Immer geradeaus

Wenn Sie sich bereits mit der Erzeugung von Grafiken befaßt haben, konnten Sie eine Programmieraufgabe sicher nicht umgehen: das Zeichnen gerader Linien. Ein besonders zügiges Verfahren bietet hier der Bresenham-Algorithmus.

Haben Sie schon selbst an einem Programm zum Zeichnen von Geraden gestrickt? Dabei ist Ihnen vermutlich nicht entgangen, daß die Programmierung eines schnellen und zuverlässigen Verfahrens nicht eben einfach ist. In Ihrer Schulzeit bedienten Sie sich zum Zeichnen von Geraden einer mathematischen Funktion:

$f(x) = m \cdot x + b$

Der Funktionswert »*f*(*x*)« (oft auch »*y*« genannt) stellt eine y-Koordinate dar, die zu einem bekannten x-Wert gehört. Dazu müssen die Parameter »m« und »b« bekannt sein. »*m*« ist die »Steigung« der Geraden, »*b*« der »Achsenabschnitt«. Ohne Addition des konstanten Faktors »*b*« würde jede Gerade durch den Nullpunkt laufen, sofern sie lang genug ist. »*b*« verschiebt nun den Punkt (0,0) der Geraden, der im Ursprung liegt, auf der y-Achse nach oben oder unten.

Sind die Parameter einmal bekannt, können Sie in einer Schleife einige Werte für »x« in die Funktionsgleichung einsetzen und daraus die y-Werte berechnen. Das Problem besteht darin, die exakte Gleichung zu finden. Im Programmieralltag sieht die Aufgabenstellung für das Zeichnen von Geraden in der Regel so aus: Zwei Punkte sind bekannt, und sollen durch eine Linie verbunden werden. Also beispielsweise die Bildschirmpunkte (100,50) und (200,200).

Die Berechnung des Steigungsparameters »m« für diese Konstellation ist einfach. Sie ergibt sich aus der Formel in Bild 1.

$$m = \frac{dy}{dx} = \frac{y2 - y1}{x2 - x1}$$

$$m = \frac{200 - 50}{200 - 100} = \frac{150}{100} = 1{,}5$$

Bild 1: So berechnen Sie die Steigung einer Geraden

Dabei stehen die beiden Paare (*x1*,*y1*) und (*x2*,*y2*) für die Koordinaten von zwei beliebigen Punkten auf der Geraden. Im Beispiel mit den Punkten (100,50) und (200,200) ergibt sich eine Steigung von 1,5 (Bild 1). Den Wert für den Achsenabschnitt »*b*« erhalten Sie durch Auflösen der Funktionsgleichung nach »*b*«:

$b = f(x) - m \cdot x$

Für »*m*« setzen Sie den soeben errechneten Wert ein, »*f*(*x*)« ersetzen Sie durch eine der beiden y-Koordinaten und »*x*« durch den Wert der dazugehörigen x-Koordinate. Das Beispiel ergibt:

$b = 200 - 200 \cdot 1{,}5 = -100$

im einen Fall, und

$b = 50 - 100 \cdot 1{,}5 = -100$

Sobald »*b*« und »*m*« bekannt sind, läßt sich der Algorithmus recht einfach in ein Programm umsetzen. »line.pas« zeigt das Zeichnen der Linie zwischen (100,50) und (200,200) in einem Pascal-Programm (Listing 1).

Eine kleine Überraschung entsteht dadurch, daß bei den bisherigen Betrachtungen der Ursprung links unten im Koordinatensystem lag. Turbo Pascal legt den Ursprung aber in die linke obere Ecke. Die Linie auf dem Bildschirm hat deshalb die Steigung −1,5.

Umständliche Fließkommaberechnungen bremsen einen einfachen Algorithmus

Testen Sie das Programm mit unterschiedlichen Werten, werden Sie bemerken, daß es mit manchen Steigungen nicht korrekt arbeitet. Zwar ließe sich dieses Problem beheben, wesentlicher ist jedoch eine andere Schwäche des Verfahrens: Es arbeitet sehr langsam.

Grundlegendes Problem des Algorithmus ist, daß er ohne Fließkomma-Arithmetik nicht auskommt. Diese Rechnungen sind zeitaufwendig und bremsen die Methode entsprechend. Schneller zeichnen Sie Linien mit dem Bresenham-Algorithmus, einem Verfahren, das der Informatiker J. E. Bresenham entwickelte und 1965 erstmals veröffentlichte. Der Bresenham-Algorithmus beruht im wesentlichen auf zwei Voraussetzungen: Erstens muß ein Algorithmus zum Zeichnen von Linien lediglich in der Lage sein, Geraden mit einer Steigung zwischen null und eins zu zeichnen. Bild 2 zeigt, in welchem Bereich sich solche Geraden bewegen dürfen. Zweitens nimmt Bresenham an, daß für solche Geraden generell nur zwei Punkte als nächste Bildpunkte der Geraden in Betracht kommen, wenn ein Punkt bereits bekannt ist.

Zur ersten These: Was soll mit Geraden geschehen, die steiler verlaufen, also nicht in den Bereich null bis eins passen? Sehen Sie sich Bild 3 an. Die Abbildung zeigt links eine Gerade, deren Steigung über eins hinausgeht. Spiegeln Sie die Gerade an der Winkelhalbierenden, resultiert daraus eine Gerade mit einer Steigung zwischen null und eins. Die Spiegelung an der Winkelhalbierenden aber ist ein einfaches Verfahren: Es reicht aus, die x- und y-Koordinaten eines jeden Punktes auszutauschen.

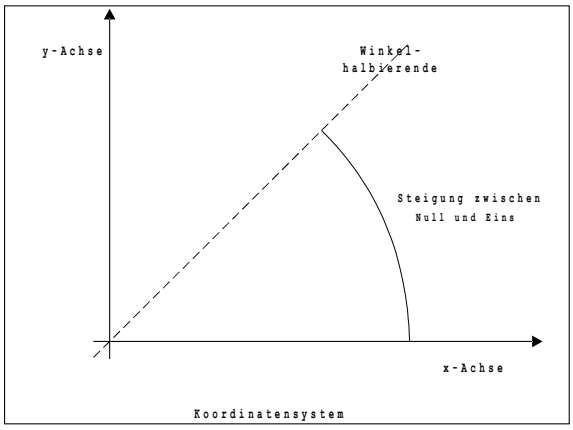

Bild 2: Geraden mit einer Steigung von null bis eins

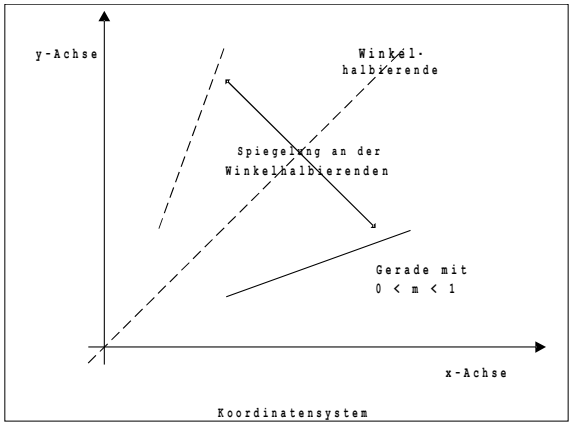

Bild 3: Geraden mit größerer Steigung lassen sich spiegeln

Zu steile Geraden spiegeln Sie an der Winkelhalbierenden

Wenn der Bresenham-Algorithmus also feststellt, daß die Steigung einer Geraden nicht im Bereich null bis eins liegt, spiegelt er die Ausgangskoordinaten. Es ergibt sich eine Steigung im Bereich null bis eins. Die Berechnung der Punkte führt er mit dieser Geraden durch. Vor dem Zeichnen spiegelt er die berechneten Punkte wieder zurück. Das Verfahren ist einfach und schnell zu realisieren, ohne große Umrechnungsroutinen.

Lassen sich alle Steigungen zeichnen, bleibt die Frage nach negativen Steigungen, also »fallenden« Geraden. Hier hilft der Bresenham-Algorithmus mit dem gleichen Trick weiter. Fallende Geraden werden zu steigenden, wenn Sie sie an der x-Achse spiegeln. Das wiederum erreichen Sie durch ein Umdrehen des Vorzeichens aller y-Koordinaten. Sie sehen, Bresenham hat das Problem des Linienzeichnens wirklich auf Geraden mit einer Steigung von null bis eins beschränkt.

Und hier setzt seine zweite Voraussetzung an. Er behauptet, bei diesen Geraden sei die aufwendige Berechnung der y-Koordinaten über die Steigung unnötig, weil grundsätzlich nur zwei Punkte als Kandidaten für den nächsten Geradenpunkt in Betracht kommen, wenn ein Punkt bereits bekannt ist.

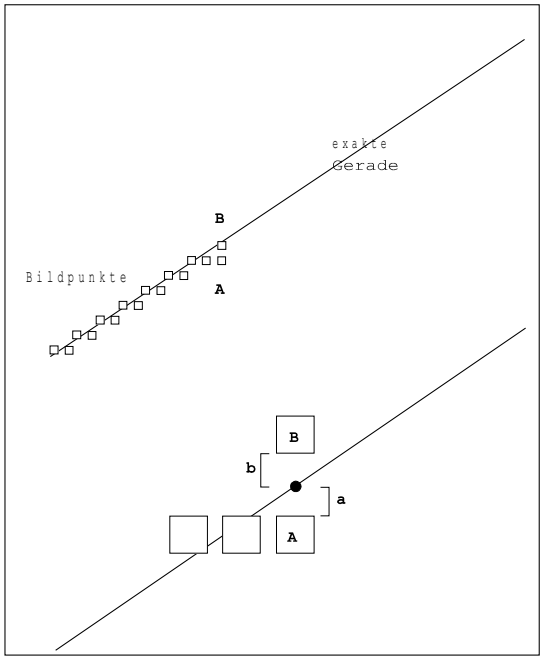

Bild 4: Nur Punkt A oder B kommt für die Gerade in Frage

Bild 4 zeigt das Prinzip: Im oberen Bereich ist der Verlauf einer Linie auf dem Bildschirm zu sehen. Der theoretische Verlauf der exakten mathematischen Geraden wird durch die Auswahl der geeigneten Bildpunkte

bestmöglich nachgebildet. Im unteren Bereich der Abbildung sehen Sie eine Detailansicht der Geraden. Die beiden Punkte links sollen bereits gezeichnet sein.

Für den dritten Punkt kommen nur zwei Kandidaten in Frage. In der Abbildung sind sie als »A« und »B« gekennzeichnet. Welcher Punkt von beiden auszuwählen ist, ergibt sich daraus, welcher näher an der mathematischen Geraden liegt. Daher gilt, die beiden Abstände »a« und »b« in der Zeichnung zu vergleichen. Wenn »a« kleiner ist, erhält Punkt »A« den Zuschlag, andernfalls Punkt »B«.

Bresenham reduziert das Problem auf die Entscheidung zwischen zwei Punkten

Auf der Basis der Geradensteigungsgleichung löst Bresenham diese Frage mathematisch auf. Die Umformung des Terms ist zwar nicht wirklich kompliziert, setzt aber mathematische Standfestigkeit voraus.

Ergebnis des Bresenham-Algorithmus: Die Differenz der beiden Punkte »A« und »B« beträgt entweder »$2 \cdot dy$« (falls zuvor Punkt »B« gesetzt wurde) oder »$2 \cdot dy - dx$« (falls zuletzt Punkt »A« gesetzt wurde). Die Parameter »*dx*« und »*dy*« stehen dabei für die Differenz zweier beliebiger x- und y-Koordinaten von Punkten auf der Geraden.

Bahnbrechend an dieser Erkenntnis ist die Tatsache, daß es sich bei beiden Ausdrücken um Konstanten handelt, die Sie einmal zu Beginn des Programms ermitteln können und die sich im Verlauf des Zeichnens nicht mehr verändern. Außerdem handelt es sich um ganzzahlige Werte. Dadurch ist der Einsatz von Fließkomma-Arithmetik überflüssig.

Zu Beginn des Zeichenvorgangs müssen Sie sich nur entscheiden, ob Punkt »A« oder »B« zu setzen ist. Danach können Sie neue Punkte schlicht und einfach durch Addition ganzzahliger Konstanten bestimmen.

Die erste Entscheidung fällt dabei auch nicht schwer. Der erste Punkt der Geraden befindet sich definitionsgemäß an den Koordinaten (0,0), die Sie dann später in das Koordinatensystem auf dem Bildschirm transponieren. Den zweiten Punkt legen Sie einfach bei (1,0) fest.

In »bline.pas« sehen Sie, wie das Verfahren in ein Pascal-Programm umgesetzt aussieht (Listing 2). Als Beispiel zeichnet die Routine wieder eine Gerade von den Koordinaten (100,50) zu (200,200). Wirklich zügig läuft jedoch auch dieses Programm noch nicht. Der Grund dafür liegt in der Programmiersprache. Effektiv arbeitet der Algorithmus nur, wenn Sie das Programm gleich in Assembler formulieren. Das ist bei »fastline.asm« geschehen (Listing 3).

Das Programm setzt alle Tricks effektiver Programmierung ein, beispielsweise selbstmodifizierenden Code und andere Optimierungsstrategien. Da diese Techniken ein Programm in der Regel schneller, aber unübersichtlicher machen, ist das Listing ausführlich kommentiert.

Alle Listings rufen zum Zeichnen der Punkte auf dem Bildschirm die BIOS-Funktion 0Chex von Interrupt 10hex auf. Die Beispielprogramme setzen aus Kompatibilitätsgründen den Videomodus 04hex ein, der ab der CGA-Karte verfügbar ist.

(Martin Althaus/wn)

```
 1: {Programm: line.pas
 2:  Funktion: Linie ziehen mit einfachem Algorithmus
 3:  Sprache : Turbo Pascal
 4:  Autor   : Martin Althaus
 5:  (C)1991 DMV Widuch GmbH & Co. KG}
 6:
 7: program linie;
 8: uses Dos,Crt;
 9: var cpu: Registers;
10:     merke: Byte;
11:
12: function altermodus: Byte;
13: { Ermittelt den aktuellen Videomodus }
14: begin
15:   cpu.ah:=$0F;
16:   Intr($10,cpu);
17:   altermodus:=cpu.al;
18: end;
19:
20: procedure vmode (modus: Byte);
21: {Setzt einen Videomodus über die BIOS-
22:  Funktion 00h von Interrupt 10h. übergeben
23:  wird der Prozedur ein Byte für den Video-
24:  modus, der gewählt werden soll. }
25: begin
26:   cpu.ah:=0;            {BIOS-Funktion 00h}
27:   cpu.al:=modus;        {gewünschter Modus}
28:   Intr($10,cpu)       {Video-BIOS aufrufen}
29: end;
30:
31: procedure setzepunkt(x,y: Word; farbe: Byte);
32: {Setzt einen Punkt an der Koordinate x,y
33:  im aktuellen Video-Modus. Dazu wird
34:  Funktion 00h des Interrupts 10h auf-
35:  gerufen. }
36: begin
37:   cpu.cx:=x;      {X-Koordinate übertragen}
38:   cpu.dx:=y;      {Y-Koordinate übertragen}
39:   cpu.al:=farbe;         {Farbe übertragen}
40:   cpu.ah:=$0C;        {Funktion 0Ch wählen}
41:   Intr($10,cpu);       {Interrupt auslösen}
42: end;
43:
44: procedure tausche(var a,b: Word);
45: { Tauscht den Inhalt von zwei Word-Variablen aus }
46: var dummy: Word; {Hilfsvariable}
47: begin
48:   dummy:=a; {Inhalt von A merken}
49:   a:=b;     {A wird zu B}
50:   b:=dummy; {B wird zu A}
51: end;
52:
53: procedure Line(x1,y1,x2,y2: Word; farbe: Byte);
54: {Zieht eine Linie zwischen den Punkten
55:  (X1,Y1) und (X2,Y2). Dazu werden die
56:  Punkte gemäß des Geradensteigungs-Algo-
57:  rithmus inkrementell berechnet.}
58: var y: Integer;
59:     m,b: real;
60: begin
61:   if x2<x1 then
62:   begin
```

```
 63:     tausche(x1,x2);
 64:     tausche(y1,y2);
 65:   end;
 66:   if x2<>x1 then
 67:   begin
 68:     m:=(y2-y1)/(x2-x1);
 69:     b:=y1-m*x1;
 70:     for x1:=x1 to x2 do
 71:     begin
 72:       y:=Trunc(m*x1+b);
 73:       setzepunkt(x1,y,farbe);
 74:     end;
 75:   end
 76:   else
 77:   begin
 78:     if y2<y1 then tausche(y1,y2);
 79:     for y1:=y1 to y2 do setzepunkt(x1,y1,farbe);
 80:   end;
 81: end;
 82:
 83: { Hauptprogramm: Setzt Videomodus und zieht eine Linie }
 84: var dummy: Char;
 85:     zaehler: Integer;
 86:     y: Word;
 87: begin
 88:   merke:=altermodus;
 89:   WriteLn('Setze Videomodus 4hex (CGA)');
 90:   WriteLn('und ziehe eine Gerade. ');
 91:   Write('Bitte mit <Return> bestätigen ');
 92:   repeat
 93:   until ReadKey=Chr(13);
 94:   vmode($4);
 95:   for zaehler:=0 to 199 do
 96:     Line(0,0,319,zaehler,zaehler AND 3);
 97:   repeat
 98:   until ReadKey=Chr(13);
 99:   vmode(merke);
100: end.
```

Listing 1. »line.pas« – die einfachste Form eines Linien-Algorithmus

```
 1: {Programm: bline.pas
 2:  Funktion: Linie ziehen mit Bresenham-Algorithmus
 3:  Sprache : Turbo Pascal
 4:  Autor   : Martin Althaus
 5:  (c)1991 DMV Widuch GmbH & Co. KG}
 6:
 7: program bline;
 8: uses Dos,Crt;
 9: var cpu: Registers;
10:     merke: Byte;
11:
12: function altermodus: Byte;
13: { Ermittelt den aktuellen Videomodus }
14: begin
15:   cpu.ah:=$0F;
16:   Intr($10,cpu);
17:   altermodus:=cpu.al;
18: end;
19:
20: procedure vmode(modus: Byte);
21: {Setzt einen Videomodus über die BIOS-
22:  Funktion 00h von Interrupt 10h. übergeben
23:  wird der Prozedur ein Byte für den Video-
24:  modus, der gewählt werden soll. }
25: begin
26:   cpu.ah:=0;            {BIOS-Funktion 00h}
27:   cpu.al:=modus;        {gewünschter Modus}
28:   Intr($10,cpu)       {Video-BIOS aufrufen}
29: end;
30:
31: procedure setzepunkt(x,y: Word; farbe: Byte);
32: { Setzt einen Punkt an der Koordinate X,Y im
33:  aktuellen Video-Modus. Dazu wird Funktion
34:  00hex des Interrupts 10hex aufgerufen. }
35: begin
36:   cpu.cx:=x;      {X-Koordinate übertragen}
37:   cpu.dx:=y;      {Y-Koordinate übertragen}
38:   cpu.al:=farbe;         {Farbe übertragen}
39:   cpu.ah:=$0C;      {Funktion 0Chex wählen}
40:   Intr($10,cpu);       {Interrupt auslösen}
41: end;
42:
43: procedure tausche(var a,b: Word);
44: { Tauscht den Inhalt von zwei Word-Variablen aus }
45: var dummy: Word; {Hilfsvariable}
46: begin
47:   dummy:=a; {Inhalt von A merken}
48:   a:=b;     {A wird zu B}
49:   b:=dummy; {B wird zu A}
50: end;
51:
52: procedure Line(x1,y1,x2,y2: Word; farbe: Byte);
53: {Zieht eine Linie zwischen den Punkten
54:  (X1,Y1) und (X2,Y2). Dazu werden die
55:  Punkte gemäß des Bresenham-Algorithmus
56:  inkrementell berechnet. Sie werden dann
57:  mit der Farbe FARBE versehen. }
58: var dx,dy,dab: Integer;
59:     incA,incB,incY: Integer;
60:     x,y: Integer;      {Die Endkoordinaten}
61: begin
62:   if x2<x1 then    {Müssen Koordianten ge-}
```

```
 63:   begin                    {tauscht werden?}
 64:     tausche(x1,x2);       {Ja, tausche sie}
 65:     tausche(y1,y2);             {beide aus}
 66:   end;
 67:   if (y1<y2)            {Steigung positiv?}
 68:   then                  {ja, also muß Y in}
 69:     incY:=1    {der Schleife erhöht werden}
 70:   else             {Steigung negativ, also}
 71:     incY:=-1;      {muß Y abgezogen werden}
 72:   dx:=x2-x1;     {Berechne die Konstanten:}
 73:   dy:=y2-y1;  {DX, DY, den Anfangswert für}
 74:   dab:=(dy SHL 1)-dx;     {Differenz (b-a)}
 75:   incA:=(dy-dx) SHL 1;   {die alternativen}
 76:   incB:=dy SHL 1;   {Werte für X-Inkrement}
 77:   x:=x1;          {der erste Punkt ist der}
 78:   y:=y1;                     {Anfangspunkt}
 79:   setzepunkt(x,y,farbe);        {setze ihn}
 80:   for x:=x1+1 to x2 do  {setze in Schleife}
 81:   begin
 82:     {die Linie}
 83:     if dab<0 then      {Zuletzt A gesetzt,}
 84:     begin
 85:                 {also ändere das Inkrement}
 86:       dab:=dab+incB;                 {und}
 87:       setzepunkt(x,y,farbe);
 88:                     {setzt den neuen Punkt}
 89:     end       {Es wurde zuletzt B gesetzt,}
 90:     else        {also ändere das Inkrement}
 91:     begin
 92:               {und bereite neue Koordinaten}
 93:       y:=y+incY;           {zum Setzen vor}
 94:       dab:=dab+incA;
 95:       setzepunkt(x,y,farbe);
 96:     end;
 97:   end;                  {Ende der Schleife}
 98: end;
 99:
100: { Hauptprogramm: Setzt Videomodus und zieht eine Linie }
101: var
102:   dummy: Char;
103:   zaehler: Word;
104:   y: Word;
105: begin
106:   merke:=altermodus;
107:   WriteLn('Setze Videomodus 4hex (CGA)');
108:   WriteLn('und ziehe eine Gerade. ');
109:   Write('Bitte mit <Return> bestätigen ');
110:   repeat
111:   until ReadKey=Chr(13);
112:   vmode(4);
113:   for zaehler:=0 to 199 do
114:     Line(0,0,319,zaehler,zaehler AND 3);
115:   repeat
116:   until ReadKey=Chr(13);
117:   vmode(merke);
118: end.
```

Listing 2. Mit »bline.pas« wird eine Gerade nach dem Bresenham-Algorithmus gezeichnet

```
 1: ;Programm: fastline.asm
 2: ;Funktion: Linienzeichnen mit Bresenham-Algorithmus
 3: ;Sprache : MASM ab 4.0
 4: ;Autor   : Martin Althaus
 5: ;(c) 1991 DMV Widuch GmbH & Co KG
 6:
 7: stack512  segment para stack 'stack'
 8:           dw 100h dup (?)
 9: stack512  ends
10: ;
11: data    segment para 'data'
12:         farbe   db 0
13: data    ends
14: ;
15: code    segment para 'code'
16:         assume cs:code, ds:data
17:         assume ss:stack512
18: fastline:
19:         mov     ax,data
20:         mov     ds,ax
21:         mov     ah,0fh      ;Videomodus
22:         int     10h         ;auslesen
23:         push    ax          ;und speichern
24:         mov     ax,4        ;Modus 04hex
25:         int     10h         ;setzen
26: ;
27: ;In einer Schleife einige Geraden zeichnen
28: ;
29:         mov     cx,198      ;99 Linien
30: linie:  mov     ax,0        ;X1=0
31:         mov     bx,198
32:         sub     bx,cx       ;Y1=0,1,2,3..
33:         mov     di,319
34:         mov     es,di       ;X2=319
35:         mov     di,cx       ;Y2=198,197...
36:         push    cx          ;Schleife merken
37:         mov     ch,cl       ;Farbe setzen
38:         call    line        ;Linie zeichnen
39:         pop     cx          ;Schleife zurück
40:         loop    linie       ;und schließen
41: ;
42: ;Taste abwarten und alten Videomode setzen
43: ;
44:         xor     ax,ax
45:         int     16h
46:         pop     ax          ;hole alten Mode
47:         xor     ah,ah       ;über Funktion 0
48:         int     10h         ;setzen
49: ;
50: ;Programm über DOS beenden
51: ;
52:         mov     ax,4c00h    ;über 4Chex
53:         int     21h         ;beenden
54: ;
55: ;Zeichnet eine Linie zwischen Pl und P2
56: ;nach dem Bresenham-Algorithmus
57: ;
58: ;Parameter: AX=X1        ES=X2
59: ;           BX=Y1        DI=Y2
60: ;           CH=Farbe
61: ;
62: line    proc    near
```

```
 63:         push    bp          ;BP retten
 64: ;
 65: ;Koordinaten speichern
 66: ;
 67:         push    ax          ;X1 und
 68:         push    bx          ;Y1 retten
 69:         mov     farbe,ch    ;Farbe speichern
 70:         mov     bx,4340h    ;Code für INC BX
 71:                             ;und INC AX
 72: ;
 73: ;Ist DELTAX positiv?
 74: ;
 75:         mov     cx,es       ;X2 laden und
 76:         sub     cx,ax       ;DELTAX=X2-X1
 77:         jge     deltax_p0   ;X2>X1?
 78: ;
 79: ;Nein, also |DELTAX| bilden
 80: ;
 81:         mov     bl,48h      ;Code für DEC AX
 82:         neg     cx          ;u. DELTAX umdr.
 83: deltax_p0:
 84:         mov     dx,di       ;Y2 laden
 85:         pop     si          ;Y1 laden
 86:         push    si          ;und retten
 87: ;
 88: ;|DELTAY| bilden und Laufvariable bestimmen
 89: ;
 90:         sub     dx,si       ;DELTAY=Y2-Y1
 91:         jge     deltay_p0   ;Y2>Y1?
 92:         mov     bh,4bh      ;Code für DEC BX
 93:         neg     dx          ;u. DELTAY umdr.
 94: deltay_p0:
 95:         mov     si,offset cs:dabgross0
 96:                             ;Ink/Dek im Code
 97:         mov     word ptr cs:[si],bx
 98:                             ;abspeichern
 99:         cmp     cx,dx       ;DELTAX>DELTAY?
100:         jge     deltax_gr0 ;ja, verzweige
101:         mov     bl,90h      ;Y ist Laufvar.
102:                             ;für X NOP
103:         xchg    cx,dx       ;DELTAX und
104:                             ;DELTAY tauschen
105:         jmp     achse0      ;und weiterm.
106: deltax_gr0:
107:         mov     bh,90h      ;X ist Laufvariable
108:                             ;für Y NOP eintragen
109: ;
110: ;Konstanten berechnen
111: ;
112: achse0: mov     si,offset cs:dabklein0
113:                         ;Ink. für Laufvariable
114:         mov     word ptr cs:[si],bx
115:                         ;im Code ablegen
116:         shl     dx,1    ;INCB bestimmen
117:         mov     bp,dx   ;nach BP übertragen
118:         sub     dx,cx   ;DAB bestimmen
119:         mov     di,dx   ;und nach BX
120:         sub     dx,cx   ;INCA nach DX
121:         pop     bx      ;Y1 und
122:         pop     ax      ;X1 laden
123: ;
124: ;Schleife zum Setzen der Punkte der Linie
```

```
125: ;
126: schleife0:
127:         push    di      ;alle Register, die
128:         push    ax      ;verändert werden,
129:         push    dx      ;auf den Stapel
130:         push    bx      ;retten
131:         push    cx
132:         push    si
133:         push    es
134:         mov     cx, ax  ;Punkt wählen
135:         mov     dx, bx
136:         mov     al,farbe;Farbe laden
137:         mov     ah,0ch  ;Funktion wählen
138:         int     10h     ;und Punkt setzen
139:         pop     es
140:         pop     si
141:         pop     cx
142:         pop     bx      ;alle Register
143:         pop     dx      ;vom Stapel holen
144:         pop     ax
145:         pop     di
146:         cmp     di,0        ;DAB>0?
147:         jge     dabgross0   ;ja, verzweige
148: ;
149: ;DAB ist kleiner oder gleich null
150: ;
151: dabklein0:
152:         inc     ax          ;hier wird der
153:         inc     bx          ;Code modifiz.
154:         add     di,bp       ;DAB=DAB+INCB
155:         loop    schleife0   ;Schleife schl.
156:         jmp     end_line0   ;und weiter
157: ;
158: ;DAB ist größer null
159: ;
160: dabgross0:
161:         inc     ax          ;hier wird der
162:         inc     bx          ;Code modifiz.
163:         add     di,dx       ;Schleife schl.
164:         loop    schleife0   ;und weiter
165: end_line0:
166:         pop     bp          ;alten Zustand
167:         ret                 ;Programm enden
168: line    endp
169: code    ends
170:         end      fastline
```

Listing 3. Das Programm »fastline.asm« ist eine sehr schnelle Implementierung des Bresenham-Algorithmus