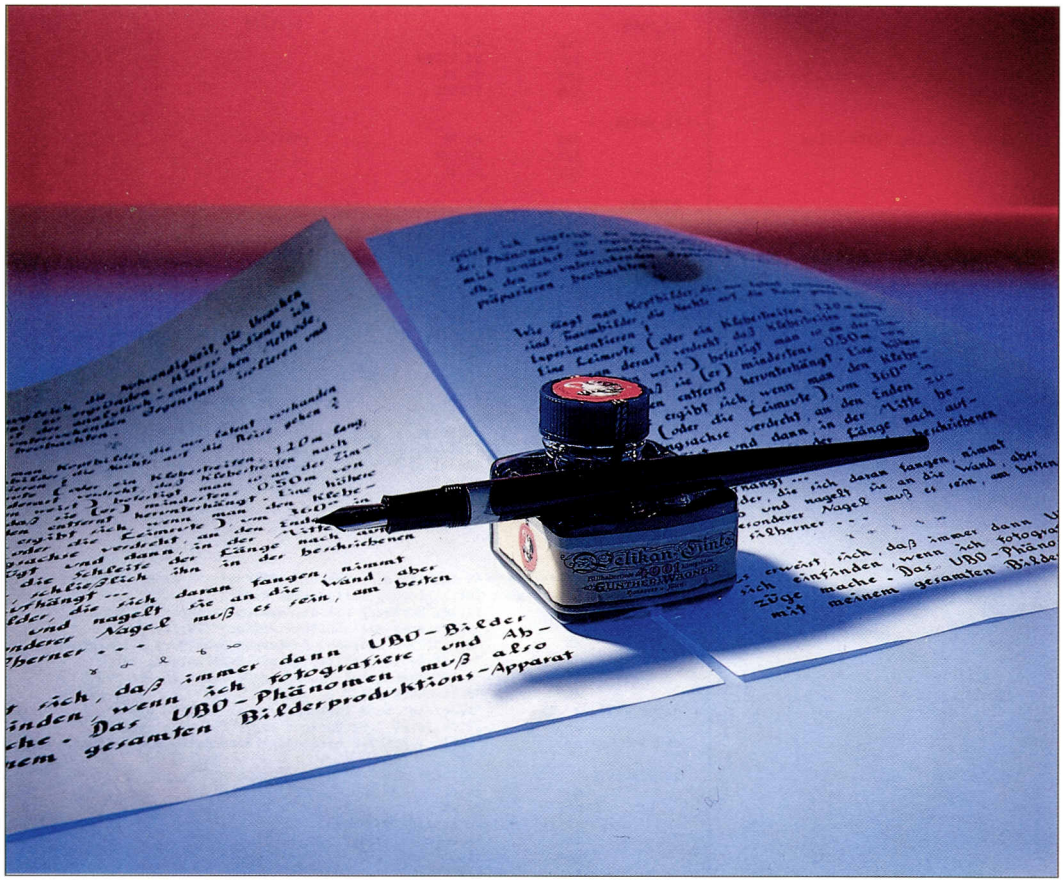


Hersheys Font-Ästhetik

Die 'stroked Fonts' in Turbo-Pascal 4/5

Dr. Hartwig Thomas

Mischen von Text und Grafik ist auf dem PC beileibe keine alltägliche Sache. Turbo-Pascal ab Version 4.0 bietet dem Programmierer in seiner GRAPHICS-Unit 'bitmapped' und Vektor-Zeichensätze an. Wer sich mit den letzteren auseinandersetzt, wird erstens auf neue Probleme und zweitens auf die Spuren von A. V. Hershey stoßen, einem Pionier der Entwicklung von Computer-Satzsystemen. Wer hätte das gedacht: Turbo-Pascals 'stroked' Zeichensätze sind mehr als zwanzig Jahre alt.

Die Integration von Text und Grafik auf dem PC würde problematisch werden. Das sahen wir alle auf den ersten Blick: Der grau-grüne Bildschirm des ersten PC-Prototyps weigerte sich standhaft, etwas anderes als Buchstaben an vorgesehenen Positionen in einer einzigen Größe und in einer einzigen Ausrichtung anzuzeigen.

Die 'Hercules-Karte' brachte dann Grafik. Seither hat IBM vier Grafikstandards eingeführt (MDA, CGA, EGA, VGA) mit zig Grafikmodi pro Standard. Seither besteht grundsätzlich die Möglichkeit für Grafik auf dem IBM PC, dafür ist das Chaos der Peripheriegeräte und Standards unübersehbar geworden. Der Programmierer, der Grafik-Anwendungen für den heutigen PC-Markt entwickelt, ist natürlich gezwungen, auf allen offiziellen und halboffiziellen Varianten etwas Annehmbares zu bieten.

Erste Hilfe ...

Und dann kam Borlands Turbo-Pascal 4.0 – gerade als ich mich um die Verwendung von PC-Grafik nicht mehr drücken konnte. Das neue Pascal hatte sogar eine recht transparente Grafik-Schnittstelle zu einer sehr starken Unit 'GRAPH', die sich ziemlich standard- und modus-unabhängig präsentierte. Für die anstehende Aufgabe ideal, wie ich glaubte.

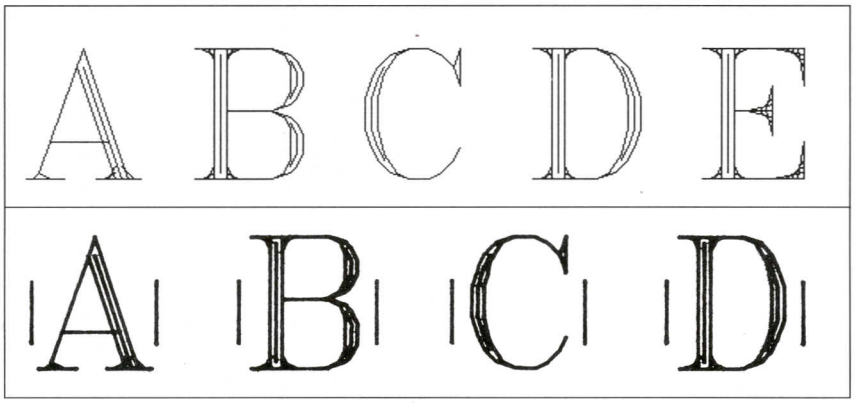
Wenn man Grafik macht, heißt dies allerdings nicht, daß nun überhaupt kein Text auf dem Bildschirm erscheinen soll. Im Grafikmodus ist aber alles anders. Für einen Macintosh-Besitzer fast schon lächerlich sind die Kapriolen, die man anstellen muß, um in einem IBM-PC-Grafikmodus Buchstaben an den Schirm zu bringen. Der Anwender muß sich schon selber seine Buchstaben pixelweise zusammenleimen. Jeglicher traditionelle Textausgabe-Befehl in irgendeiner Programmiersprache funktioniert natürlich im Grafikmodus nicht.

Turbo-Pascal 4.0 stellte dafür die speziellen Grafik-Textausgabe-Prozeduren OutText und OutTextXY zur Verfügung, zusammen mit einer Reihe von Schrifttypen-Spezifikations-Prozeduren. Statt einer einzigen Methode des Schreibens offeriert dieser Compiler deren zwei: 'bitmapped' Buchstaben oder 'stroked' Zeichensätze. Die Erklärung im dicken Turbo-Pascal-Manual meint ein bißchen kurzatmig, daß der bitmapped Zeichensatz durch eine 8 × 8-Matrix von Pixeln definiert sei und daß die stroked Fonts aus Vektoren bestehen, und verweist noch auf Vorteile der stroked Fonts beim Skalieren. Die stroked Fonts werden in vier Zeichensätzen angeboten und sind zur Freude des Ästheten, aber zum Leidwesen des Programmierers Proportionalschriften.

... nur ein Trostpflaster

Tatsächlich erwiesen sich die bitmapped Buchstaben als für CGA ausgelegt. Auf VGA sind sie kaum noch mit der Lupe zu entziffern. Ein Programm für eine breite Palette von Grafikstandards kann sie nicht brauchen.

Bei der Benutzung der stroked Fonts stellt man mit dem Groll des unterdrückten Europäers fest: wieder mal ohne Umlaute! Daß der deutsche Paragraph (ASCII-Code 21) sehr zum Leidwesen deutschsprachiger Juristen auf kaum einem Textprogramm oder Drucker problemlos produzierbar ist, daran hat man sich nachgerade gewöhnt. Daß aber noch 1988 auf dem IBM PC ein Zeichensatz eingeführt wird, der das bloße, definierte 7-Bit-Minimum des amerikanischen ASCII-Standards enthält ...!? (Erst ab Version 5.0 hatten die Entwickler ein Einsehen.)



Die Buchstaben, die Turbo-Pascal (4.0) aus dem 'stroked Font' TRIP.CHR malt (oben), ähneln den von Hershey veröffentlichten bis auf den letzten Strich.

Noch offensichtlicher wird die Ähnlichkeit beim Gothik-Font (unten wie von Hershey veröffentlicht, links eine Hardcopy von Turbo-Pascals GOTH.CHR).

Alte Bekannte

Die nähere Betrachtung der in den vier Zeichensatz-Hilfsdateien gespeicherten Buchstaben weckte meine Erinnerung: Diese Buchstaben hatte ich doch schon irgendwo einmal gesehen. Stöbern in meiner Sammlung historischer Computerliteratur förderte denn auch zutage, worum es sich handelte: Alle 'stroked Fonts' von Borland sind bis auf kleinste Modifikationen identisch mit Teilen von 'Hershey's Repertory of Occidental Type Fonts and Graphic Symbols' [1].

Der Zeichensatz in LITT.CHR entspricht dabei dem Hershey-Alphabet 'SIMPLEX ROMAN', SANS.CHR entpuppt sich als 'DUPLEX ROMAN' (eine 'Sans-Serif'-Schrift), TRIP.CHR ist natürlich 'TRIPLEX ROMAN' und GOTH.CHR schließlich entspricht 'GOTHIC ENGLISH', wovon Sie sich in den Bildern überzeugen können.

Die Hershey-Fonts wurden offenbar 1967 erarbeitet [2], wie Hersheys Veröffentlichungen vermuten lassen. Noch 1980 lernte ich sie kennen als eine der wichtigsten und schönsten Möglichkeiten, auf dem Tektronix-Speicher-Bildschirm Grafiken zu beschriften. Ihre Vorteile sind noch heute: relativ kleiner Speicherbedarf, Flexibilität in der Darstellung (Vergrößern, Verzerren, Drehen sind Vektor-Manipulationen, die für bitmapped Zeichensätze praktisch unmöglich sind), Flexibilität in der Organisation der Daten, um Modifikationen und Erweiterungen zu ermöglichen.

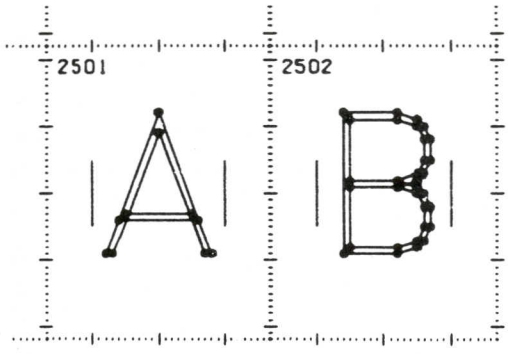
Die Vektorfonts funktionieren folgendermaßen: nach der Nummer des Buchstabens folgen zunächst der rechte und der linke Rand des Zeichens. Die folgenden Zahlenpaare bedeuten x-y-Koordinaten in einem kleinen Rechteck, bezogen auf dessen Mittelpunkt. Das Paar (–64,0) bedeutet 'pen up', das heißt: zum nächsten Zahlenpaar Cursor bewegen, ohne zu zeichnen.

Den Buchstaben 2501 (das 'A' in Duplex Roman) erhält man also unter Berücksichtigung, daß die y-Koordinate nach unten zeigt, indem man den Bleistift am Punkt (0, –12) ansetzt, zum Punkt (–8,9) einen Strich zieht, den Bleistift neu beim Punkt (0,–9) ansetzt, zum Punkt (–7,9) zeichnet, zum Punkt (–8,9) weiterzeichnet und so fort.

Das Hershey-Büchlein [1] enthält 68 engbedruckte mit Zahlen gefüllte Seiten; außerdem 65 Seiten mit grafischen Darstellungen von 1377 Buchstaben und Zeichen (inkl. mathematischer Symbole, Musiknoten, Kursivschriften, zweier griechischer Alphabete, Kyrillisch, dreier Fraktur-Alphabete, Schreibschriften und vieler Spezialzeichen).

In einer kurzen Einleitung machen die Herausgeber darauf aufmerksam, daß diese Zeichensätze ursprünglich von A. V. Hershey, einem mathematischen Physiker im Naval Weapons Laboratory, entwickelt wurden und daß sie beim National Technical Information Service auch auf Band im BCD- oder ASCII-Format bestellt werden können.

Die darauf folgende Diskussion der vielfältigen Verwendungsmöglichkeiten der Vektorfonts für Computersatz mit Plottern und Speicher-Bildschirmen stimmt beim heutigen Durchblättern nostalgisch. Obwohl vieles noch heute Gültigkeit hat, was von den Herausgebern



Hershey hatte seine Buchstaben in Rechtecken angeordnet mit dem Koordinatenursprung in der Mitte.

Die Hershey-Alphabete bestehen aus langen Zahlenkolonnen von x/y-Koordinaten.

```
2501  :-10  10:  0 -12: -8   9:-64   0:  0  -9: -7   9: -8   9:-64   0:  0  -9:  7   9:  8   9:-64   0:  0 -12:  8   9:-64   0:
      : -5   3:  5   3:-64   0: -6   4:  6   4:-64 -64:
2502  :-10  10: -6 -12: -6   9:-64   0: -5 -11: -5   8:-64   0: -6 -12:  2 -12:  5 -11:  6 -10:  7  -8:  7  -5:  6  -3:  5  -2:
      :  2  -1:-64   0: -5 -11:  2 -11:  5 -10:  6  -8:  6  -5:  5  -3:  2  -2:-64   0: -5  -2:  2  -2:  5  -1:  6   0:  7   2:
      :  7   5:  6   7:  5   8:  2   9: -6   9:-64   0: -5  -1:  2  -1:  5   0:  6   2:  6   5:  5   7:  2   8: -5   8:-64 -64:
```

```
-d aF0
226C:0AF0  8F 86 90 85 9A 00 00 00-88 15 80 80 81 80 88 92
226C:0B00  8F 80 90 80 88 95 83 06-8D 86 82 05 8E 85 95 00
226C:0B10  00 00 80 15 80 80 81 14-81 81 80 15 88 95 8B 94
```

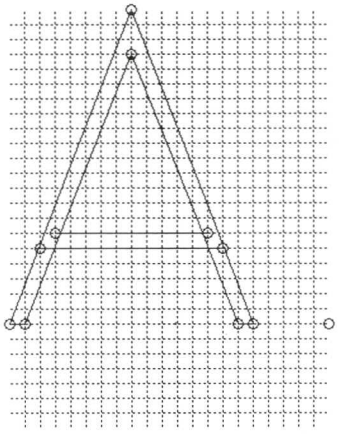
Auch in den .CHR-Daten findet man die Koordinatenpaare, hier das 'A' in SANS.CHR.

vorgebracht wird, leben wir doch heute im Zeitalter der Raster-Bildschirme und -Drucker, wo die Vektor-Peripheriegeräte endgültig obsolet geworden sind. Selbst ein so exzellentes Buch wie 'Principles of Interactive Computer Graphics' von Newmann und Sproull [3] leidet unter dem an Vektorgeräte verschwendeten Platz.

Gerade dort findet man aber eine der klarsten Darlegungen der Vektor-zu-Raster-Konversions-Algorithmen, welche die Arbeit mit Vektor-Datenstrukturen von der Bindung an Vektor-Geräte befreit haben. Heute ist uns die Trennung zwischen elektronischer Bildverarbeitung (Datenstruktur: Pixel-Matrix) und Computer-Grafik (Datenstruktur: Vektoren, d. h. Linien definiert durch Anfangspunkt und Endpunkt meistens in Gleitkomma-Koordinaten) vertraut und problemlos. Offensichtlich verwenden die Turbo-Pascal-Prozeduren intern einen solchen Algorithmus, um die Vektor-Buchstaben zu zeichnen.

Copy – right?

Mit Hilfe der Hershey-Liste seiner Zeichenvektoren war die Aufschlüsselung der CHR-Dateien nicht allzu schwierig. Der pompöse Copyright-Vermerk am Beginn der Dateien (etwa: 'Sans Serif font V100 – 19 October 1987 Copyright (c) 1987 Borland International') überrascht zwar etwas in Hinblick auf die Bemerkungen in der Chamber-of-Commerce-Publikation, wo Hersheys Großzügigkeit gepriesen wird, der die Früchte seiner Arbeit anderen zugänglich mache. Möglicherweise hat Borland die Rechte von Hersheys Arbeitgeber, der Regierung der USA, abgekauft, die ja wohl die Rechte an den Lizenzen besitzt. Das Autorenrecht der Namensnennung hätte man ihm allerdings lassen dürfen.



Diese Figur erhält man, wenn man mit einem Stift den Angaben der Zahlenpaare für das 'A' in SANS.CHR folgt.

Im Hexdump-Ausschnitt von SANS.CHR findet sich wieder der Buchstabe 'A', ab Adresse 0AF8h. Dasselbe Bild wie bei Hershey ist hier leicht abweichend kodiert: Je zwei aufeinanderfolgende Bytes stellen ein x/y-Koordinatenpaar dar. Das Koordinaten-Rechteck ist im positiven Quadranten angesiedelt, statt symmetrisch um Null. Wenn das höchste Bit in der x-Koordinate 0 ist, bedeutet dies das Buchstaben-Ende. Hier ist immer ein x/y-Paar zu Null gesetzt.

Wenn das höchste Bit in der y-Koordinate, die diesmal nach oben zeigt, gesetzt ist (>= 80h), wird gezeichnet ('pen down'), sonst wird nur der Cursor bewegt ('pen up'). Die mysteriösen Bytes bedeuten also: Gehe zu (8,21), zeichne bis (0,0), zeichne bis (1,0), zeichne bis (8,18), zeichne bis (15,0) und so weiter.

Nun war mit ein bißchen Raten und Pröbeln noch der Header der CHR-Dateien zu entziffern; er enthält – wie bei Font-Headern üblich – unter anderem einige Offset-Tabellen (die genaue Struktur ist am einfachsten dem Listing zu entnehmen). Mit diesem Wissen fällt es nicht schwer, die CHR-Dateien sinngemäß für Buchstaben mit Code größer als 127 zu erweitern. Für Umlaute etwa kopiert man einfach die Vektoren des Buchstabens ohne Akzent und pflanzt dann noch ein paar Vektoren für die Pünktchen oben drauf. Die Hoffnung, daß damit das Umlautproblem im Turbo-Pascal gelöst ist, wird aber noch ein letztes Mal enttäuscht.

Die eingebauten Turbo-Pascal-Prozeduren weigern sich leider, Buchstaben mit ASCII-Code größer als 127 zu verarbeiten. In einer kleinen Unit GRAPHUTI habe ich deshalb einerseits die wichtigsten Bildschirm-Treiber dazugelinkt und gleichzeitig die GRAPH-Prozeduren OutText und OutTextXY überdefiniert (siehe Listing).

Diese Variante kann sowohl original Borland-CHR-Dateien als auch neue, erweiterte auf den Bildschirm schreiben. Sie dokumentiert zugleich den Aufbau der CHR-Dateien für den Programmierer. Mit einem kleinen CHR-Compiler gelang es dann schließlich, wenigstens für LITT.CHR den gesamten IBM-Zeichensatz (Codes 32 bis 254) auch der grafischen Textausgabe zugänglich zu machen: damit kann man endlich auch Wörter mit Umlauten oder Cedille und Spezialzeichen wie 'µg' schreiben.

Beim Versuch, dasselbe für die anderen Zeichensätze durchzuführen, wuchs der Respekt vor A. V. Hershey. Der Aufwand der ästhetischen Buchstabenge-

EINSTEIN: Zur einheitlichen Feldtheorie

der Divergenzbildungen bei Tensordichten von beliebigem Range:

$$\mathfrak{A}_{\cdot}^{\cdot ik}{}_{/i/k} - \mathfrak{A}_{\cdot}^{\cdot ik}{}_{/k/i} \equiv -\left(\mathfrak{A}_{\cdot}^{\cdot ik}\Lambda^{\sigma}_{ik}\right)_{/\sigma} \tag{5}$$

Die Punkte bei $\mathfrak{A}$ bedeuten beliebige Indizes, die in allen drei Gliedern der Gleichung dieselben sind, nämlich diejenigen, welche bei den Divergenzbildungen nicht betroffen werden.

Der Beweis von (5) stützt sich ausser auf die Definitionsformel

$$\mathfrak{A}^{\sigma\cdots i}_{\tau\cdots}{}_{/i} = \mathfrak{A}^{\sigma\cdots i}_{\tau\cdots}{}_{,i} + \mathfrak{A}^{\alpha\cdots i}_{\tau\cdots}\Delta^{\sigma}_{\alpha i}\cdots - \mathfrak{A}^{\sigma\cdots i}_{\alpha\cdots}\Delta^{\alpha}_{\sigma i}\cdots \tag{6}$$

insbesondere auf die Identität (2). Gleichung (5) hängt eng zusammen mit dem Vertauschungsgesetz der kovarianten Differentiation, das ich der Vollständigkeit halber ebenfalls angeben will. Sei T ein beliebiger Tensor, dessen Indizes ich der Bequemlichkeit halber weglasse, so gilt

$$T_{;i;k} - T_{;k;i} \equiv -T_{;\sigma}\Lambda^{\sigma}_{ik} \tag{7}$$

Von der Identität (5) machen wir nun Anwendung auf die Tensordichte $\mathfrak{B}^{\alpha}_{\underline{kl}}$ deren untere Indizes wir heraufgezogen denken. Wir finden so als einzige nicht triviale Identität

$$\mathfrak{B}^{\alpha}_{\underline{kl}/l/\alpha} - \mathfrak{B}^{\alpha}_{\underline{kl}/\alpha/l} \equiv -\left(\mathfrak{B}^{\alpha}_{\underline{kl}}\Lambda^{\sigma}_{l\alpha}\right)_{/\sigma}$$

welche man mit Rücksicht auf (3b) auf die Form bringen kann

$$\left(\mathfrak{B}^{\alpha}_{\underline{kl}/l} - \mathfrak{B}^{\sigma}_{\underline{k\tau}}\Lambda^{\alpha}_{\sigma\tau}\right)_{/\alpha} \equiv 0 \tag{8}$$

Ein Beispiel der Hershey-Ästhetik aus [1] (verkleinerter Plotterausdruck)

staltung steigt mit der Komplexität der Zeichen ins Unermeßliche. Sollten c't-Leser am Buchstabendesign interessiert sein, stelle ich gerne den CHR-Compiler zur Verfügung im Austausch gegen die Zeichen, die der Betreffende damit kreiert.

Pixel oder Vektor

Die Vektor-Alphabete der Ursprungszeit der Computer-Typografie schienen, von der ersten PC-Welle überspült, mit den Vektor-Schirmen untergegangen zu sein. Die neuen Rastergeräte benötigten Raster-Alphabete, ihre recht bescheidene Auflösung machte die Rasterdarstellung von Vektor-Alphabeten häßlich. Nur Donald Knuth entwickelte die Vektor-Idee für Alphabete in Form seiner Spline-Darstellung für Buchstaben in METAFONT weiter. Als echter Mathematiker arbeitete er ohne Ansehung der real existierenden Drucker, deren TeX-Seiten anfangs recht bescheidene Buchstabenqualität aufwiesen.

Mit der inflationären Proliferation der Grafikstandards und dem Aufkommen hochauflösender Drucker stellt sich das Zeichensatzproblem für das Desktop Publishing seit kurzem als Pferdefuß heraus: Um auf verschiedensten Geräten nach den strengen Gesetzen der Göttin WYSIWYG ungefähr ähnlich aussehende Seiten darstellen zu können, braucht man für jede Schriftgröße und jeden Schriftstil und jedes Darstellungsgerät einen bitmapped Zeichensatz.

Für die Hersteller solcher Systeme bedeutet das die Produktion riesiger Mengen von Buchstaben-Pixelmatrizen, an die hohe ästhetische Anforderungen gestellt sind. Für den Benutzer ist die Folge, daß er spielend 20 Megabyte auf seiner kostbaren Festplatte für Zeichensatz-Dateien opfern muß, die als 'read only'-Material einen so teuren Speicherplatz eigentlich nicht verdienen.

Die neuesten Softfonts von Hewlett-Packard (etwa 'Premier Collection') gehen da wie Borlands Turbo-Pascal neue Wege: Auf der wieder handlichen Diskette (statt deren 20) sind die Zeichensätze in Vektorform (oder gar Splines?) in Umrissen gespeichert. Die Herstellung beliebiger Schriftgrößen für beliebige Ausgabegeräte mit beliebigen Zeichensatztabellen wird dann durch ein 'Installationsprogramm' für eine Reihe von wichtigen Programmprodukten vorgenommen. Solange diese immer noch pixelorientiert arbeiten, läßt sich der vergeudete Platz auf der Festplatte nicht ganz vermeiden.

Der Wiederentdeckung der Vektorfonts auf dem PC-Markt steht also nichts mehr im Wege. Alte Pioniere wie A. V. Hershey, der Trendsetter auf dem Gebiet der Computer-Typographie-Qualität, sollten dabei nicht ganz in Vergessenheit geraten. (bb)

Literatur

[1] NBS Special Publication 424: A Contribution to Computer Typesetting Techniques, Hg.: Norman M. Wolcott, Joseph Hilsenrath, National Bureau of Standards, Washington, April 1976

[2] Einen Überblick über die Entwicklung der Hershey-Fonts entnimmt man folgender Literaturliste:

A. V. Hershey, Calligraphy for Computers, Technical Report No. 2101 (1. Aug. 1967), U.S. Naval Weapons Laboratory, Dahlgren, Virginia 22448

A. V. Hershey, FORTRAN IV Programming for Cartography and Typography, NWL Technical Report TR-2339 (Sept. 1969), U.S. Naval Weapons Laboratory, Dahlgren, Virginia 22448

A. V. Hershey, Preparation of Reports with the FORTRAN Typographic System, NWL Technical Note TN-K/27-70 (Sept. 1970, U.S. Naval Weapons Laboratory, Dahlgren, Virginia 22448

A. V. Hershey, A Computer System for Scientific Typography, in Computer Graphics and Image Processing, Vol 1, pp 373–385 (1972)

[3] William M. Newman und Robert F. Sproull, Principles of Interactive Computer Graphics, McGraw-Hill, 1979 ISBN: 0-07-066455-2


```
{ GRAPHUTI:    Adapted from DRIVERS.PAS and FONTS.PAS                    }
{              by H. Thomas, (c) Enter AG, Zürich 1988                   }
{-------------------------------------------------------------------------}
{ Unit to use for direct linking of Drivers and fonts into EXEs          }
{ only links CGA, EGA/VGA and Hercules Drivers,                           }
{ TRIP, SANS and LITT fonts and implements another outtext-procedure      }
{ for Font-Extensions as well as automatic detection of graphmode.        }
{-------------------------------------------------------------------------}
unit graphuti;

interface

uses graph, crt;

var                                  { use these for init, }
  graphdriver, graphmode: integer;   { set of graphmode    }

{-------------------------------------------------------------------------}
{ The following drivers are linked into the EXE that uses this unit.      }
{ They are not to be called by the user.                                  }
{-------------------------------------------------------------------------}
procedure CgaDriverProc;
procedure EgaVgaDriverProc;
procedure HercDriverProc;
procedure SansSerifFontProc;
procedure SmallFontProc;
procedure TriplexFontProc;

{-------------------------------------------------------------------------}
{ The following two procedures overdefine the corresponding GRAPH-procedures.}
{ They are 100% identical to their originals except for allowing          }
{ Enter-compiled enhanced character sets for stroked (Hershey-) Fonts     }
{-------------------------------------------------------------------------}
procedure outtext(s: string);  { redefines outtext for full char set }

procedure outtextxy(x,y: integer;s: string);  { dito for outtextxy   }

{-------------------------------------------------------------------------}
{ The following two procedures overdefine the corresponding GRAPH-procedures.}
{ The original procedures do not handle Thickness correctly, but "line" does.}
{-------------------------------------------------------------------------}
procedure lineto(x,y: integer);

procedure linerel(dx,dy: integer);

{-------------------------------------------------------------------------}
implementation

const
  lf  = #10;
  cr  = #13;
  esc = #27;

type
  bytearray = array[0 .. 0] of byte;
  wordarray = array[0 .. 0] of word;
  fonttype = record case boolean of
    false:
      (id    : byte;          { $2B }
       ct    : byte;          { evtl. word ?, counts the number of chars      }
                              { in font and lookup table                      }
       f1    : word;          { unidentified filler $00 $00                   }
       first : char;          { first character in font                       }
       offs  : word;          { offset of vector codes from start of fonttype }
       f2    : byte;          { unidentified filler $00                       }
       size  : shortint;      { normal character size from baseline           }
       f3    : byte;          { unidentified filler $00                       }
       exc   : shortint;      { (negative) excess below baseline              }
       f4    : byte;          { unidentified filler $00                       }
       f5    : longint;       { filler evtl. for charsiz factors ?            }
       lkup  : wordarray);    { lookup table for addresses of vector desc.    }
    true:
      (buffer: bytearray);
  end;

var
  fontdesc : array[1 .. 3] of record
    fontptr: ^fonttype;
    width  : ^bytearray;
    data   : ^bytearray;      { points to origin (offs from start) of vectors }
  end;

{-------------------------------------------------------------------------}
procedure initfonts;
var
  p: ^bytearray;
  i: integer;
  ch: char;

  {-----------------------------------------------------}
  procedure setptr(i: integer);
  var
    j : integer;
  begin
    with fontdesc[i] do begin
      j := 0;
```

```
      while p^[j] <> $1A do inc(j);
      inc(j);
      move(p^[j],j,2);            { get header length }
      fontptr := @p^[j]; { set fontptr to point after header }
      if fontptr^.id <> $2B then begin
        writeln('Invalid Font found');
        halt(1);
      end;
    end;
  end; { setptr }
  {-------------------------------------------------------}

begin
  p := @TriplexFontProc;
  setptr(1);
  p := @SmallFontProc;
  setptr(2);
  p := @SansSerifFontProc;
  setptr(3);
  for i := 1 to 3 do with fontdesc[i] do with fontptr^ do begin
    width := @lkup[ct];
    data := @buffer[offs];
  end;
end;

{-----------------------------------------------------------------------------}
procedure Abort(Msg : string);
begin
  Writeln(Msg, ': ', GraphErrorMsg(GraphResult));
  Halt(1);
end;

{-----------------------------------------------------------------------------}
procedure CgaDriverProc; external;
{$L CGA.OBJ }
procedure EgaVgaDriverProc; external;
{$L EGAVGA.OBJ }
procedure HercDriverProc; external;
{$L HERC.OBJ }
procedure SansSerifFontProc; external;
{$L SANS.OBJ }
procedure SmallFontProc; external;
{$L LITT.OBJ }
procedure TriplexFontProc; external;
{$L TRIP.OBJ }

{-----------------------------------------------------------------------------}
procedure outtext{s: string)};

type
  desctype = record
    dx,dy: shortint;
  end;
  descarray = array[0 .. 0] of desctype;

var
  ox,oy,  { original position }
  nh,nw,  { norm height/width }
  rh,rw,  { real height/width }
  sx,sy,  { start point for letter }
  mx,my,
  x,y,
  i,j: integer;
  t: textsettingstype;
  l: linesettingstype;
  d: ^descarray;

begin
  gettextsettings(t);
  with t do if (font < 1) or (font > 3)
  then graph.outtext(s)                          { can't handle other fonts }
  else with fontdesc[font] do begin
    getlinesettings(l);
    ox := getx;
    oy := gety;
    setlinestyle(solidln,0,normwidth);
    with fontptr^ do for i := 1 to length(s) do  { otherwise textwidth undef }
      if (s[i] < first) or (s[i] >= chr(ord(first)+ct)) then s[i] := ' ';
    rh := textheight(s); { real height of all letters }
    { treat text adjustment }
    if (horiz <> lefttext) or (vert <> bottomtext) then begin
      rw := textwidth(s);
      if horiz = righttext then moverel(-rw,0)
      else if horiz = centertext then moverel(-(rw div 2),0);
      if vert = toptext then moverel(0,rh)
      else if vert = centertext then moverel(0,(rh div 2));
    end;
    with fontptr^ do begin
      nh := size-exc;   { normal height of all letters }
      for i := 1 to length(s) do begin
        j := ord(s[i]) - ord(first);
        nw := width^[j];   { normal width in table }
        rw := textwidth(s[i]);       { real width }
        j := lkup[j];
        d := @data^[j];
        sx := getx; sy := gety;
        j := 0;
        while (d^[j].dx and $80) <> 0 do with d^[j] do begin
          { extract x,y from description }
          x := dx and $7F;
          if (dy and $40) <> 0 then y := dy or $FF80
            else y := dy and $7F;
          { scale for current char size settings }
          x := x*rw div nw;
          y := y*rh div nh;
          { turn around according to direction of writing }
          if direction = horizdir then begin
            mx := sx + x;
            my := sy - y;
          end
          else begin
            mx := sx - y;
            my := sy - x;
          end;
          { now move or draw according to m/d-bit }
          if (dy and $80) = 0
          then moveto(mx,my)
          else lineto(mx,my);
          { next vector }
          inc(j);
        end;  { vector loop over j }
      end; { character loop over i }
      { restore line style }
      with l do setlinestyle(linestyle,pattern,thickness);
      { evtl. restore original position }
      with t do if (horiz <> lefttext) or (direction <> horizdir) then
        moveto(ox,oy)
      else if vert = toptext then moverel(0,-rh)
      else if vert = centertext then moverel(0,-(rh div 2));
    end;           { with fontptr^ }
  end;    { if it is a linked font }
end; { outtext }

{-----------------------------------------------------------------------------}
procedure outtextxy{(x,y: integer; s: string)};
var
  ox,oy: integer;
begin
  ox := getx;
  oy := gety;
  moveto(x,y);
  outtext(s);
  moveto(ox,oy);
end; { outtextxy }

{-----------------------------------------------------------------------------}
procedure lineto{(x,y: integer)};
begin
  line(getx,gety,x,y);
  moveto(x,y);
end; { lineto }

{-----------------------------------------------------------------------------}
procedure linerel{(dx,dy: integer)};
begin
  line(getx,gety,getx+dx,gety+dy);
  moverel(dx,dy);
end;

{-----------------------------------------------------------------------------}
begin

  if RegisterBGIdriver(@CGADriverProc) < 0 then
    Abort('CGA');

  if RegisterBGIdriver(@EGAVGADriverProc) < 0 then
    Abort('EGA/VGA');

  if RegisterBGIdriver(@HercDriverProc) < 0 then
    Abort('Herc');

  if RegisterBGIfont(@SansSerifFontProc) < 0 then
    Abort('SansSerif');

  if RegisterBGIfont(@SmallFontProc) < 0 then
    Abort('Small');

  if RegisterBGIfont(@TriplexFontProc) < 0 then
    Abort('Triplex');

  detectgraph(graphdriver,graphmode);

  initfonts;

end.
```

Damit die Grafik-Prozeduren OutText und OutTextXY auch Zeichen größer 127 ausgeben, werden sie hier für Turbo-Pascal 4.0 überdefiniert.