Ingo Eickmann

Grafik im Textmodus

Grafik und Text mischen

Bei CGA- und Hercules-Videoadaptern beschränkt sich die Darstellung von Grafiken im Textmodus auf die Grafiksymbole des erweiterten ASCII-Zeichensatzes. EGA- und VGA-Farbkarten bieten hier mehr: Hochauflösende Grafikfenster lassen sich im Textmodus darstellen.

Das Mischen von Text und Grafik ist seit jeher ein schwieriges Problem, sowohl bei der Software-, als auch bei der Hardwareentwicklung von Videoadaptern. Textdarstellung erfordert lediglich die Speicherung eines Character Code pro Zeichen, das aus einem Zeichengenerator die Matrix auswählt, die angezeigt wird. Grafikmodi mit Einzelpunktdarstellung verlangen hingegen Bildschirmspeicher, deren Seiten aus mindestens ebenso vielen Bit bestehen, wie auf einer Bildschirmseite Punkte angezeigt werden können. Bei Farbgrafikadaptern müssen insgesamt n solcher Bildschirmseiten parallel liegen, um 2^n verschiedene Farben gleichzeitig darzustellen.
Da die Informationsdichte einer Bildseite um mehrere Größenordnungen höher als bei einer Textseite ist, benötigt der Grafikmodus auch dementsprechend mehr Videospeicher. Bildschirmoperationen, wie die Ausgabe eines Zeichens oder das Scrollen, verlaufen viel langsamer als im Textmodus, da sie mehr Speicherzugriffe erfordern. Beim Scrollen beispielsweise kann durch das Lesen und Schreiben eines Wortes ein komplettes Zeichen im Textmodus verschoben werden. In einem Grafikmodus dagegen müssen alle Bits der Matrix bewegt werden, um den gleichen Teil des Bildschirms zu bewegen.
Besonders für Programme, die textorientiert sind und fertige Grafiken in den Text integrieren wollen, ist die Darstellung von Grafikfenstern im Textmodus interessant. Der Vorteil der hohen Verarbeitungsgeschwindigkeit im Textmodus wird mit der Darstellbarkeit von Grafik kombiniert. EGA- und VGA-Karten verwenden zur Darstellung von ASCII-Symbolen RAM-residente Zeichengeneratoren, sogenannte Character Definition Tables. Hier sind in Map 2 des Videospeichers die Bitmasken aller Zeichen abgelegt. Sie bestimmen das Aussehen des Zeichens auf Pixelebene in der Auflösung, die der angewählte Textmodus vorsieht. Im Textmodus 3 (80x25 Color) einer VGA-Karte ist die Bitmaske für jedes Zeichen 8x16 Bit groß. Wer mehr über die Zeichengeneratoren im RAM und deren Programmierung wissen will, sollte in [1] und [2] nachlesen.
Die Voraussetzungen für das Setzen und Rücksetzen einzelner Bits im Textmodus sind also gegeben: Das Erscheinungsbild einzelner Zeichen kann durch Veränderung des Character Definition Table auf Pixelebene bestimmt werden. Wie werden nun aus frei definierbaren Zeichen hochauflösende Grafikfenster?

Fenster für Grafik

Von einem Grafikfenster fordern wir, daß alle Pixel einzeln und unabhängig voneinander gesetzt und zurückgesetzt werden können. Hierzu muß der Bereich des Grafikfensters im Bildschirmspeicher mit unterschiedlichen Character Codes gefüllt werden, die von jetzt an nicht mehr ihre ursprünglich zugedachten ASCII-Zeichen darstellen *(Bild 1).* Ihre Bitmasken werden im Zeichengenerator gelöscht und stehen jetzt ausschließlich der Grafikprogrammierung zur Verfügung. Den hiermit verbundenen Nachteil der Grafikfenster im Textmodus haben viele sicher längst bemerkt: Für jedes Zeichen auf dem Bildschirm, das jetzt zum Grafikfenster gehört, muß ein Character Code geopfert werden, der nicht mehr das ursprüngliche ASCII-Zeichen darstellen kann. Das erklärt auch, warum wir grundsätzlich nur von Grafikfenstern sprechen, die Größe des so nutzbaren Bildschirmbereichs ist durch die Anzahl der normalerweise verfügbaren 256 Character Codes beschränkt. Diese Einschränkung läßt sich etwas lindern, indem man oft wiederkehrende Bitmasken, zum Beispiel Randsymbole, nur einmal definiert und den zugehörigen Character Code im Bildschirmspeicher mehrfach verwendet. Für die freie Programmierung des Grafikbereichs ist jedoch eine systematische Belegung des Videospeichers mit fortlaufenden Character Codes sinnvoll, wie in Bild 1 für ein 4x3 Zeichen großes Fenster. Die Character Codes 192 bis 203 können jetzt nur noch für das Grafikfenster verwendet werden. Die EGA- und VGA-Grafikadapter bieten die Möglichkeit, 512 verschiedene Zeichen gleichzeitig darzustellen. Die Unterscheidung der zwei jeweils 256 Character großen Zeichensätze erfolgt, wie in [2] beschrieben, anhand des Intensity/Character Select Bit (Bit 3 im Attribute Byte) jedes dargestellten Zeichens. Wenn man sich darauf beschränkt, Zeichen lediglich in einer Intensitätsstufe auszugeben, stehen bei der Verwendung von zwei Zeichensätzen alle 256 ASCII-Zeichen und zusätzlich ein 256 Character großes Grafikfenster gleichzeitig zur Verfügung. Diese Konfiguration hat sich im praktischen Einsatz als sehr vorteilhaft erwiesen, da alle ASCII-Zeichen weiterhin eingesetzt werden können. Für die Zeichen des Grafikfensters können - wie bei einem normalen Zeichen im Textmodus auch - eine Vorder- und eine Hintergrundfarbe gewählt werden.

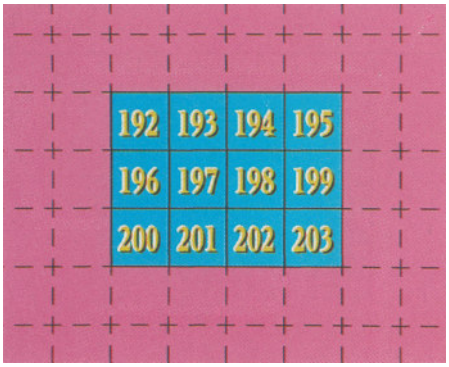

Bild 1. Der Bereich des Grafikfensters muß im Bildschirmspeicher mit fortlaufenden Character Codes beschrieben werden

Fenster für Turbo Pascal

Die Listings in *Bild 2* und *3* zeigen die Implementierung von Grafikfenstern für Turbo Pascal, basierend auf der Einteilung in 256 ASCII-Zeichen einer Helligkeitsstufe und 256 Character Codes für Grafikfenster. Bild 2 enthält die Assemblermodule, die Unit in Bild 3 bindet diese Module ein und bietet dem Pascal-Programmierer eine kleine Library mit den wichtigsten Routinen für den Einsatz von Grafikfenstern. Diese Library enthält folgende Befehle:

GWindowOn:
Die CPU erhält Zugriff auf die Zeichengeneratoren in Map 2 und kann nun die Befehle *GWindowClear, GWindowSet* und *GWindowReset* ausführen. Die Ausgabe von Zeichen auf dem Bildschirm ist nicht möglich und muß unbedingt verhindert werden.

GWindowOff:
Die CPU kann nun wieder auf den normalen Bildschirmspeicher zugreifen, Zeichenausgaben sind wieder möglich. Dagegen können die Befehle *GWindowClear, GWindowSet* und *GWindowReset* nicht ausgeführt werden.

GWindowInit(X1,Y1,X2,Y2,Golf,Colb):
Ein Grafikfenster wird durch 2 beliebige diagonalliegende Eckpunkte (X1,Y1) und (X2,Y2) definiert. Die Koordinaten beziehen sich auf die Textspalten (X = 0..79) und Textzeilen (Y = 0..24). Das Fenster erhält die Hintergrundfarbe *Colb,* alle gesetzten Pixel erscheinen in der Vordergrundfarbe *Colf. GWindowInit* ist die einzige Funktion in der Library. Ihr Ergebnis ist vom Typ boolean und liefert nur dann den Wert TRUE, wenn genügend Character Codes für das Grafikfenster zur Verfügung standen, und es wirklich initialisiert wurde. Zur Ausführung benötigt die CPU Zugriff auf den Bildschirmspeicher *(GWindowOff).*

GWindowClear:
Der Inhalt des gesamten Grafikfensters wird gelöscht. Die CPU benötigt hierfür Zugriff auf den Character Definition Table *(GWindowOn).*

GWindowSet(X,Y):
Der Punkt mit den Pixelkoordinaten (X,Y) relativ zur linken oberen Ecke des Grafikfensters wird in der Vordergrundfarbe gesetzt. Hierfür benötigt die CPU ebenfalls Zugriff auf den Zeichensatz *(GwindowOn).*

GWindowReset(X,Y):
Der Punkt mit den Pixelkoordinaten (X,Y) nimmt wieder die Hintergrundfarbe des Grafikfensters an. Der Befehl *GWindowOn* muß auch hier vorausgegangen sein. Diese Library wurde speziell für VGA-Grafikkarten entwickelt, alle Änderungen für den Einsatz mit EGA-Karten sind im kommentierten Listing vermerkt. Die Funktion *GWindowInit* ist für den Einsatz eines Grafikfensters ausgelegt. Wenn mehrere Fenster unabhängig voneinander gleichzeitig dargestellt werden sollen, muß die Unit um fortlaufende Character Codes im zweiten Fenster erweitert werden. Der Einsatz von Grafikfenstern und die damit verbundene Einschränkung des Zeichensatzes lassen sich durch eine erneute Anwahl eines Videomodus aufheben.
VGA-Karten müssen - im Gegensatz zu EGA-Grafikadaptern - zusätzlich von der 9 Bit breiten Zeichenmaske, bei der die letzte Bitspalte ja lediglich angehängt ist, auf die Darstellung der wirklich vorhandenen 8 Bit breiten Maske umgeschaltet werden. Hierzu wird das Clocking-Mode-Register des Sequencers umprogrammiert. Dies erfolgt im Modul *GWindowOn.* Heute übliche Multisync- oder Multiscan-Monitore haben keine Probleme, diese um 1/9 je Rasterzeile kürzeren Videosignale zu synchronisieren. Sollten sie mit Ihrem Monitor jedoch Probleme

haben, so müssen die Register des CRT-Controllers neu programmiert werden (siehe [1] und [4]). Grundsätzlich sind Grafikfenster in allen Textmodi möglich, lediglich die unterschiedlichen Matrizengrößen für einzelne Character erfordern eine Anpassung.

Literatur

[1] *Cebulla, H.:* So funktioniert die VGA, mc 10/88 bis 2/89.

[2] *Eickmann, I.:* Zeichensatz verdoppeln, mc 1/90.

[3] *Smode, D.:* Das große MS-DOS-Profi-Arbeitsbuch, Franzis-Verlag, 1987.

[4] *Wilton, R.:* The programmer's guide to PC and PS/2 video Systems, Microsoft Press, 1987.


```
;-------------------------------------------------------------------------
;     Graphic Windows for TextMode Co80 (VGA)                  release 1.0
;     Copyright (c) Ingo Eickmann, 1989                           07/09/89
;-------------------------------------------------------------------------
;     Assemblieren mit MASM 5.x :                    I. Eickmann
;            MASM GWindow;                           Im Leuchterbruch 8
;     Getestet unter MS-DOS 3.3, Turbo Pascal V4.0   5000 Köln 80
;-------------------------------------------------------------------------
      PAGE  65,80
      TITLE Graphic Windows for TextMode Co80 (VGA)

Sequencer  equ  3C4h                  ; Adresse des Sequencer
GraphicsC  equ  3CEh                  ; Adresse des Graphics Controller
Scanlines  equ  16                    ; Anzahl der Scanlines/Char (EGA: 14)

if1
   Port_out macro low,high            ; Ausgabe eines Wortes an einen
         mov  al,low                  ; Port (dx)
         mov  ah,high
         out  dx,ax
   endm

   BitPosMask macro X,Y,d_X           ; Bitmaske des Pixel (X,Y) ermitteln:
         mov  ax,0B800h               ; Position in es:di, Maske in al
         mov  es,ax
         mov  di,4000h                ; Offset des Character Definition
         mov  ax,Y                    ; Tables 1
         mov  dl,Scanlines
         div  dl                      ; Ermittlung der betroffenen Textzeile
         mov  dl,ah
         mov  bx,d_X
         mul  bl
         mov  bx,X
         mov  cl,3
         shr  bx,cl                   ; Ermittlung der betroffenen Textspalte
         add  ax,bx
         mov  cl,5
         shl  ax,cl                   ; Offset des Characters im Definition
         xor  dh,dh                   ; Table
         add  ax,dx
         add  di,ax

         mov  cx,X                    ; Ermittlung der Bitmaske für das
         and  cl,111b                 ; zugehörige Byte im Character
         mov  al,10000000b            ; Definition Table
         shr  al,cl
   endm
endif

code  segment word public 'CODE'
      assume cs:code

public GWindowOn                      ; CPU erhält Zugriff auf Character
GWindowOn proc far                    ; Definition Tables (Map 2)
         cli
         mov  dx,Sequencer
         Port_out 0,1                 ; Sync. reset
         Port_out 1,1                 ; Clocking Mode: 8 dots/char
         Port_out 2,100b              ; CPU beschreibt Map 2
         Port_out 3,4                 ; Map Select Register
         Port_out 4,7                 ; Sequential Addressing
         Port_out 0,3                 ; Clear Sync. reset
         sti
         mov  dx,GraphicsC
```

```
         Port_out 4,10b              ; CPU liest Map 2
         Port_out 5,0                ; Sequential Addressing
         Port_out 6,1100b            ; Miscellaneous Register
         mov  ax,1000h               ; reset horz. panning
         mov  bx,0013h
         int  10h
         ret
GWindowOn endp

public GWindowOff                    ; CPU erhält wieder Zugriff auf den
GWindowOff proc far                  ; Bildschirmspeicher (Map 0 und 1)
         cli
         mov  dx,Sequencer
         Port_out 0,1                ; Sync. reset
         Port_out 2,3                ; CPU beschreibt Map 0 und 1
         Port_out 4,11b              ; Odd/Even Addressing
         Port_out 0,3                ; Clear Sync. reset
         sti
         mov  dx,GraphicsC
         Port_out 4,0                ; CPU liest Map 0 und 1
         Port_out 5,10h              ; Odd/Even Addressing
         Port_out 6,1110b            ; Miscellaneous Register
         ret
GWindowOff endp

public _GWindowInit                  ; Ein Grafikfenster wird initialisiert
_GWindowInit proc far
                                     ; Parameterübergabe für Turbo Pascal:
   X1    equ  [bp+16]                ;   Koordinaten der linken oberen Ecke
   Y1    equ  [bp+14]
   d_X   equ  [bp+12]                ;   Ausdehnung des Fensters in Text-
   d_Y   equ  [bp+10]                ;   spalten und Textzeilen
   Colf  equ  [bp+8]                 ;   Vordergrundfarbe
   Colb  equ  [bp+6]                 ;   Hintergrundfarbe

         push bp
         mov  bp,sp
         push es
         push di
         mov  ax,0B800h              ; Anwahl des Bildschirmspeichers
         mov  es,ax
         mov  ax,Y1
         mov  bl,80
         mul  bl
         add  ax,X1
         shl  ax,1                   ; Offset der linken oberen Ecke im
         mov  di,ax                  ; Bildschirmspeicher
         mov  ah,Colb
         mov  cl,4
         shl  ah,cl
         or   ah,Colf
         or   ah,1000b               ; Attribute Byte für das Grafikfenster
         and  ax,7F00h
         mov  cx,d_Y
         mov  dx,d_X
         shl  dx,1
Lop2:    xor  bx,bx                  ; Schleife für die Textzeilen
Lop1:    mov  es:[di+bx],ax          ;   Schleife für die Textspalten
         inc  al                     ;     Füllen aller Character Codes im
         add  bx,2                   ;     Grafikfenster mit fortlaufenden
         cmp  bx,dx                  ;     Werten (al)
         jl   Lop1
         add  di,80*2
         dec  cx
         jg   Lop2
         pop  di
         pop  es
         pop  bp
         ret  12                     ; Freigabe der Parametereinträge im
_GWindowInit endp                    ; Stack

public GWindowClear                  ; Löschen aller Bitmasken für das
GWindowClear proc far                ; Grafikfenster
         push es
         push di
         mov  ax,0B800h              ; Anwahl des Videospeichers
         mov  es,ax
         mov  di,4000h               ; Offset für Table 1
         xor  ax,ax
         mov  cx,2000h shr 1         ; Anzahl der Bytes im Char. Def. Table
         cld
         rep  stosw                  ; Überschreiben des gesamten Bereichs
```

```
         pop  di
         pop  es
         ret
GWindowClear endp

public _GWindowSet                    ; Setzen eines Bildpunktes im Fenster
_GWindowSet proc far
                                      ; Parameterübergabe für Turbo Pascal:
   X    equ  [bp+10]                  ;   Pixelkoordinaten
   Y    equ  [bp+8]
   d_X  equ  [bp+6]                   ;   Breite des Fensters

        push bp
        mov  bp,sp
        push es
        push di
        BitPosMask X,Y,d_X            ; Bitmaske und Position ermitteln
        mov  ah,es:[di]
        or   ah,al                    ; Verknüpfung der Maske mit dem
        mov  es:[di],ah               ; ursprünglichen Inhalt
        pop  di
        pop  es
        pop  bp
        ret  6
_GWindowSet endp                      ; Freigabe der Parametereinträge

public _GWindowReset                  ; Zurücksetzen eines Bildpunktes
_GWindowReset proc far
                                      ; Parameterübergabe für Turbo Pascal:
   X    equ  [bp+10]                  ;   Pixelkoordinaten
   Y    equ  [bp+8]
   d_X  equ  [bp+6]                   ;   Breite des Fensters

        push bp
        mov  bp,sp
        push es
        push di
        BitPosMask X,Y,d_X            ; Bitmaske und Position ermitteln
        mov ah,es:[di]
        not al
        and ah,al                     ; Verknüpfung der Maske mit dem
        mov es:[di],ah                ; ursprünglichen Inhalt
        pop di
        pop es
        pop bp
        ret 6                         ; Freigabe der Parametereinträge
_GWindowReset endp

code ends
     end
```

Bild 2. GWINDOW.ASM enthält die Assemblermodule zur Initialisierung und Verwaltung von Grafikfenstern

```
(* Unit - Deklaration für Turbo Pascal 4.0/5.x, I.Eickmann 1989 *)

unit GWindow;

{$F+}

interface
  const Scanlines = 16;                 (* Anzahl der Scanlines (EGA: 14) *)
        Points    = 8;                  (* Anzahl der Dots pro Scanline   *)
  var   d_x,d_y   : Integer;

  procedure GWindowOn;
  procedure GWindowOff;
  procedure _GWindowInit(x1,y1,d_x,d_y : Integer; Colf,Colb : Byte);
  function  GWindowInit(x1,y1,x2,y2 : Integer; Colf,Colb : Byte) : Boolean;
  procedure GWindowClear;
  procedure _GWindowSet(x,y,d_x : Integer);
  procedure _GWindowReset(x,y,d_x : Integer);
  procedure GWindowSet(x,y : Integer);
  procedure GWindowReset(x,y : Integer);

implementation
  {$L GWindow.obj}
  procedure GWindowOn;   external;
  procedure GWindowOff;  external;
  procedure _GWindowInit; external;
```

```
  function GWindowInit; var x,y: Integer;
  begin
    if x1>x2 then
    begin
      x:=x2; x2:=x1; x1:=x;
    end;
    if y1>y2 then
    begin
      y:=y2; y2:=y1; y1:=y;
    end;
    d_x:=x2 - x1 + 1;
    d_y:=y2 - y1 + 1;
    if d_x*d_y > 256 then GWindowInit:=false
    else begin
      _GWindowInit(x1,y1,d_x,d_y,Colf,Colb);
      GWindowInit:=true;
    end;
  end;
  procedure GWindowClear; external;
  procedure _GWindowSet; external;
  procedure _GWindowReset; external;
  procedure GWindowSet;
  begin
    if ((0 <= x) and (x < d_x*8) and (0 <= y) and (y < d_y*Scanlines)) then
      _GWindowSet(x,y,d_x);
  end;
  procedure GWindowReset;
  begin
  if ((0 <= x) and (x < d_x*8) and (0 <= y) and (y < d_y*Scanlines)) then
    _GWindowReset(x,y,d_x);
  end;
end.
```

Bild 3. Die Unit GWINDOW.PAS bindet die Assemblermodule ein und bietet dem Pascal-Programmierer eine Library zur Programmierung von Grafikfenstern

```
(* Demo für Grafikfenster im Textmode Co80, I.Eickmann 1989 *)

program GWindowDemo;

uses Crt,GWindow;

var  a             : Boolean;
     x,y,xMax,yMax : Integer;
     arg           : Real;
Begin
  TextMode(C80);

  a:=GWindowInit(8,1,71,4,red,cyan);

  xMax := d_x * Points;
  yMax := d_y * Scanlines;

  GWindowOn;
  GWindowClear;
  for x:=0 to xMax-1 do
  begin
    arg := 4*PI/xMax * x;
    arg := ((yMax shr 1) - 1) * sin(arg);
    y  := (yMax shr 1) - trunc(arg);
    GWindowSet(x,y);
  end;
  GWindowOff;

  GotoXY(1,7);
  TextColor(LightGray);
End.
```

Bild 4. Das Pascal-Programm DEMO.PAS zeichnet eine Sinusfunktion in ein Grafikfenstern