähigkeiten der EGA-Karte.


```
[ZN0] 69: end;
[ZN0] 70: end;
[000] 71:
[C95] 72: procedure rect ( xl,y1,w,h : Word;
[P13] 73: col,mask : Byte );
[000] 74:
[GG2] 75: var i,j : Word;
[000] 76:
[MU0] 77: begin
[DU2] 78: for i:=0 to h-1 do
[DV2] 79: for j:=0 to w-1 do
[PS4] 80: PutPixel(xl+j,y1+i,col,mask);
[ZN0] 81: end;
[000] 82:
[MU0] 83: begin
[VC2] 84: karte:=Detect;
[7W2] 85: activatemode($0d);
[RV4] 86: print(0,24,15,'320x200, 16 Farben');
[V86] 87: print(0,0,9,'PutPixel/GetPixel-Test');
[R95] 88: print(0,1,10,'Taste drücken...');
[222] 89: ch:=ReadKey;
[VP0] 90: i:=1;
[P31] 91: repeat
[2T1] 92: Str(i+1,s);
[PG2] 93: print(37,24,i,s);
[3V2] 94: for y:=0 to 191 do
[DV2] 95: for x:=0 to 319 do
[UR3] 96: PutPixel(x,y,i,vmove);
[3V2] 97: for y:=0 to 191 do
[DV2] 98: for x:=0 to 319 do
[KW3] 99: if GetPixel(x,y)=i then
[FN3] 100: PutPixel(x,y,0,vmove);
[951] 101: Inc(i);
[YG4] 102: until (i=16) or KeyPressed;
[222] 103: ch:=ReadKey;
[8W2] 104: activatemode($0e);
[YV4] 105: print(0,24,15,'640x200, 16 Farben');
[4M3] 106: rect(0,0,640,110,1,vmove);
[0M3] 107: rect(0,60,50,40,10,vmove);
[KW3] 108: rect(160,60,50,40,11,vmove);
[9W3] 109: rect(320,60,50,40,12,vmove);
[5X3] 110: rect(480,60,50,40,13,vmove);
```

```
[VL3] 111: rect(0,10,50,40,10,vmove);
[7R3] 112: rect(160,10,50,40,11,vand);
[Eh3] 113: rect(320,10,50,40,12,vor);
[LV3] 114: rect(480,10,50,40,13,vxor);
[0M3] 115: rect(0,60,50,40,10,vmove);
[KW3] 116: rect(160,60,50,40,11,vmove);
[9W3] 117: rect(320,60,50,40,12,vmove);
[5X3] 118: rect(480,60,50,40,13,vmove);
[GE3] 119: print(0,0,10,'NORMAL');
[DU2] 120: print(20,0,10,'AND');
[SN2] 121: print(40,0,10,'OR');
[0X2] 122: print(60,0,10,'XOR');
[VY5] 123: print(0,14,15,'Sie sehen jeweils zwei'+
[1S4] 124: ' Rechtecke untereinander. ');
[AT5] 125: print(0,15,15,'Die unteren Rechtecke '+
[1T6] 126: 'zeigen die normalen Farben, die oberen'+
[AF2] 127: ' weisen Farb- ');
[PX5] 128: print(0,16,15,'verfälschungen durch '+
[VV6] 129: 'logische Verknüpfungen der EGA-Karte ' +
[V21] 130: 'auf. ');
[222] 131: ch:=ReadKey;
[RV2] 132: activatemode($10);
[6W4] 133: print(0,23,15,'640x350, 16 Farben');
[R95] 134: print(0,0,10,'Rechteck kopieren');
[4Y3] 135: rect(96,96,96,100,12,vmove);
[R95] 136: print(0,1,10,'Taste drücken...');
[222] 137: ch:=ReadKey;
[X94] 138: copyrect(96,96,224,120,96,100);
[222] 139: ch:=ReadKey;
[D95] 140: print(0,0,15,'Bildschirmrollen ');
[W95] 141: print(0,1,15,'Taste drücken...');
[DT2] 142: for i:=0 to 410 do
[MU0] 143: begin
[5L1] 144: waitfly2;
[X33] 145: setlinearadr(i*80);
[ZN0] 146: end;
[PA2] 147: regs.ax:=$0003;
[4C2] 148: intr($10,regs);
[KN0] 149: end.
```

Listing 2. »demo.pas« demonstriert Ihnen einige Fähigkeiten der EGA-Karte.

Impressum

Herausgeber: Helmuth Schmitz
Redaktionsdirektor: Volker Everts

Redaktion

Chefredakteur: Ulrich Eike (ue)
Chefin vom Dienst: Gabi Börgers (gb)
Leitender Redakteur: Wolfgang Hofner (wh)
Redaktion: Burkhard Müller (bm), Erhard Thomas (et),
Kerrin Uhlmann (uk)
Redaktionsmarketing: Ingo Everts (ie)
Textredaktion: Eva-Maria Brautlacht (eb), Thomas Bruer (tb),
Margarete Lang (ml), Kristiana Schulz (ks)
Technische Assistenz: Patrick Dürr (du), Assen Vassilev (av)
Redaktionsassistent: Anke Ludwig, Helga Menne, Lydia Roitzheim
Gestaltung und Layout: Christiana Seiser, Michaela Küppers
Grafik: Leonhard Büttner
Fotografie: Heinz Harcuba
Leitung des Redaktionsbüros: Volker Everts
Anschrift Redaktion: Redaktion DOS International,
Gruberstr. 46c, 8011 Poing bei München,
Telefon: (0 81 21) 769-0,
Telefax: (0 81 21) 790 46

Sie erreichen die Redaktion werktags telefonisch
von 15 bis 17 Uhr.

Manuskripte, Programme bitte an die Redaktion DOS International,
Gruberstr. 46c, 8011 Poing bei München, senden.
Für unverlangt eingesandte Manuskripte und Datenträger sowie
Fotos übernehmen der Verlag und die Redaktion keine Haftung.
Die Zustimmung zum Abdruck wird vorausgesetzt. Das
Urheberrecht für veröffentlichte Manuskripte liegt ausschließ-
lich beim Verlag. Nachdruck sowie Vervielfältigung oder son-
stige Verwertung von Texten nur mit schriftlicher Genehmigung
des Verlages. Für veröffentlichte Programme und Basteleien
übernehmen wir weder Gewähr noch Haftung. Schaltungen
und verwendete Bezeichnungen müssen nicht frei von gewerb-
lichen Schutzrechten sein. Namentlich gekennzeichnete
Fremdbeiträge geben nicht in jedem Fall die Meinung der Re-
daktion wieder.

Anzeigenverkauf

Anzeigenleitung: Wolfgang Brill
Anzeigenverkauf für PLZ 1, 4, 5:
Karina Ehrlich (0 56 51) 80 93 71, Telefax (0 56 51) 80 94 44
Bernd Heckmann (0 56 51) 80 93 81,
Gerlinde Rachow (0 56 51) 80 93 90,
Sylvia Stephani (0 56 51) 80 93 80,
Anzeigenverkauf für PLZ 2 + 3
DMV-Verlagsbüro Hamburg, Kunastr. 4a, 2000 Hamburg 65,
Telefon: (0 40) 60 03 075 - 77, Telefax: (0 40) 60 15 475
Leitung: Sylvia Ehrenpfordt
Anzeigenverkauf: Sabine Bindseil
Coordinating Manager Asia: Ralph Streng
Coordinating Manager USA: Tobias J. Hoffmann
Anzeigenverkauf für PLZ 6 - 8
DMV-Verlagsbüro München, Zaunkönigweg 2c,
8000 München 82,
Telefon: (0 89) 4 39 10 87, Telefax: (0 89) 4 39 10 80
Leitung: Britta Fiebig
Anzeigenverkauf: Peter Schätzle, Hannelore David, Ilona Sehm
Repräsentanten im Ausland:
Great Britain: Huson European Media, Gerry Rhoades-Brown,
10-11 The green Business Centre, The Causeway, Staines, Mid-
dlesex, TW18 3AL, Telephone No: 0784 46 99 00 Fax No: 0784
46 99 96
Taiwan: PRISCO, Mr. Parson Lee, 2F, No.17, Alley 7, Lane 217,
Chung Hsiao E. Rd. Sec. 3, Taipei, Taiwan, R.O.C. Phone:
[886]-2-711 48 33-6, Fax: [886]-2-741 51 10,
Hong Kong: ELECTRONIC (HK) PUBLISHING CO: Mr. Herman
Lau, 1301 Po Sang Bank Building, 33 Argyle Street, Kin., Hong
Kong. Tel.: 391 2977, 391 5088, FAX: 852 789 8027, Cable: ELPUB
Singapore: Major Media Private Ltd., Mr. Hoo Siew-Sai, 6th
Fl., 52 Chin Swee Road, Singapore 0316. Phone: (65)-738 0122,
Fax: (65)-738 2108
Anzeigenverwaltung: Andrea Giese, Beate Kranz
Druckunterlagen-Disposition
Christina Wabra (0 56 51) 809 - 372
Anzeigenpreise
Es gilt die Anzeigenpreisliste Nr. 7 vom 01.12.1991
Anzeigenrundpreise
1/1 Seite s/w DM 9.900, - . Farbzuschlag: erste und zweite
Zusatzfarbe aus Europaskala je DM 1.800, -
Vierfarbzuschlag DM 5.400, -.

Verlag

Produktionsleitung: Uwe Siebert
Koordination: Martina Gress
Fotosatz: Martina Kühn
Montage/Reprografie: Margarete Schenk (Lg.),
Helmut Skoupy (Lg.), Manuela Eska, Andrea Gundlach
Art director: Mohamed Hawa
Werbegrafik: Uwe Kalischinsky, Norbert Risch
Anschrift Verlag: DMV Daten- und Medienverlag Widuch
GmbH & Co. KG, Fuldaer Str. 6, 3440 Eschwege,
Telefon: (0 56 51) 8 09-0, Telefax: (0 56 51) 8 09-3 33
Vertrieb: Verlagsunion Erich Pabel-Arthur Moewig KG (VPM)
Friedrich-Bergius-Straße 20, 6200 Wiesbaden
Druck: Druckerei Jungfer, 3420 Herzberg
Bezugspreise
DOS Extra erscheint dreimonatlich. Einzelpreis DM 18/
sfr 18/6S 145/Lit 15.900
Abonnementpreise
Die Preise verstehen sich grundsätzlich
einschließlich Porto und Verpackung.
Inland:
Bezugspreis 4 Ausgaben: DM 60, -
Europäisches Ausland:
Bezugspreis 4 Ausgaben: DM 70, -
Außereuropäisches Ausland:
Bezugspreis 4 Ausgaben: DM 80, -
Bankverbindungen:
Postcheck Frankfurt/M: Kto.-Nr.: 23043-608
Raiffeisenbank Eschwege: BLZ: 522 603 85, Kto.-Nr.: 245 7008
Die Abonnementbestellung kann innerhalb einer Woche nach
Auftrag beim DMV-Verlag, Postfach 250, 3440 Eschwege,
schriftlich widerrufen werden. Zur Wahrung der Frist reicht
der Poststempel. Das Abonnement verlängert sich automatisch
um 6 bzw. 12 Monate, wenn es nicht mindestens 6 Wochen vor
Ablauf beim Verlag schriftlich gekündigt wird.

DMV PROGRAMMIERER-TOOLS

Neo Graph

Die Turbo-Treiber-Toolbox für EGA/VGA-Grafikkarten

Turbo Pascal macht bei 640 x 480 Schluß.
Hier fängt NeoGraph an!

NeoGraph versetzt Sie in die Lage, die Möglichkeiten Ihrer Grafikkarte voll auszunutzen: höchste Auflösungen bei optimaler Farbwiedergabe. Die Programmsammlung NeoGraph stellt ein unverzichtbares Komplettpaket für die Programmierung von Grafiktreibern in (Turbo) Pascal dar, insbesondere, wenn es um höhere Auflösungen als 640 x 480 geht.



- NeoGraph enthält einen Satz vorgefertigter Grafikartentreiber (bis 1024 x 768 bei 256 Farben), zum Beispiel für Tseng, Genoa, Video7, u.a.
- Einige Prototypen von Treibern dienen dazu, auch kommende Formate auszunutzen, und um eigene Sonderformate und Programme anpassen zu können.
- Alle Routinen liegen im Quellcode bei.
- Die Programme sind ausgelegt, um auf DOS-Rechnern in Turbo Pascal ab der Version 4.0 eingebunden zu werden.
- Damit die erweiterten Grafikmodi genutzt werden können, gibt es zwei neue Units: NGRAPH und NCRT!
- Beide haben wesentliche Erweiterungen und Verbesserungen gegenüber den Originalen.
- Ausführliche Dokumentierung versetzt auch "Normalmenschen" in die Lage, die Programme professionell zu nutzen.
- Drei Testprogramme dienen der Erforschung der Video-Hardware.
- Ein sehr detailliertes Handbuch führt auch Einsteiger behutsam an die Programmierung heran. Der Autor Arne Schäpers ist bekannt für seine hintergründige, aber plausible Art zu schreiben.

NeoGraph

DM 99,—*

GraphBas

BGI-Erweiterung für Microsoft Quick Basic & Turbo Basic

GraphBas PC erschließt Ihrem Basic-Compiler bisher ungeahnte Grafikwelten. Was Turbo Pascal und Turbo C berühmt gemacht hat, gibt es jetzt auch für Basic: • Volle Nutzung der BGI-Vektorfonts • Editor für eigene Schriften, Logos und Pictogramme • 3D-Balken- und Tortendiagramme, kinderleicht aufgerufen • Farben, Füllmuster und Figuren mit Assembler-routinen • einfachste Handhabung durch Quellcodes-Einbindung • Unterstützung aller Grafikkarten auf dem PC

GraphBas PC

je DM 99,—*

BGI-Grafikerweiterung für Microsoft Quick oder Turbo Basic

MaskEdit PLUS 5.0

Professioneller Maskengenerator liefert Quellcodes Ihrer Wahl

- Statt Listings tippen, Menüpunkte auswählen.
- Schnelle Bildschirmmasken für Ihre Programme: • hierarchische Menüs • verschiedene Ausgabefenster • 8 Rahmentypen, 16 Farben, Schattenwurf • ASCII-, dBase-, Binär- und ANSI-Ausgabeformate als Standard • optimaler Treiber für die Ausgabesprache Ihrer Wahl • alle Treiber im Quelltext • einfaches, aber effizientes Arbeiten mit über 50 Funktionen • Komfort durch kontextbezogene Hilfen

Zusatztreiber-Auswahl:

Turbo Pascal 3/4/5/6, Turbo C und Microsoft C, Topspeed-Modula-2, Turbo Basic, Quick Basic

MaskEdit Plus, ein Zusatztreiber

DM 99,—*

MaskEdit Plus, zwei Zusatztreiber

DM 149,—*

MaskEdit Plus, drei Zusatztreiber

DM 199,—*

Prof Ed

Der Editor für Programmierer

Ob Sie programmieren, einfach Texte schreiben oder Ihre Config.sys- oder Autoexec.bat-Dateien umändern wollen: ProfEd ist das ideale Werkzeug für Sie:

- Bis zu fünf Dateien lassen sich gleichzeitig bearbeiten.
- Mehrere MByte lange Dateien sind kein Problem mehr.
- Jede Taste der Tastatur kann mit einem oder mehreren Kommandos versehen werden.
- Makros gestatten, beliebig viele Editor-kommandos abzuabufen.
- Mehrere Undo-Funktionen lassen den letzten Arbeitsschritt rückgängig machen.
- Teile einer oder verschiedener Dateien lassen sich gleichzeitig am Bildschirm zur Bearbeitung bereitstellen.
- Suchen und Ersetzen von Zeichenfolgen ist problemlos.
- Zeilen- und spaltenweises Löschen, Kopieren, Verschieben und Füllen von markierten Textbereichen gestatten ein schnelles Bearbeiten Ihrer Dateien. Diese Funktion arbeitet dateiübergreifend und ermöglicht jede nur denkbare Textmanipulation.

ProfEd

DM 99,—*

neu neu neu neu neu neu neu

DMV-PowerTools für Basic

neu neu neu neu neu neu neu

Professionelle Tools für Basic-Programmierer zur Erstellung von Programmen und Oberflächen gemäß dem SAA-Standard.

Reizen Sie Basic voll aus! Steigen Sie ein in die Interrupt-Programmierung! Schreiben Sie sich Ihre eigene Arbeitsumgebung mit allen Raffinessen! Die DMV-PowerTools für Basic stellen die richtigen Module zur Verfügung, damit Sie unabhängig Ihre eigenen Routinen entwickeln können. Wie wäre es zum Beispiel mit einem Kopierprogramm?

Die PowerTools beinhalten über 200 Befehle und Funktionen, und zwar mit folgenden Schwerpunkten:

- MAUSTEXT/MAUSGRAF: Routinen zur Maussteuerung im Textmodus/Grafikmodus
- CONVERT: Konvertier-routinen, Funktionen zur Bitmanipulation, Funktionen zur Umrechnung von Zahlensystemen
- BIOS/2: BIOS-Funktionen (Grundfunktionen + Erweiterungen)
- DOS: DOS-Funktionen • WINDOW: Fensterverwaltung • STDIO1/2/3: Ein-/Ausgabe (u.a. Listenauswahl, Menüs, Zeilen-/Maskeneditor, Dialogroutinen)
- LASERJET: Anweisungen zur Steuerung eines HP-kompatiblen Laserdruckers
- MATH: mathematische Funktionen (Trigonometrie, Runden, Formatierung, Umrechnungsfunktionen)
- MAUS.MCD: vordefinierte Mauscursor für den Grafikmodus • u.a.m.

Sämtliche Module liegen im jeweiligen Basic-Quelltext vor.

Sofort bestellen!

Denn jedes PowerToolspaket enthält
jetzt eine komplette Anwendung
(Musikbox) als kostenlose Zugabe
inklusive 30-seitigem Handbuch

(Oder fordern Sie für 10,— DM ein Software-Pröbchen mit fünf Demoprogrammen an)

DMV-PowerTools I für PowerBasic 2.10

DM 149,—*

DMV-PowerTools II für Quick Basic 4.5

DM 149,—*

DMV-PowerTools III für MS-Basic PDS 7.1

DM 149,—*

* Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.

DMV-Verlag • Postfach 250 • W-3440 Eschwege

DMV Windows Tools

Wir machen Windows perfekt!

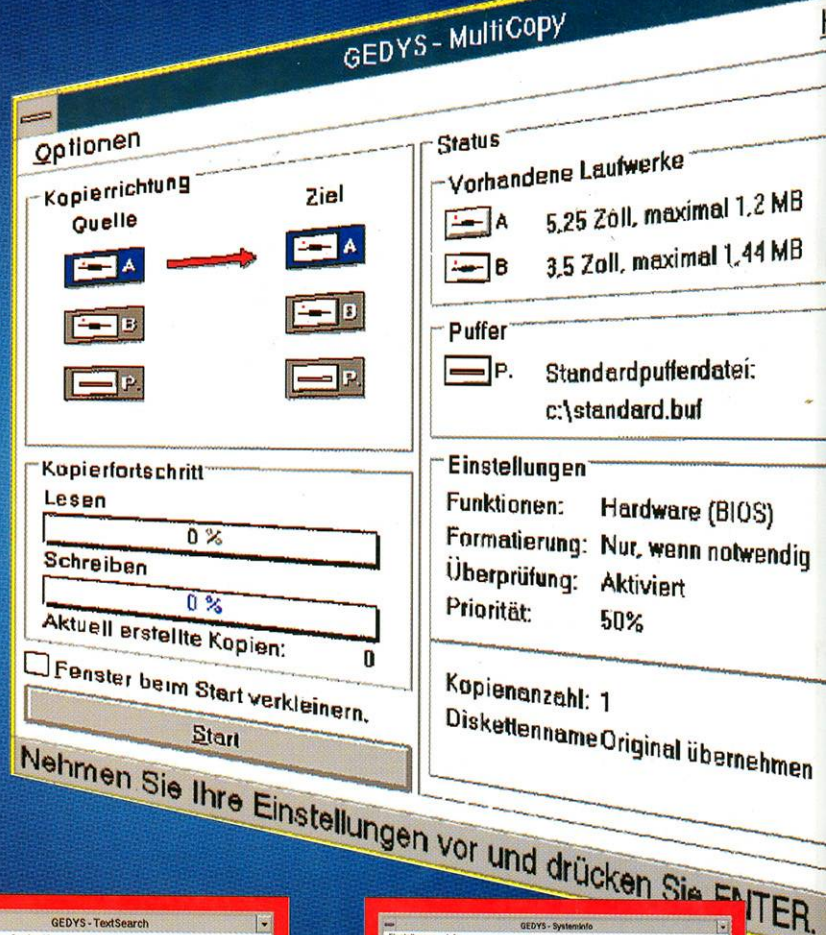
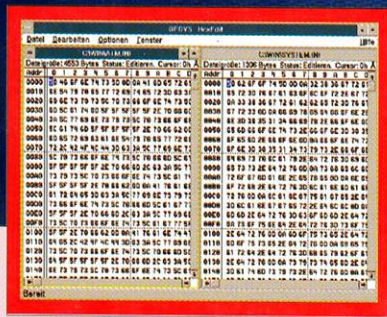


MultiCopy

Spielen Sie gerne Diskjockey?
MULTICOPY dupliziert Disketten super-
schnell und ohne Wechslen

- nutzt EMS, RAM-Disk oder Festplatte
- macht Mehrfachkopien
- kopiert auch zu späterem Zeitpunkt
- bietet Sicherheit durch Datenüberprüfung
- formatiert automatisch nach Bedarf
- kopiert auch verschiedene Diskettenformate
- berücksichtigt auch den Bootsektor
- numeriert Disketten und vergibt Namen
- arbeitet im Hintergrund

99,-DM*

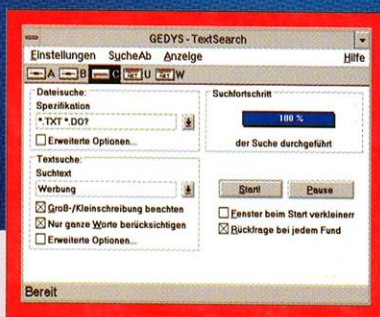


HexEdit

Wie war doch die interne Versionsnummer?
HEXEDIT zeigt Daten und Programme hexa-
dezimal und zeichenorientiert

- sucht Text und Bytefolgen
- läßt Änderungen im Programmcode zu
- kann abspeichern
- hat Suchen-und-Ersetzen-Funktion
- gibt Sicherheit durch Undo-Funktionen
- kann schrittweise rückgängig machen
- verwendet ANSI- und IBM-Zeichensatz
- Bildschirmteilung horizontal und vertikal
- editiert mehrere Dateien gleichzeitig bis
- zu einer Größe von 64 MByte

99,-DM*

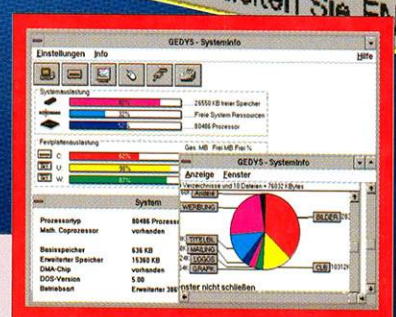


TextSearch

Wer hat den Brief Müller wegen Windows?
TEXTSEARCH sucht schnell und einfach

- auch Kombinationen von Suchbegriffen
- verarbeitet Wildcards
- findet Texte und Dateien
- sucht über ganzes Netz
- zeigt im Protokoll alle Ergebnisse
- selektiert komfortabel die Suchergebnisse
- zeigt das Textumfeld des Fundortes an
- lädt die Texte mit Programmen
- verwendet DOS- oder Windows-Zeichensatz
- arbeitet als Hintergrundprogramm

99,-DM*



SystemInfo

Wie viele Farben hat eigentlich Ihre Grafikkarte?
SYSTEMINFO gibt Auskunft über die
Hardware

- listet die aktuelle Konfiguration
- analysiert die Systemauslastung
- überwacht die Prozessorkapazitäten
- zeigt die Belegung der Speicher
- gibt grafisch Auskunft über Platzbelegungen
- gibt interne Infos bei Hardware-Problemen
- über sämtliche Festplatten, auch im Netz
- hilft, bevor Windows zu langsam wird
- optimale Nutzung Ihres Windows-Systems

99,-DM*



* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,- bzw. für das Ausland DM 6,- Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.
DMV-WINDOWS-TOOLS sind Produkte der
Gedys Software Consulting GmbH

DMV-Verlag • Postfach 250 • 3440 Eschwege

