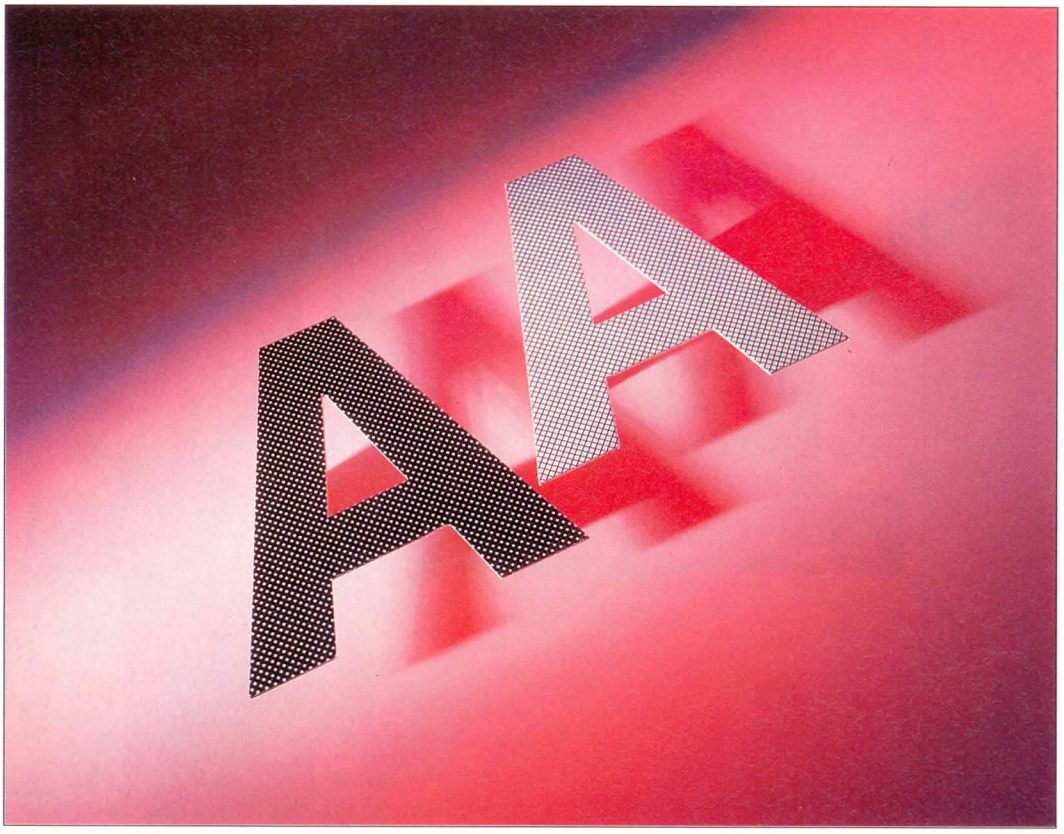

Von Hercules bis VGA

Neue Pixelfonts für die BGI-Treiber von Turbo-Pascal 5.5 und für Turbo C 2.00 (C++)

Hartwig Thomas

Der professionelle Programmierer ist immer wieder mit dem Problem konfrontiert, auf der IBM-Bildschirm-Architektur im Grafikmodus Texte auszugeben. Für zwei beliebte Hochsprachen wird hier vorgeführt, wie man die grafische Textausgabe der jeweiligen Rechnerausstattung anpaßt.

Schriftzüge oder Texte im Grafikmodus darzustellen, ist mit den Entwicklungsumgebungen Turbo C oder Turbo-Pascal von Borland kein Problem, da zum BGI (Borland Graphics Interface) auch die zugehörigen Vektorschriften (BGI-Schrifttyp 1–4) mitgeliefert werden. Nun sind diese Vektorschriften zwar sehr hübsch, bereiten aber – bedingt durch Rundungsfehler – Schwierigkeiten bei der Wiedergabe kleiner Schriften für verschiedene Bildschirmauflösungen.

Damit ein Programm aber auf jedem Rechner genutzt werden kann, muß es die heute gängigen Bildschirmtypen beziehungsweise Grafikkarten Hercules, CGA (Color Graphics Adapter), EGA (Enhanced Graphics Adapter), VGA (Video Graphics Adapter), SuperVGA und andere unterstützen. Eine Alternative wäre der BGI-Schrifttyp 0 – die Pixelschrift. Gegen deren Benutzung spricht zwar nichts, im VGA-Modus stellt sie sich jedoch als sehr klein dar, da die Buchstaben aus 8×8-Punktmatrizen bestehen, und je größer die Auflösung, desto kleiner sind demzufolge auch die Buchstaben. Die Idealgröße einer Pixelmatrix für Buchstaben in hochauflösenden VGA-Modi bei einem grafischen 'Textbildschirm' von 80 × 25 Zeichen wäre demnach 8 × 19 Pixel. Bei einer Pixelmatrix von 8 × 8 Pixeln könnte man sogar 60 Zeilen auf den Bildschirm quetschen, was der guten Lesbarkeit aber kaum dient.

Mit 'Pixelmatrix' ist die Matrix eines einzelnen Buchstabens gemeint, wogegen sich die 'Charactermatrix' auf die Anzahl der Zeilen und Spalten des Bildschirms bei gegebener 'Pixelmatrix' bezieht.

Was fehlt dem BGI-Pixelfont?

Aus diesen Überlegungen heraus ist zu erkennen, daß man nicht einfach drauflos programmieren kann und den Benutzern dieser Programme eine Lupe mitliefert, falls diese zufällig eine VGA-Karte benutzen. Auch der freiwillige Verzicht auf die hohe VGA-Auflösung durch die Ansteuerung weniger hochauflösender Grafikmodi wird die VGA-Besitzer wenig begeistern.

Zu noch größerer Enttäuschung kommt es, wenn man die Pixelfonts im CGA-Modus benutzt, denn Umlaute werden wieder einmal nicht dargestellt. Da wir das Programm aber unverändert von einem PC mit VGA-Karte auf einen anderen PC mit EGA-Karte transportiert haben, muß die Umlautfeindlichkeit wohl im System (DOS oder BIOS) und nicht am BGI liegen.

Nach längerem Forschen fand ich dann auch im MSDOS den mir bis dahin unverstandenen DOS-Befehl GRAFTABL. Dieser Befehl löst zwar das Problem mit den Umlauten, doch steht im Handbuch leider nur, daß das System nach Benutzung dieses Befehls jetzt auch die Anzeige der ASCII-Zeichen von 128 bis 255 unterstützt, wenn der Farb/Grafikadapter im Grafikmodus verwendet wird. Kein Wort darüber, an welche Speicheradresse der Zeichensatz geladen wird, kein Wort darüber wie das System die 'Unterstützung' leistet oder ähnliches. Das einzige, was dem Leser bleibt, ist die freie Wahl, die Zeichen des oberen ASCII-Bereichs zu laden oder nicht. Da ein normaler Mensch sowieso nie mit dem DOS im Grafikmodus kommuniziert, sondern dies seinen Grafikprogrammen überläßt, so steht's in meinem Handbuch, braucht er auch diesen Befehl nicht. Jedenfalls installiert GRAFTABL die obere Hälfte des Zeichensatzes, auf dessen Speicherung im ROM ursprünglich verzichtet wurde, resident im RAM.

Das BGI und seine Pixel

All diese Experimente zeigen, daß das BGI den Pixelfont nicht intern im Code gespeichert hat, sondern diesen im System abholt. Beim Disassemblieren des BGI stößt man dann auch auf die Adressen F000h :FA6Eh und 0000h:007Ch. In der 'IBM-Grafikbibel' [1] findet man Näheres zu diesen Zahlen: ab der Adresse F000h :FA6Eh steht tatsächlich der 8×8-Pixelfont für die Zeichen 0–127. Je acht Bytes beschrei-

Zahlenmystik

Mode (Hex)	Typ	Auflösung	Pixelmatrix für 80 × 25	Pixelmatrix für 8 × 8	Charactermatrix für 8 × 14	Charactermatrix für 8 × 16
06	CGA	640 × 200	8 × 8	80 × 25	< 25	< 25
07	Herc	720 × 348	9 × 14	90 × 43	90 × 25	< 25
0E	EGA	640 × 200	8 × 8	80 × 25	< 25	< 25
0F	EGA	640 × 350	8 × 14	80 × 43	80 × 25	< 25
10	EGA	640 × 350	8 × 14	80 × 43	80 × 25	< 25
11	VGA	640 × 480	8 × 19	80 × 60	80 × 34	80 × 30
12	VGA	640 × 480	8 × 19	80 × 60	80 × 34	80 × 30

In der Tabelle sind nur die grafischen Modi verzeichnet, die 40 × 25-Textmodi sind nicht aufgeführt. Bei Auflösungen, die weniger als 25 Zeilen auf dem Bildschirm darstellen, ist die Anzahl der Zeilen mit einem '<' gekennzeichnet; nichtstandardisierte 'SuperVGA'-Auflösungen fehlen in der Tabelle ganz.

ben die Pixelmatrix eines jeden Buchstabens. Die zweite Adresse (0000h:007Ch) liegt zwar im Adreßbereich der Interrupts (Int 1Fh), sie enthält aber nur die Adresse der zweiten Hälfte des 8×8-Pixelfonts (Buchstaben 128–255), sofern diese durch den Befehl GRAFTABL geladen wurde.

Jedes Byte der Pixelfont-Codes steht für eine Pixelzeile eines Zeichens. Ein Zeichen braucht also zu seiner Beschreibung soviele Bytes, wie es Pixelzeilen hat. So findet man zum Beispiel die acht Bytes für das 'A', das den Code 41h hat,

8 × 41h = 208h Bytes

nach Beginn der Font-Tabelle (F00h:FA6h + 0208h =F00h:FC7h): 30h, 78h, CCh, CCh, FCh, CCh, CCh, 00h. Zeichnet man die Bits dieser Bytes zeilenweise auf, erhält man das Bild des Buchstabens.

```
30h  =  __XX____
78h  =  _XXXX___
CCh  =  XX__XX__
CCh  =  XX__XX__
FCh  =  XXXXXX__
CCh  =  XX__XX__
CCh  =  XX__XX__
00h  =  ________
```

Warum geht's mit WriteLn?

Überraschenderweise kann man zum Beispiel im Turbo-Pascal mit dem Befehl 'WriteLn' Buchstaben vernünftiger Größe im Grafikmodus schreiben. Dazu muß allerdings DirectVideo:= False gesetzt sein, damit das BIOS zum Schreiben verwendet wird. Offensichtlich stellt das BIOS die Anwesenheit des Grafikmodus fest und produziert seine eigenen größeren, und damit lesbaren, Pixelfonts. Je nach Grafikmodus, erscheinen 8×8-, 8×14- oder 8×16-Pixelmatrizen. Damit läßt sich in allen Fällen recht passabel schreiben.

Woher hat das BIOS nun diese Pixelfonts? Im Programmer's Guide [1] findet man, daß das EGA- und VGA-BIOS über neuere Funktionen zur Ausgabe der Adressen verfügt, an denen geeignete Fonts im EGA/VGA-BIOS stehen. Wenn das BIOS 'freundlich' war, hat es beim Einstellen des Grafikmodus den Interrupt-Vektor 43h (000h:010Ch) auf den entsprechenden Pixelfont gesetzt. Dieser hat nun 8, 14 oder 16 Punktzeilen pro Buchstabe und sieht im eingestellten Grafikmodus sehr brauchbar aus.

Wenn das BIOS nun einen Textausgabe-Befehl erhält und bemerkt, daß es sich in einem Grafikmodus befindet, greift es auf diese Adresse zu, um die Lage der Font-Tabelle im Speicher zu erfahren.

In unserer Applikation – einem BTX-Software-Dekoder – war der BGI-Treiber unverzichtbar, da dieser von der verwendeten Software für die Gestaltung der grafischen Oberfläche benutzt wurde. Die Tatsache, daß die Pixelfonts keine Proportionalschriften sind (die Pixelmatrizen aller Buchstaben haben dieselbe Größe), war wegen der einfacheren Handhabung sehr erwünscht.

Die Möglichkeit, die Ausgaberoutinen auf die entsprechenden BIOS-Calls umzulenken, wurde aus den folgenden Gründen ausgeschlossen: die BIOS-Calls schreiben die Pixel-Buchstaben nur auf ganze 8er-Vielfache. Sowohl der Vordergrund, wie auch der Hintergrund werden neu geschrieben. Die grafische Information um den Buchstaben herum wird also gelöscht. Obwohl wir in unserer Applikation dies nicht brauchten, wäre die Ausgabe um 90° gedrehter Schriften, wie dies im BGI vorgesehen ist, in vielen Fällen sehr nützlich gewesen.

Zugriff auf die Pixelfonts

Der Versuch, die jeweils in der installierten Grafikkarte residenten Pixelfonts zu benutzen, scheitert an der Vielfalt der verwendeten Fonts und der eventuell nicht richtig initialisierten Vektoren. Aus diesem Grund entschieden wir uns, Pixelfonts als Dateien abzulegen und bei Bedarf ins Programm einzulesen.

Besitzer einer VGA-Karte können sich die Pixelfont-Dateien mit wenig Aufwand mittels DOS-Debugger selbst erstellen. Diese lassen sich dann bequem als Grundlage eigener Designs verwenden.

Natürlich ist es schon verschwenderisch, sämtliche Buchstabenbeschreibungen in sämtlichen Pixelgrößen ins RAM einzulesen, wenn sie nicht alle verwendet werden. Noch speicherintensiver ist aber die simultane Unterstützung sämtlicher verbreiteter Grafikkarten. In den Borland-Grafik-Interfaces ist darum mit dem 'Registrieren'

```
 1 Program TestChar;
 2
 3 Uses Dos, Crt, Graph, GraphUti;
 4
 5 {-----------------------------------------------------------------}
 6
 7 Var
 8   x, y, i          : Integer;
 9   s                : String;
10   PixFont, a       : Word;
11
12 Begin { TestChar }
13   InitGraph(GraphDriver, GraphMode, GraphPath);
14   ClearViewport;
15   setbkcolor(blue);
16   setcolor(magenta);                    { Text normal ausgeben }
17   DirectVideo:= False; { bei TRUE wird nicht über das BIOS ausgegeben }
18   WriteLn('Hello Wörld');
19   Repeat Until ReadKey = #13;
20   { Pixel markieren }
21   PutPixel(121, 174, White);  {linke, obere Ecke des ersten Buchstaben}
22   DetectPixFont(PixFont);
23   a:= InitPixFont(PixFont);
24   If a < 0 then Halt(12);
25   SetTextStyle(0, HorizDir, 1);       { vertdir, horizdir }
26   SetTextJustify(LeftText, TopText);
27   WriteLn('1');
28   Repeat Until ReadKey = #13;
29   { go to exotic place and write }
30   Graph.OutTextXY(121, 174, 'Hello Wörld');
31   Repeat Until ReadKey = #13;
32   { now use the local one }
33   SetColor(1);
34   For i:= 1 to 25 do
35   Begin
36     OutTextXY(10, i * TextHeight('H'),
37   'Viel, viel, sehr viel, molto viel Text für den Geschwindigkeitstest');
38   End;
39   PutPixel(221, 174, White);
40   OutTextXY(221, 174, 'Hello Wörld');
41   Repeat Until ReadKey = #27;
42   CloseGraph;
43
44 End. { testchar }
```

Ein kleiner Test für die GrafUti-TPU deutet an, was durch den Gebrauch der Unit alles möglich ist.

schon vor Jahren das 'Dynamic Linking' eingeführt worden. Die Idee ist, daß man zur Laufzeit 'detektieren' kann, welche Grafikhardware vorliegt, und dann nur das Benötigte lädt.

Dank der Technik des Registrierens von BGI-Treibern und Fonts hat der Programmierer die Möglichkeit, alles in eine EXE-Datei zu verpacken. Das bedingt aber die Konversion der BGI-, CHR- und PIX-Dateien in OBJ-Dateien, die man in C mit Hilfe von BGIOBJ beziehungsweise in Turbo-Pascal mit BINOBJ erreichen kann. Dazu ruft man zum Beispiel BINOBJ auf und gibt als ersten Parameter die Quelldatei, dann die Zieldatei und als dritten Parameter den Fontnamen an. Die zweite Möglichkeit ist, daß man platzsparend zur Laufzeit nur das lädt, was benötigt wird. Bei kleinen Programmen wird man die 'kompakte Bauweise' vorziehen, größere Pakete bestehen sowieso aus einer Menge von Hilfsdateien, haben aber oft mit beschränkten RAM-Ressourcen zu kämpfen. Hier stellen wir eine Unit vor, die beide Möglichkeiten bietet.

Und so sieht's in Pascal aus ...

Die Funktionen zur Textausgabe im Grafikmodus wurden in C und in Pascal programmiert. Hier wird nur die Pascal-Version gezeigt. Die C-Variante ist auf einer der nächsten Sammeldisketten erhältlich (zusammen mit den Pascal-Listings). In Pascal gestaltete sich das Aufrufen der originalen Textausgabe-Prozeduren einfacher. Statt OldTextXY mit dem unschönen Trick via _DDO_ADD und _gr_textheight aufzurufen, kann man in Pascal einfach Graph.OutTextXY und Graph.TextHeight schreiben. Das Unit-Konzept von Turbo-Pascal erlaubt bedeutend eleganteren Umgang mit überdefinierten Bezeichnern, als das seiner Definition gemäß nicht-modulare C. Die Information über _DDO_ADD und _gr_textheight mußten wir deshalb auch mühsam dem Library-Verzeichnis der GRAPH.LIB und einigen Testläufen im Turbo-Debugger entnehmen. Ältere Ausgaben von c't zu der BGI-Struktur waren dabei sehr nützlich.

Mit der Definition der Compilersymbole BGIresident und PIXresident kann der Benutzer angeben, ob zur Compiler-Zeit oder zur Laufzeit gelinkt werden soll. Die Aufspaltung der Buchstabenausgabe in OutCharXYHor1 (horizontal, Größe 1), OutCharXYHorn (horizontal, Größe n), OutCharXYVer1 (vertikal, Größe 1) und OutCharXYVern (vertikal, Größe n) ist aus Effizienzgründen vorgenommen worden, damit möglichst viele Fallunterscheidungen aus der inneren Schleife verschwinden.

Die 'selbstgestrickte' Ausgabe der Pixelfonts ist deutlich langsamer, als die BGI-interne Routine. Unser Abschieben der Textausgabe auf die einzige BGI-Routine PutPixel hat aber den unschätzbaren Vorteil, daß das Programm unabhängig von der Grafikkarte ist. Durch Hinzukauf von BGI-Treibern für neue Grafikkarten ist das Pixelfont-Problem für diese automatisch mitgelöst. Aus diesem Grund verkneifen wir uns die mögliche, aber grafikstandard-abhängige Optimierung unserer Pixelfont-Ausgabe. Unsere grafische Oberfläche läßt sich jedenfalls mit den neuen Textausgabe-Prozeduren dazu bringen, endlich lesbare Menüs auf dem VGA-Schirm hervorzuzaubern.

Die Pascal-Unit GRAPHUTI-.PAS ist die Enkelin der entsprechenden Unit des Hershey-Font-Artikels [2]. Obwohl der Teil, der die Vektorfonts zeich-

```
C>DEBUG                                  Starten des DOS-Debuggers.

-n char8x81.pix                          Speichern der ersten Hälfte des 8 x 8-Zei-
-r cx                                    chensatzes.

CX 0000                                  Dieser hat 8*80h = 400h Bytes...
:400
-w f000:fa6e                             ... und fängt auf Adresse F00h:FA6h an.

Schreiben von 0400 Byte                  Die erste Hälfte des 8 x 8-Zeichensatzes
                                         ist unter CHAR8x81.PIX gespeichert.

-d 0000:007C 14                          Als nächstes nehmen wir die zweite Hälfte
   0000:0070  00 0C 00 C0                vor.
-n char8x82.pix
-r cx                                    Der Inhalt von Interrupt 1h führt uns auf
                                         seine Adresse.

CX 0400                                  Auch diese Hälfte ist 400h Bytes groß.
:400
-w c000:0c00
Schreiben von 0400 Byte                  Nun ist die zweite Hälfte des 8 x 8-Zei-
                                         chensatzes als CHAR8x82.PIX gespeichert.

-a
5C47:0100 mov ax,0010                    Für den 8 x 14-Zeichensatz, müssen wir in
5C47:0103 int 10                         den 350-Zeilen Grafikmodus schalten ...

5C47:0105 mov ax,1130                    ... und dann die Adresse des 8 x 14-Zeichen-
5C47:0108 mov bh,2                       satzes abfragen.
5C47:010A int 10
5C47:010C nop
5C47:010D

-g 10c                                   Nach dem Ausführen dieser Instruktionen

AX=1130  BX=0200  CX=000E  DX=0018  SP=FFEE  BP=1000  SI=0000 DI=0000
DS=5C47  ES=C000  SS=5C47  CS=5C47  IP=010C   NV UP EI PL NZ NA PO NC
5C47:010C 90            NOP
                                         ... kann man den 8x14-Zeichensatz ab der
                                         Adresse C00h:100h (=ES:BP) holen.

-n char8x14.pix
-r cx
CX 000E
:E00
-w C000:1000

Schreiben von 0E00 Byte                  Und schon ist auch der 8 x 14-Zeichensatz
                                         gespeichert.

-r ip                                    Für den 8 x 16-Zeichensatz benutzen wir
IP 010C                                  denselben Trick.
:100
-a 100
```

```
5C47:0100 mov ax,0012
5C47:0103 int 10
5C47:0105 mov ax,1130
5C47:0108 mov bh,6
5C47:010A int 10
5C47:010C nop
5C47:010D

-g 10c

AX=1130  BX=0600  CX=0010  DX=001D  SP=FFEE  BP=1F2D  SI=0000 DI=0000
DS=5C47  ES=C000  SS=5C47  CS=5C47  IP=010C   NV UP EI PL NZ NA PO NC
5C47:010C 90            NOP
                                         Die Adresse des 8 x 16-Zeichensatzes ist
                                         also C00h:1F2h.

-n char8x16.pix
-r cx
CX 0010
:1000
-w c000:1F2D

Schreiben von 1000 Byte                  Damit haben wir eigentlich alles 'im Ka-
                                         sten'.
-r ip
IP 010C
:100
-a 100
5C47:0100 mov ax,0003                    Wir schalten noch zurück in einen ver-
5C47:0103 int 10                         nünftigen Textmodus
5C47:0105 nop
5C47:0106

-g 105

AX=0003  BX=0600  CX=1000  DX=001D  SP=FFEE  BP=1F2D  SI=0000 DI=0000
DS=5C47  ES=C000  SS=5C47  CS=5C47  IP=0105   NV UP EI PL NZ NA PO NC
5C47:0105 90            NOP

-q                                       ... und verlassen den DOS-Debugger


Schließlich kombinieren wir die beiden Hälften des 8 x 8-Zeichensatzes
in einer Datei:

C>copy /b char8x81.pix+char8x82.pix char8x8.pix
CHAR8X81.PIX
CHAR8X82.PIX
        1 Datei(en) kopiert
```

Die einzelnen Zeichensätze können mit DEBUG aus dem Speicher ausgelesen und auf Platte/Disk abgespeichert werden.

net, mit Turbo-Pascal 5.5 überflüssig geworden ist, haben wir das Registrierungskonzept auch für diese Unit voll durchgezogen. Alle Hilfsdateien werden im Verzeichnis GraphPath gesucht. Dieses wird auf dasjenige Verzeichnis initialisiert, von welchem das Programm geladen wurde. Bei der Auslieferung und Installation von Programmen ist es wünschenswert, die Programmdatei und alle Hilfsdateien im selben Unterverzeichnis unterzubringen.

Halftonefonts

Nachdem ich vor einem Jahr behauptet habe, den Vektorfonts gehöre die Zukunft, mutet meine ausführliche Beschäftigung mit Pixelfonts vielleicht etwas seltsam an. Wir müssen bildschirmseitig jedenfalls noch eine Weile mit den EGA/VGA-Auflösungen rechnen. Höhere Bildschirmauflösungen benötigen mehr Speicherplatz und höhere Datenraten. Diese nehmen zwar jedes Jahr zu, die Bildschirmauflösung wächst aber nur mit der Quadratwurzel dieser Zunahme. Aus diesem Grund werden wir noch eine Weile mit Bildschirmauflösungen leben, wo die Rundungsproblematik der Vektorfonts bestehen bleibt.

Wenn sich seitens der Drucker die vektorielle Speicherung von Buchstaben (etwa für die 'skalierbaren Fonts' beim HP LaserJet III) durchgesetzt hat, so ist man bei der Bildschirmausgabe immer noch an recht unflexible Pixelfonts gebunden. Außerdem lassen sich diese nicht bequem von einer Auflösung zur nächsten transformieren, sondern müssen für jede Auflösung 'von Hand' entworfen werden.

Nun gibt es allerdings ein Mittel, die wahrgenommene Auflösung des Bildschirms zu erhöhen: man beschränkt sich nicht nur auf die Farbtöne Schwarz und Weiß, sondern nimmt auch Grautöne bei der Buchstabendarstellung hinzu. Beim 16farbigen EGA mit Standardpalette kommen etwa die Farben Weiß (15), Hellgrau (7), Dunkelgrau (8) und Schwarz (0) in Betracht. Beim groben Rastern eines hochaufgelösten Druckbuchstabens für den Bildschirm berechnet man den Prozentsatz des Bildschirmrasterquadrats, der schwarz ist, und gibt dieses Pixel mit dem entsprechenden Grauton wieder.

Diese, als Antialiasing bekannte Technik stellt zwar einige Ansprüche an Farbmischung und Skalierungstechniken, ihre Anwendung in einem VGA-Grafikmodus mit 256 Farben ist jedoch frappant: Noch nie hat man so formschöne und klar definierte Buchstaben auf einem Computerbildschirm (auch nicht bei Mac, Atari oder Amiga) gesehen! Wer schreibt uns die BGI-Erweiterung für Halftonefonts? (kp)

Literatur

[1] Richard Wilton, Programmer's Guide to PC&PS/2 Video Systems, Microsoft Press 1987

[2] Hartwig Thomas, Hershey-Font-Ästhetik, c't 1/90, S. 288 ff.


```
 1 {-------------------------------------------------------------------}
 2 { GRAPHUTI:   von H. Thomas,   Zürich 1988                          }
 3 {-------------------------------------------------------------------}
 4 { Unit zum direkten Einbinden von Treibern und Fonts in EXE-Programme }
 5 { für Hercules-, CGA- und EGA/VGA-Treiber, die mitgelieferten Fonts   }
 6 { TRIP, SANS und LITT sowie selbstgestrickter Fonts.                  }
 7 {-------------------------------------------------------------------}
 8 {$DEFINE BGIresident  }   { Nur der oder die Treiber, die mit dem }
 9 { DEFINE PIXresident  }   { $-Zeichen definiert sind, werden bei der }
10 { DEFINE LITTresident }   { Initialisierung compiliert. Dadurch kann }
11 { DEFINE SANSresident }   { das $-Zeichen quasi als Schalter benutzt }
12 { DEFINE TRIPresident }   { werden. }
13 { DEFINE GOTHresident }
14
15 Unit GraphUti;
16
17 Interface
18
19 Uses Graph, Dos;
20
21 Const
22   Pix8x8Font       = 0;
23   Pix8x14Font      = 1;
24   Pix8x16Font      = 2;
25
26 Var
27   GraphDriver, GraphMode     : Integer;
28   GraphPath                  : DirStr;
29
30 Function RegisterPixFont(PixFont: Word; ptp: Pointer): Integer;
31 Procedure DetectPixFont(Var PixFont: Word);
32 Function InitPixFont(PixFont: Word): Integer;
33 Function RegisterBGIFont(Font: Pointer): Integer;
34 Procedure SetTextStyle(Font, Direction, CharSize: Word);
35 Procedure OutTextXY(x, y: Integer; s: String); { dito for outtextxy }
36 Function TextHeight(s: String): Word;
37 Function LineY(Line: Integer): Integer;  { line = 1 .. termlines }
38 Function ColX(Col: Integer): Integer;  { col = 1 .. termcolumns }
39
40 {=================================================================}
41
42 Implementation { graphuti }
43
44 Uses Crt;
45
46 Const
47   lf          = #10;
48   cr          = #13;
49   sub         = #26;
50   esc         = #27;
51
52   TermLines   = 25;
53   TermColumns = 80;
54
55   PixMaxBytes = 32; { maximale Anzahl der Bytes für einen Buchstaben }
56
57   MaxByte     = Word($FFF0);
58   MaxWord     = $7FF0;
59
60 Type
61   { general stuff }
```

```
62   ByteArray         = Array[0..MaxByte] of Byte;
63   BytePtr           = ^ByteArray;
64   WordArray         = Array[0..MaxWord] of Word;
65   WordPtr           = ^WordArray;
66   PixTable          = Array[0..pred(256 * PixMaxBytes)] of Byte;
67   PixTPtr           = ^PixTable;
68   HersheyFontType   = Record      { Beschreibung der CHR-Files          }
69     ID    : Byte;       {  0  $2B }
70     ct    : Byte;       {  1  Anzahl der Zeichen im Font               }
71     f1    : Word;       {  2  Füllbytes $00 $00                        }
72     First : Char;       {  4  Erstes Zeichen im Font                   }
73     Offs  : Word;       {  5  Offset des Vektor-Kodes im Hershey-Font  }
74     f2    : Byte;       {  7  Füllbyte $00                             }
75     Size  : ShortInt;   {  8  normale Zeichengröße                     }
76     f3    : Byte;       {  9  Füllbyte $00                             }
77     Exc   : ShortInt;   {  A  Unterlänge (negativ)                     }
78     f4    : Byte;       {  B  Füllbyte $00                             }
79     f5    : LongInt;    {  C  Füllbyte; evtl. Faktor für Zeichengröße  }
80                         { 10  Ab hier Zeichentabelle                   }
81   End;
82   HersheyDescType   = Record
83     Count    : Byte;    { Anzahl der kodierten Zeichen }
84     FirstCh  : Char;    { Erstes Zeichen in der Tabelle }
85     ChSize   : ShortInt;{ normale Zeichengröße }
86     Excess   : ShortInt;{ Unterlänge (negativ) }
87     FontPtr  : ^HersheyFontType;
88     LkUp     : WordPtr;
89     Width    : BytePtr; { Zeiger auf die Breiten-Tabelle }
90     Data     : BytePtr; { Zeiger auf den Ursprung der Vektoren }
91   End;
92   PixelNameType    = Array[Pix8x8Font..Pix8x16Font] of NameStr;
93   HersheyNameType  = Array[TriplexFont..GothicFont] of NameStr;
94
95 { global variables }
96 Const
97   DrawColor   : Integer = White;
98   XPoints     : Integer = 8; { Zeichenbreite im Pixelfont }
99   YPoints     : Integer = 8; { Zeichenhöhe im Pixelfont }
100  PixelFont   : Word = Pix8x8Font; { aktiver Pixelfont }
101  PixelName   : PixelNameType = ('CHAR8X8','CHAR8X14','CHAR8X16');
102  PixelExt    : ExtStr = '.PIX';
103  HersheyName : HersheyNameType = ('TRIP','LITT','SANS','GOTH');
104  HersheyExt  : ExtStr = '.CHR';
105
106 Var
107   HersheyDesc : Array[TriplexFont..GothicFont] of HersheyDescType;
108   PixelDesc   : Array[Pix8x8Font..Pix8x16Font] of PixTPtr;
109
110 {-------------------------------------------------------------------}
111 { Die folgenden Treiber werden je nach Gebrauch in die EXE-Datei mit  }
112 { eingebunden. Sie sind nicht vom Benutzer aufzurufen.                }
113 {-------------------------------------------------------------------}
114 {$IFDEF BGIresident}
115 Procedure CGADriverProc; External;
116 {$L CGA.OBJ }
117 Procedure EGAVGADriverProc; External;
118 {$L EGAVGA.OBJ }
119 Procedure HERCDriverProc; External;
120 {$L HERC.OBJ }
121 {$ENDIF}
122
```

```
123 {$IFDEF LITTresident}
124 Procedure SmallFontProc; External;
125 {$L LITT.OBJ }
126 {$ENDIF}
127 {$IFDEF SANSresident}
128 Procedure SansSerifFontProc; External;
129 {$L SANS.OBJ }
130 {$ENDIF}
131 {$IFDEF TRIPresident}
132 Procedure TriplexFontProc; External;
133 {$L TRIP.OBJ }
134 {$ENDIF}
135 {$IFDEF GOTHresident}
136 Procedure GothicFontProc; External;
137 {$L GOTH.OBJ }
138 {$ENDIF}
139
140 {$IFDEF PIXresident}
141 Procedure Char8x8Proc; External;
142 {$L CHAR8X8.OBJ }
143 Procedure Char8x14Proc; External;
144 {$L CHAR8X14.OBJ }
145 Procedure Char8x16Proc; External;
146 {$L CHAR8X16.OBJ }
147 {$ENDIF}
148
149 {-----------------------------------------------------------------}
150 { Locale Prozedur zum initialisieren von GraphPath                }
151 {-----------------------------------------------------------------}
152 Procedure InitGraphPath;
153
154 Var
155   n      : NameStr;
156   e      : ExtStr;
157
158 Begin { initgraphpath }
159   FSplit(FExpand(ParamStr(0)), GraphPath, n, e);
160 End; { initgraphpath }
161
162 {-----------------------------------------------------------------}
163 { Local Prozedur zum initialisieren der Font-Beschreibung         }
164 {-----------------------------------------------------------------}
165 Procedure InitFontDesc;
166
167 Var
168   i      : Integer;
169
170 Begin { initfontdesc }
171   For i:= TriplexFont to GothicFont do HersheyDesc[i].FontPtr:= nil;
172   For i:= Pix8x8Font to Pix8x16Font do PixelDesc[i]:= nil;
173 End; { initfontdesc }
174
175 {-----------------------------------------------------------------}
176 { Analysiert den Font-Header in RegisterBGI                        }
177 {-----------------------------------------------------------------}
178 Function Scan(Ch: Char; Var Buffer; Len: Word): Word;
179
180 Begin { scan }
181   Inline(
182     $C4/$BE/Buffer/     {  les di, buffer[bp]  }
183     $8B/$86/Ch/         {  mov ax, ch[bp]      }
184     $8B/$8E/Len/        {  mov cx, len[bp]     }
185     $89/$CA/            {  mov dx, cx          }
186     $FC/                {  cld                 }
187     $F2/$AE/            {  repne scasb         }
188     $75/$03/            {  jne scanexit        }
189     $4A/                {  dec dx              }
190     $29/$CA/            {  sub dx,cx           }
191                         { scanexit:            }
192     $89/$96/Scan);      {  mov scan[bp], dx    }
193 End; { scan }
194
195 {-----------------------------------------------------------------}
196 { InitHersheyFont wird von SetTextStyle aufgerufen und registriert
197 { den Hersheyfont. Wurde er gefunden, ist Fehlercode = 0          }
198 {-----------------------------------------------------------------}
199 Function InitHersheyFont(HFont: Word): Integer;
200
201 Var
202   ret    : Integer;
203   hfptr  : BytePtr;
204   hpath  : String;
205   hfile  : File;
206
207 Begin { inithersheyfont }
208   If HersheyDesc[HFont].FontPtr = nil then
209   Begin      { File zum Öffnen vorbereiten }
210     hpath:= GraphPath;
211     If hpath <> '' then
212       If not (hpath[Length(hpath)] in [':','\']) then
213         hpath:= hpath + '\';
214     hpath:= hpath + HersheyName[HFont] + HersheyExt;
215     {$i-}
216     Assign(hfile, hpath);
217     Reset(hfile, 1);
218     If IOResult <> 0 then
219       InitHersheyFont:= GrFontNotFound
220     Else
221     Begin
222       If MaxAvail < FileSize(hfile) + 16 then
223         InitHersheyFont:= GrNoFontMem
224       Else
225       Begin
226         GetMem(hfptr, FileSize(hfile));
227         BlockRead(hfile, hfptr^, FileSize(hfile), ret);
228         If (IOResult <> 0) or (ret <> FileSize(hfile)) then
229           InitHersheyFont:= GrInvalidFont
230         Else
231           InitHersheyFont:= RegisterBGIFont(hfptr);
232         Close(hfile);
233       End;
234       {$i+}
235     End
236   End;
237 End; { inithersheyfont }
238
239 {-----------------------------------------------------------------}
240 { Diese Prozedur vermerkt den Pixelfont als verfügbar            }
241 {-----------------------------------------------------------------}
242 Function RegisterPixFont(PixFont: Word; ptp: Pointer): Integer;
243
244 Var
245   ret: Integer;
246
247 Begin { registerPIXfont }
248   PixelDesc[PixFont]:= ptp;
249   RegisterPixFont:= 0;
250   If PixFont = Pix8x8Font then
251     YPoints:= 8
252   Else
253     If PixFont = Pix8x14Font then
254       YPoints:= 14
255   Else
256     If PixFont = Pix8x16Font then
257       YPoints:= 16
258   Else
259     RegisterPixFont:= GrInvalidFontNum;
260 End; { registerPIXfont }
261
262 {-----------------------------------------------------------------}
263 { DetectPixFont erkennt, welcher Pixelfont für den aktuellen     }
264 { Grafikmodus der beste ist                                      }
265 {-----------------------------------------------------------------}
266 Procedure DetectPixFont(Var PixFont: Word);
267
268 Var
269   CharHeight  : Integer;
270
271 Begin { detectPIXfont }
272   CharHeight:= Succ(GetMaxY) div TermLines;
273   If charheight < 12 then
274     PixFont:= Pix8x8Font
275   Else
276     If CharHeight <= 15 then
277       PixFont:= Pix8x14Font
278   Else PixFont:= Pix8x16Font;
279   PixFont:= 2;
280 End; { detectPIXfont }
281
282 {-----------------------------------------------------------------}
283 { InitPixFont lädt den ausgewählten Pixelfont                    }
284 {            returns error <> 0, wenn nicht gefunden.           }
285 {-----------------------------------------------------------------}
286 Function InitPixFont(PixFont: Word): Integer;
287
288 Var
289   ret, Size   : Integer;
290   ptp         : PixtPtr;
291   PixFile     : File;
292   PixPath     : String;
293
294 Begin { initpixfont }
295   If PixFont = Pix8x8Font then
296     YPoints:= 8
297   Else
298     If PixFont = Pix8x14Font then
299       YPoints:= 14
300   Else
301     If PixFont = Pix8x16Font then
302       YPoints:= 16;
303   PixelFont:= PixFont;
304   InitPixFont:= 0;
305   PixelFont:= PixFont;
306   If PixelDesc[PixelFont] = nil then
307   Begin
308     Size := 256 * YPoints;
309     PixPath:= GraphPath;
310     If PixPath <> '' then
311       If not (PixPath[Length(PixPath)] in [':','\']) then
312         PixPath:= PixPath + '\';
313     PixPath:= PixPath + PixelName[PixelFont] + PixelExt;
314     { read the pixel font into buffer }
315     {$i-}
316     Assign(PixFile, PixPath);
317     Reset(PixFile, 1);
318     If IOResult <> 0 then
```

```
319       InitPixFont:= GrFontNotFound
320     Else
321     Begin
322       If MaxAvail < Size + 16 then
323         InitPixFont:= GrNoFontMem
324       Else
325       Begin
326         GetMem(ptp, Size);
327         BlockRead(PixFile, ptp^, Size, ret);
328         If (IOResult <> 0) or (ret <> Size) then
329           InitPixFont:= GrInvalidFont
330         Else
331           InitPixFont:= RegisterPixFont(PixelFont, ptp);
332         Close(PixFile);
333       End;
334       {$i+}
335     End
336   End;
337 End; { initpixfont }
338
339 {-----------------------------------------------------------------}
340 { RegisterBGIFont vermerkt den Hersheyfont als verfügbar          }
341 {-----------------------------------------------------------------}
342 Function RegisterBGIFont(Font: Pointer): Integer;
343
344 Const
345   HersheyNameLen   = 4;
346
347 Var
348   i, l, FontCode   : Integer;
349   FontName         : String[4];
350   p                : Pointer;
351
352 Begin { registerBGIfont }
353   l:= Scan(Sub, Font^, $100);
354   p:= Font;
355   Inc(Word(p), 3 + l);
356   Move(p^, FontName[1], HersheyNameLen);
357   FontName[0]:= Chr(HersheyNameLen);
358   FontCode:= GrInvalidFont;
359   For i:= TriplexFont to GothicFont do
360     If FontName = HersheyName[i] then
361       FontCode:= i;
362   If FontCode >= 0 then
363   With HersheyDesc[FontCode] do
364   Begin
365     Dec(Word(p), 2);
366     Move(p^, l, 2);
367     FontPtr:= Font;
368     Inc(Word(FontPtr), l);
369     With FontPtr^ do
370     Begin
371       Count:= ct;
372       FirstCh:= First;
373       Chsize:= Size;
374       Excess:= Exc;
375       l:= Offs;
376     End;
377     RegisterBGIFont:= Graph.RegisterBGIFont(Font);
378     FontPtr:= Ptr(Seg(FontPtr^), Ofs(FontPtr^) and $FFF0);
379     lkup:= WordPtr(FontPtr);
380     Inc(Word(lkup), 16);
381     Width:= @lkup^[Count];
382     Data:= BytePtr(FontPtr);
383     Inc(Word(Data), l);
384   End
385   Else
386     RegisterBGIFont:= FontCode;
387 End; { registerBGIfont }
388
389 {-----------------------------------------------------------------}
390 Procedure SetTextStyle(Font, Direction, CharSize: Word);
391
392 Var
393   gr: Integer;
394
395 Begin { settextstyle }
396   If Font = 0 then
397     gr:= InitPixFont(PixelFont)
398   Else
399     gr:= InitHersheyFont(Font);
400   If gr >= 0 then
401     Graph.SetTextStyle(Font, Direction, CharSize);
402 End; { settextstyle }
403
404 {-----------------------------------------------------------------}
405 Function TextHeight(s: String): Word;
406
407 Var
408   TextSettings: TextSettingsType;
409
410 Begin { textheight }
411   GetTextSettings(TextSettings);
412   With TextSettings do
413   If (Font = 0) and (PixelDesc[PixelFont] <> nil) then
414     TextHeight:= YPoints * CharSize
415   Else
416     TextHeight:= Graph.TextHeight(s);
```

```
417 End; { textheight }
418
419 {-----------------------------------------------------------------}
420 { OutBarXY schreibt ein Quadrat;                                  }
421 { die linke ober Ecke wird durch x,y angegeben                    }
422 {-----------------------------------------------------------------}
423 Procedure OutBarXY(x, y: Integer; BarSize: Word);
424
425 Var
426   sx, sy      : Integer;
427
428 Begin { outbarxy }
429   sy:= y;
430   For sy:= y to pred(y + BarSize) do
431     For sx:= x to pred(x + BarSize) do
432       PutPixel(sx, sy, DrawColor);
433 End; { outbarxy }
434
435 {-----------------------------------------------------------------}
436 { OutCharXYVern schreibt ein Zeichen in der gegebenen vertikalen Größe }
437 { die linke obere Ecke wird durch sx,sy angegeben                  }
438 {-----------------------------------------------------------------}
439 Procedure OutCharXYVern(x, y: Integer; Ch: Char; CharSize: Word);
440
441 Var
442   CharPtr     : ^Byte;
443   BitMask     : Byte;
444   sx, sy      : Integer;
445
446 Begin { outcharxyvern }
447   CharPtr:= Addr(PixelDesc[PixelFont]^[ord(Ch) * YPoints]);
448   sx:= x;
449   While sx < x + YPoints * CharSize do
450   Begin
451     BitMask:= $80;
452     sy:= y + pred(XPoints) * CharSize;
453     While BitMask <> 0 do
454     Begin
455       If (CharPtr^ and BitMask) <> 0 then
456         OutBarXY(sx, sy, CharSize);
457       BitMask:= BitMask shr 1;
458       Dec(sy, CharSize);
459     End;
460     Inc(sx, CharSize);
461     Inc(Word(CharPtr));
462   End;
463 End; { outcharxyvern }
464
465 {-----------------------------------------------------------------}
466 { OutCharXYHorn schreibt ein Zeichen inder gegebenen horizontalen Größe}
467 { die linke obere Ecke wird durch sx,sy angegeben                 }
468 {-----------------------------------------------------------------}
469 Procedure OutCharXYHorn(x, y: Integer; Ch: Char; CharSize: Word);
470
471 Var
472   CharPtr     : ^Byte;
473   BitMask     : Byte;
474   sx, sy      : Integer;
475
476 Begin { outcharxyhorn }
477   CharPtr:= Addr(PixelDesc[PixelFont]^[ord(Ch) * YPoints]);
478   sy:= y;
479   While sy < y + YPoints * CharSize do
480   Begin
481     BitMask:= $80;
482     sx:= x;
483     While BitMask <> 0 do
484     Begin
485       If (CharPtr^ and BitMask) <> 0 then
486         OutBarXY(sx, sy, CharSize);
487       BitMask:= BitMask shr 1;
488       Inc(sx, CharSize);
489     End;
490     Inc(sy, CharSize);
491     Inc(Word(CharPtr));
492   End;
493 End; { outcharxyhorn }
494
495 {-----------------------------------------------------------------}
496 { OutCharXYVer1 schreibt ein Zeichen mit der vertikalen Größe 1    }
497 { linke obere Ecke wird durch sx,sy angegeben                     }
498 {-----------------------------------------------------------------}
499 Procedure OutCharXYVer1(x, y: Integer; Ch: Char);
500
501 Var
502   CharPtr     : ^Byte;
503   BitMask     : Byte;
504   sx, sy      : Integer;
505
506 Begin { outcharxyver1 }
507   CharPtr:= Addr(PixelDesc[PixelFont]^[ord(Ch) * YPoints]);
508   sx:= x;
509   While sx < x + YPoints do
510   Begin
511     BitMask:= $80;
512     sy:= y + pred(XPoints);
513     While BitMask <> 0 do
514     Begin
```

```
515       If (CharPtr^ and BitMask) <> 0 then
516         PutPixel(sx, sy, DrawColor);
517       BitMask:= BitMask shr 1;
518       Dec(sy);
519     End;
520     Inc(sx);
521     Inc(Word(CharPtr));
522   End;
523 End; { outcharxyverl }
524
525 {-----------------------------------------------------------------}
526 { OutCharXYHorl schreibt ein Zeichen mit der horizontalen Größe 1      }
527 { die linke obere Ecke wird durch sx,sy angegeben                      }
528 {-----------------------------------------------------------------}
529 Procedure OutCharXYHorl(x, y: Integer; Ch: Char);
530
531 Var
532   CharPtr     : ^Byte;
533   BitMask     : Byte;
534   sx, sy      : Integer;
535
536 Begin { outcharxyhorl }
537   CharPtr:= Addr(PixelDesc[PixelFont]^[ord(Ch) * YPoints]);
538   sy:= y;
539   While sy < y + YPoints do
540   Begin
541     BitMask:= $80;
542     sx:= x;
543     While BitMask <> 0 do
544     Begin
545       If (CharPtr^ and BitMask) <> 0 then
546         PutPixel(sx, sy, DrawColor);
547       BitMask:= BitMask shr 1;
548       Inc(sx);
549     End;
550     Inc(sy);
551     Inc(Word(CharPtr));
552   End;
553 End; { outcharxyhorl }
554
555 {-----------------------------------------------------------------}
556 Procedure OutTextXY(x, y: Integer; s: String);
557
558 Type
559   DescType    = record
560                   dx, dy   : ShortInt;
561                 End;
562   DescArray   = Array[0..0] of DescType;
563
564 Var
565   d                              : ^DescArray;
566   rh,rw                          : Integer;  { real height/width }
567   sx,sy                          : Integer;  { start point for letter }
568   ox,oy                          : Integer;  { original position }
569   cx,cy                          : Integer;  { relative coordinats }
570   mx,my                          : Integer;  { move coordinates }
571   CharHeight, CharWidth, i, j    : integer;
572   LineSettings                   : LineSettingsType;
573   TextSettings                   : TextSettingsType;
574
575 Begin { outtextxy }
576   sx:= x;  { Startposition }
577   sy:= y;
578   { color for text }
579   DrawColor:= GetColor;   { Farbe für den Text }
580   GetTextSettings(TextSettings);
581   With TextSettings do
582   Begin
583     If Direction = HorizDir then
584     Begin
585       rh:= TextHeight(s);
586       rw:= TextWidth(s);
587     End
588     Else
589     Begin
590       sy:= sy + TextWidth(s) - TextWidth(s[Length(s)]);
591       rw:= TextHeight(s);
592       rh:= TextWidth(s);
593     End;
594     If Horiz = RightText then
595       sx:= sx - rw
596     Else
597       If Horiz = CenterText then
598         sx:= sx - rw div 2;
599     If Vert = BottomText then
600       sy:= sy - rh
601     Else
602       If Vert = CenterText then
603         sy:= sy - rh div 2;
604     If (Font = 0) then
605       If (PixelDesc[PixelFont] <> nil) then
606       Begin
607         CharWidth:= XPoints * CharSize;
608         If Direction = HorizDir then
609           For i:= 1 to Length(s) do
610           Begin
611             If CharSize = 1 then
612               OutCharXYHorl(sx, sy, s[i])
613             Else
614               OutCharXYHorn(sx, sy, s[i], CharSize);
615             sx:= sx + CharWidth;
616           End
617         Else
618           For i:= 1 to Length(s) do
619           Begin
620             If CharSize = 1 then
621               OutCharXYVerl(sx, sy, s[i])
622             Else
623               OutCharXYVern(sx, sy, s[i], CharSize);
624             sy:= sy - CharWidth;
625           End;
626       End
627     Else
628       Graph.OutTextXY(x, y, s)
629   End;
630 End; { outtextxy }
631
632 {-----------------------------------------------------------------}
633 Function LineY(Line: Integer): Integer;  { line = 1 .. termlines }
634
635 Begin { liney }
636   LineY:= Line * Succ(GetMaxY) div TermLines;
637 End; { liney }
638
639 {-----------------------------------------------------------------}
640 Function ColX(Col: Integer): Integer;  { col = 1 .. termcolumns }
641
642 Var
643   w : Word;
644
645 Begin { colx }
646   w:= pred(Col);
647   w:= w * Succ(GetMaxX);
648   ColX:= w div TermColumns;
649 End; { colx }
650
651 {=================================================================}
652
653 Begin { initialization graphuti }
654
655   { initialize the font descriptions }
656   InitFontDesc;
657
658 {$IFDEF BGIresident}
659   If RegisterBGIDriver(@CGADriverProc) < 0 then
660     Halt(12);
661   If RegisterBGIDriver(@EGAVGADriverProc) < 0 then
662     Halt(12);
663   If RegisterBGIDriver(@HercDriverProc) < 0 then
664     Halt(12);
665 {$ENDIF}
666 {$IFDEF LITTresident}
667   If RegisterBGIFont(@SmallFontProc) < 0 then
668     Halt(12);
669   If InitHersheyFont(SmallFont) < 0 then
670     Halt(12);
671 {$ENDIF}
672 {$IFDEF SANSresident}
673   If RegisterBGIFont(@SansSerifFontProc) < 0 then
674     Halt(12);
675   If InitHersheyFont(SansSerifFont) < 0 then
676     Halt(12);
677 {$ENDIF}
678 {$IFDEF TRIPresident}
679   If RegisterBGIFont(@TriplexFontProc) < 0 then
680     Halt(12);
681   If InitHersheyFont(TriplexFont) < 0 then
682     Halt(12);
683 {$ENDIF}
684 {$IFDEF GOTHresident}
685   If RegisterBGIFont(@GothicFontProc) < 0 then
686     Halt(12);
687   If InitHersheyFont(GothicFont) < 0 then
688     Halt(12);
689 {$ENDIF}
690 {$IFDEF PIXresident}
691   If RegisterPixFont(Pix8x8Font, @Char8x8Proc)  < 0 then
692     Halt(12);
693   If RegisterPixFont(Pix8x14Font, @Char8x14Proc) < 0 then
694     Halt(12);
695   If RegisterPixFont(Pix8x16Font, @Char8x16Proc) < 0 then
696     Halt(12);
697   If InitPixFont < 0 then
698     Halt(12);
699 {$ENDIF}
700
701   DetectGraph(GraphDriver, GraphMode);
702   InitGraphPath;
703
704 End. { graphuti }
```

Ein gutes Beispiel für die bedingte Kompilierung gibt die Initialisierungs-Routine der Unit ab. Je nach Treiber wird auch nur dieser initialisiert. ct

c't magazin für computer technik
Redaktion
K.-F. Probst
Postfach 61 04 07

D-W3000 Hannover

Deutschland

Zürich, den 21. Mai 1991

Betrifft: Artikel „Pixelfonts" im „c't", Juni 1991

Sehr geehrter Herr Probst,

bei der Lektüre meines „Pixelfont"-Artikels in der Juni-Ausgabe des „c't" ist mir eine Reihe sinnentstellender Veränderungen aufgefallen, auf die ich Sie möglichst noch vor etwaigen Leserbriefen aufmerksam machen möchte:

- In Spalte 2 würde der Satz „Die Idealgrösse einer Pixelmatrix . . ." wesentlich gewinnen, wenn dort auf die Tabelle „Zahlenmystik" verwiesen würde, die jetzt etwas verloren dasteht. Nach dem Anschluss für das „demnach" in diesem Satz suchte ich vergebens.

- In dieser Tabelle war ursprünglich unter dem Titel „Pixelmatrix" eine einzelne Spalte vorgesehen und unter dem Titel „Charactermatrix" drei. Im Schriftsatz ist eine der Spalten nach links gerutscht, so dass der ursprüngliche Sinn der Tabelle nur noch mit Mühe erraten werden kann.

- Das „Aus diesen Überlegungen" unter dem Untertitel „Was fehlt dem BGI-Pixelfont" konnte ich auch nicht so recht einordnen. Inwiefern implizieren welche Überlegungen, dass „man nicht einfach drauflos programmieren kann"?

- Die Kodierung der Pixelfonts in der ersten Spalte auf Seite 172 (von „Jedes Byte . . ." bis zur Bitmusterdarstellung des A) schliesst ein wenig unvermittelt an das Vorherige an. Eine Umrahmung in einem separaten

Kasten, wie es ursprünglich von mir vorgeschlagen war, hätte der Klarheit gedient. Andernfalls ist eine Überleitung vonnöten.

- In Spalte 2 der Seite 173 findet sich nach der Erwähnung des Interrupt 43h das erste Vorkommnis eines Fehlers vom „Global Change"-Typ, wo von den vierstelligen Hex-Zahlen die letzte Ziffer in ein kleines „h" verwandelt wurde. Vermutlich wollten Sie das „h" anhängen, um zu verdeutlichen, dass die Adressen hexadezimal gemeint sind. Unglücklicherweise ist dabei (besonders auch im Debugger-Listing) überall die letzte Stelle verloren gegangen. Das Verständnis wird damit doch recht erschwert.

- Der grössere Abstand in der Spalte 4 der Seite 173 vor „Natürlich ist . . ." hätte wohl auch durch einen Verweis zum Debugger-Listing auf der nächsten Seite gewonnen.

Obwohl ich bei der Lektüre des Artikels über diese Versehen und Unschönheiten enttäuscht war, möchte ich mit dieser Kritik nicht einfach meine ramponierte Autoreneitelkeit gegen „meinen" Redaktor ins Feld führen. Ich kann mir vorstellen, dass die endgültige Gestaltung unter Zeitdruck stand. „Global Change"-Fehler können jedem passieren. Auch das partielle Umschreiben des Anfangs stört mich im Prinzip nicht, da es aus Gründen der benötigten Artikellänge und einer redaktionell etwas veränderten Stossrichtung notwendig wurde. Ich habe bisher von der Bearbeitung durch „c't"-Redaktoren immer noch etwas dazugelernt und bin weit davon entfernt, zu glauben, ich könne oder wisse alles besser.

Da jedoch immerhin mein Name über diesem Artikel steht, wünsche ich mir für zukünftige Artikel, dass ich diese in ihrer endgültigen Version per Fax noch einmal zu sehen kriege, bevor sie in den Schriftsatz gehen. Diese Technik des „Korrekturen-Lesens", wie sie von Herrn Behr und Herrn Storm schon immer in der Korrespondenz mit mir gepflegt wurde, hat es mir bis anhin überflüssig erscheinen lassen, explizite schriftliche Bedingungen zu formulieren, unter denen ich dem „c't" meine Artikel zum Abdruck anbiete. Ich bin sicher, dass auch die Qualität Ihrer Zeitschrift und somit Ihre Leser von einem sorgfältigen Korrekturen-Lesen profitieren.

Mit freundlichen Grüssen,

Hartwig Thomas