Löffelmann · Plenge

DAS GRAFIK-BUCH

zu

C 16
C 116
Plus/4

EIN DATA BECKER BUCH

Löffelmann · Plenge

DAS GRAFIK-BUCH

zu

C 16
C 116
Plus/4

EIN DATA BECKER BUCH

ISBN 3-89011-205-6

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Vorwort

Zum ersten Mal auf der Orga-Technik 1984 vorgestellt, konnte sich zunächst keiner für die neue Computergeneration bestehend aus den Computern C16, C116 und Plus/4 (damals nannte man ihn noch C264) begeistern.

Auch zahlreiche Namensänderungen, die Commodore bei dem größten aus dieser Reihe vornahm, waren da recht sinnlos. Es schien, ja es war später sogar von diversen Fachzeitschriften bestätigt: Dieser Computer sollte der größte Flopp sein, den Commodore je auf den Markt gebracht hatte.

Wieso das jedoch so war, nun, das weiß bis heute keiner so recht. Es mag wahrscheinlich daran liegen, daß dieser Computer nur mit sich selbst kompatibel ist, was seine Hardwareeigenschaften angeht. Die Leistung, die der C16 z.B. bringt, sind jedoch durchaus vergleichbar mit denen des C64. Und dort kommt ein weiterer Nachteil. Warum sollte zu der damaligen Zeit jemand 800 DM für einen Computer auf den Tisch legen, mit dem sein Anwender anschließend nur die eingebaute Software nutzen konnte. Da machte sich der Commodore 64 mit seinem zwar sehr bescheidenen BASIC, jedoch dafür mit der wesentlich größeren Softwarepalette doch schon erheblich besser, zumal, wenn man den Preis der dann noch benötigten Software mit einkalkulierte, dieser viel preiswerter war.

Wie dem auch sei, eines hat Commodore jedenfalls erkannt: Den zu hohen Preis für alle Geräte dieser "neuen" Reihe. So war es Ende des Jahres 1985 möglich, einen C16 mit Datasette bereits für, man lese und staune, 99,- DM zu bekommen. Nun öffnete Commodore den Leuten die Türen zur Computerei, die durch die sonst für sie viel zu hohen Kosten, die mit einer Anschaffung eines Computers verbunden waren, abgeschreckt wurden.

Jedoch, wie könnte es anders sein, das nächste Problem kam auf diese glücklichen Neu-Computerbesitzer zu. Es gab und gibt bis heute kaum vernünftige Software zu kaufen, und nur vereinzelt Lektüre zu diesem Computer. Dieses Problem erkannten wir nach einiger Zeit, angespornt eben von solchen Computerfans,

die uns schon fast verzweifelt fragten, ob nicht in geraumer Zeit irgendwelche Bücher für den C16 erscheinen würden, und es entstand das Grafikbuch zu diesen neuen Commodore-Rechnern C16, C116, Plus/4.

Nicht unerwähnt möchten wir an dieser Stelle unsere beiden Helfer lassen, es sind dies Torsten Fahle und Uwe Thiemann, die uns bei der Fertigstellung dieses Buches sehr große Dienste erwiesen.

Ihnen wünschen wir nun viel Spaß bei der Lektüre des Buches und hoffen, daß dieses Buch etwas dazu beträgt, den Softwaremangel, der bei diesen Computern besteht, dadurch auszugleichen, daß Sie nachher in der Lage sind, gute Programme für diese Commodore-Computer zu schreiben.

Klaus Löffelmann
Axel Plenge

Lippstadt, im Juni 1986

Inhalt

1. Einführung - Die Grafikbefehle des C16, C116 und Plus/4

Zu diesem Kapitel muß man folgendes anmerken. Ich habe schon beim Commodore 128 die Erfahrung gemacht, daß immer wieder Leute zu mir kamen und mich um Rat bei dieser oder jener Befehlserkärung fragten. Das ist auch nicht sehr verwunderlich, denn wer kennt die Aufmachung und Vollständigkeit der Commodore-Handbücher nicht?

Aus diesem Grund habe ich in diesem Buch, wie das auch beim Grafikbuch zum Commodore 128 der Fall war, alle Befehlserklärungen noch einmal ausführlich erarbeitet und mit gut lesbaren und verstehbaren BASIC-Demoprogrammen versehen. Ein weiteres Faktum, das mich in diesem Beschluß bestätigte, war die Tatsache, daß es beim Schreiben von Programmen sehr mühselig ist, ständig zwischen Commodore-Handbuch und Grafikbuch hin und her zu blättern, weil man in einem Buch die Befehlssyntax eines Befehls und im anderen die Methode irgendeiner Problemlösung braucht. Hinzu kommt auch noch, daß sich an einigen Stellen im Handbuch zum C16, C116 und Plus/4 Fehler eingeschlichen haben.

Nun also zunächst einige Grundlagen zum Umgang mit den Befehlen, anschließend alle Grafikbefehle des C16, C116 und Plus/4.

1.1 Grundlagen zum Umgang mit den Grafikbefehlen

Zunächst einmal müssen wir klarstellen, was Grafik eigentlich ist. Im Grunde genommen kann man das, was man nach dem Einschalten des Computers zu Gesicht bekommt, schon als Grafik bezeichnen. Dann nämlich erscheint die sogenannte Textgrafik. Damit ist es auch schon möglich, recht hübsche Bilder zu erzeugen, wie das im folgenden Beispiel gemacht wird:

```
100 print "     +                 ++          "
110 print "    +++              +   +         "
120 print "     +              + + + +        "
130 print "    + +            + +     +       "
140 print "   +   +          +   + + +        "
150 print "  +     +        +             +   "
160 print "  +       +      + + + +           "
170 print "  ++++++++++++++    ++  +           "
180 print "  + +++      +      +  +           "
190 print "  + + +      +      +  +           "
200 print "  + +++      +      +  +           "
210 print "  +          ++     +  +           "
220 print "  ++++++++++++++++   ++++++         "
230 print "                                   "
```

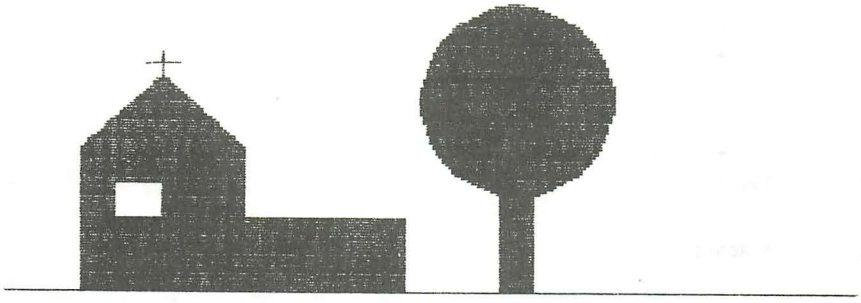
Sie selbst werden sicherlich viel schönere Gebilde als dieses zustande bringen. Experimentieren Sie doch einmal! Probieren Sie dabei auch die Tastenkombinationen, die Sie jeweils links und rechts vor einer Buchstabentaste finden. Diese Zeichen, sie werden auch Grafikzeichen oder Sonderzeichen genannt, erreichen Sie jeweils mit der Commodore-Taste oder mit der Shift-Taste.

Weiterhin ist es möglich, auch feinere Gebilde zu erzeugen, als es jemals mit der groben Textgrafik möglich wäre. Diese Grafikart nennt sich dann hochauflösende Grafik. Hochauflösend deswegen, weil Sie nun gezielt einzelne Punkte setzen können.

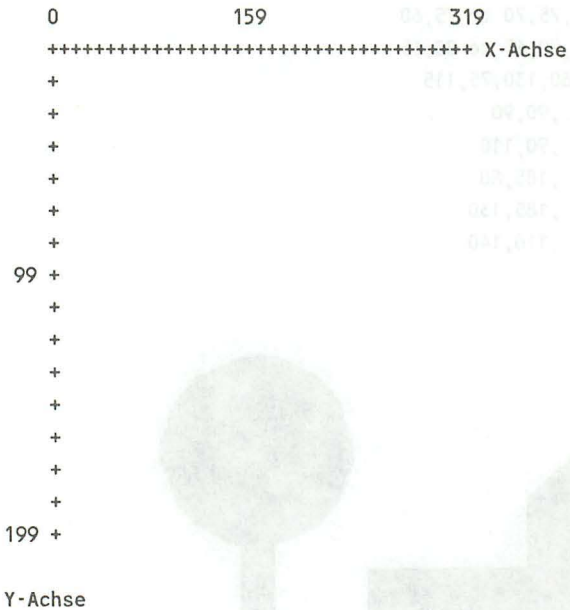
Nehmen Sie dafür einmal das folgende Programm als Beispiel, ohne sich zunächst um die Funktionsweise des Programms Gedanken zu machen:

```
100 graphic 1,1
110 box ,50,160,100,100
120 draw ,100,100 to 75,70 to 50,100
130 box ,100,130,150,160
140 box ,180,160,190,120
150 circle ,185,80,30,40,,,,12
160 draw ,0,161 to 319,161
```

```
170 draw ,75,70 to 75,60
180 draw ,70,65 to 80,65
190 box ,60,130,75,115
200 paint ,90,90
210 paint ,90,110
220 paint ,185,80
230 paint ,185,130
240 paint ,110,140
```



Wie funktioniert nun so etwas? Nun, im Prinzip ist das ganz einfach. Der Bildschirm Ihres Computers ist wie ein Koordinatensystem eingeteilt. Dabei liegt die Koordinate (0,0) in der linken oberen Ecke. Ein Schaubild mag dieses verdeutlichen:



Sie brauchen nun Ihrem Computer nur noch zu "sagen", an welcher Position Sie welches Gebilde plaziert haben möchten, und schon führt dieser das durch. Sie sehen selbst, so schwer ist das doch gar nicht.

Leider ist es in dieser Auflösung nur möglich, einfarbige Bilder zu erzeugen. Aber es existiert noch eine weitere Grafikart, die sogannte Multicolor-Grafik. In der Multicolor-Grafik halbiert sich die Anzahl der horizontalen Punkte, also der Punkte, die auf der X-Achse liegen. Dafür stehen dann insgesamt vier Farben zur Verfügung.

Die nun folgenden Befehle befassen sich mit diesem Koordinatensystem. Sie sollten, bevor Sie nun mit Eifer diese Befehle ausprobieren, zunächst alles über den Befehl "GRAPHIC" nachlesen und die folgenden Dinge beherzigen:

Sie können grundsätzlich die Parameter eines Grafikbefehls, die Sie nicht benötigen, weglassen. In der Regel wird für diesen

Parameter dann der Wert "0" eingesetzt. Anstelle dieses Parameters muß dann ein Komma gesetzt werden.

1.1.1 Die hochauflösende Grafik

Die BOX-Anweisung:

Format: BOX fq,x1,y1,x2,y2,wn,au

Parameter:

- fq Farbquelle.
  0=Hintergrund
  1=Vordergrund
  2=Zusatzfarbe 1
  3=Zusatzfarbe 2
  Der Wert definiert die Quelle der Zeichenfarbe

- x1,y1 Koordinaten des linken, oberen Eckpunktes
  x1: HGR 0-319 MC 0-159
  y1: HGR 0-199 MC 0-199

- x2,y2 Koordinaten des rechten, unteren Eckpunktes. Wertebereiche wie x1 bzw. y1

- wn Winkel, um den das Rechteck gedreht werden soll (in Grad). Voreingestellt ist der Wert 0.

- au Legt fest, ob das Reckteck ausgemalt werden soll (1) oder nicht (0).

Beispiel: BOX 1,10,10,100,100,45,1

Funktion: Zeichnen eines Rechtecks

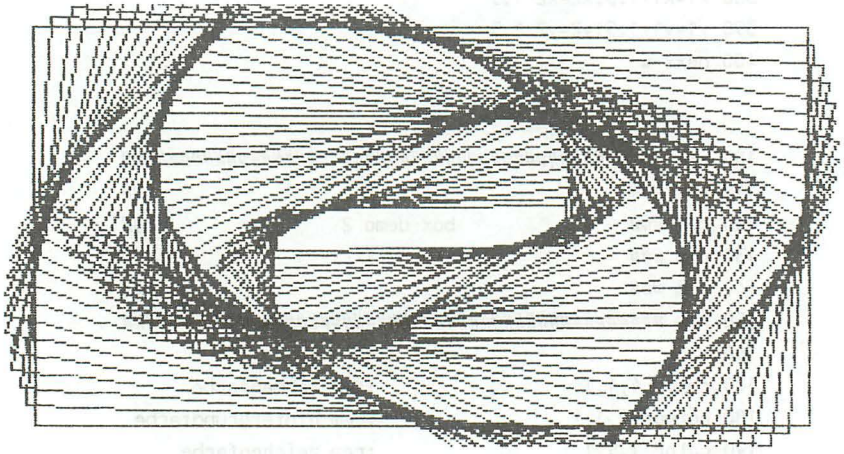
Erläuterungen:

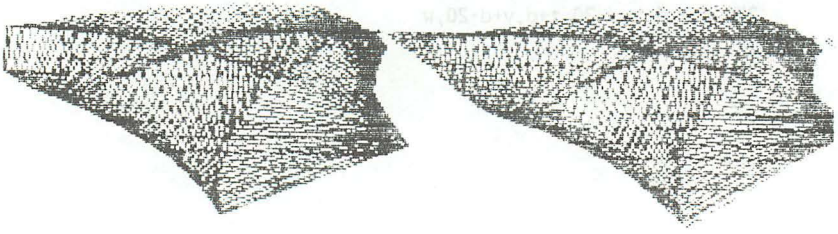
Dieser Befehl schließt gleich mehrere Optionen in sich ein: Das Rechteck ist ausfüllbar und läßt sich drehen. Die in den Parametern angegebene Farbquelle gibt praktisch an, welche Farbe benutzt werden soll. Die Definition der einzelnen Farbquellen bewirkt der COLOR-Befehl. Dort werden die Quellen auch erklärt.

Beispielprogramme:

```
100 rem ##############################################
110 rem ###                                        ###
120 rem ###              box-demo 1                ###
130 rem ###              ----------                ###
140 rem ###                                        ###
150 rem ##############################################
160 :
170 color 4,1,1                 :rem rahmenfarbe
180 color 0,1,1                 :rem hintergrundfarbe
190 color 1,5,1                 :rem zeichenfarbe
200 graphic 1,1                 :rem hgr einschalten
210 x1=30:x2=290                :rem ausgangskoordinaten
220 y1=10:y2=190
230 :
240 for w=0 to 40 step 3        :rem einmal nach rechts
250 box 1,x1,y1,x2,y2,w         :box zeichnen
260 x1=x1+1.5:x2=x2-1.5
270 y1=y1+1.5:y2=y2-1.5
280 next w
290 :
300 for w=400 to 320 step -3    :rem dann weit nach links
310 box 1,x1,y1,x2,y2,w
320 x1=x1+1.5:x2=x2-1.5
330 y1=y1+1.5:y2=y2-1.5
340 next w
350 :
360 for w=320 to 360 step 3     :rem und wieder Normal-
370 box 1,x1,y1,x2,y2,w         :rem stellung
```

```
380 x1=x1+1.5:x2=x2-1.5
390 y1=y1+1.5:y2=y2-1.5
400 next w


100 rem ##############################################
110 rem ###                                          ###
120 rem ###              box-demo 2                  ###
130 rem ###             --------------               ###
140 rem ###                                          ###
150 rem ##############################################
160 :
170 color 4,1,1                 :rem rahmenfarbe
180 colo0r 0,1,1                :rem hintergrundfarbe
190 color 1,5,1                 :rem zeichenfarbe
200 graphic 1,1                 :rem hgr einschalten
210 :
220 d=20:f=2
230 for z=0 to 319 step 2
240 y=40*sin(x/25)+100
250 d=d+f:w=w+2
260 if d>80 or d<0 then f=-f
270 box 1,z,y-20,z+d,y+d-20,w
280 next z
```





Der CIRCLE-Befehl

Format:		CIRCLE fq,x,y,xr,yr,st,en,wn,sw
Parameter:	- fq	Farbquelle (siehe COLOR)
	- x,y	Koordinaten des Mittelpunkts
	- xr	Radius der Hauptachse
	- yr	Radius der Nebenachse Voreingestellt ist der Wert xr
	- st	Startwinkel (in Grad) Voreingestellt ist der Wert 0
	- en	Endwinkel (in Grad) Voreingestellt ist der Wert 360
	- wn	Winkel, um den die Figur gedreht werden soll (in Grad) Voreingestellt ist der Wert 0
	- sw	Gibt den Winkelabstand zwischen zwei Berechnungspunkten an. Wertebereich: 1-255 -je kleiner der Wert, desto runder die Figur. Voreingestellt ist der Wert 2
Beispiel:		CIRCLE 1,100,100,60,30,,,40,3
Funktion:		Zeichnen einer Ellipse, eines Kreises oder eines bestimmten Polygons.

Erläuterungen:

Der CIRCLE-Befehl ist sehr vielfältig. So ist es nicht nur möglich, Kreise oder Ellipsen zu zeichnen. Sogar Polygone sind mit diesem leistungsfähigen Befehl kein Problem. Hierbei ist der

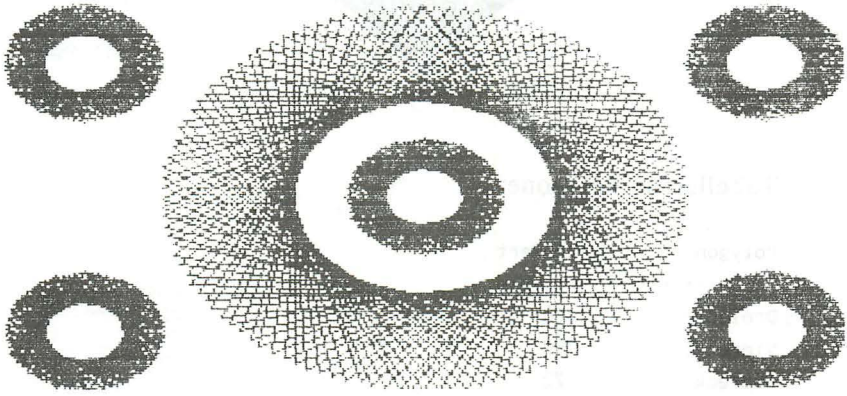
Parameter sw sehr wichtig. Je kleiner dieser Wert, desto kreisähnlicher das Gebilde. Die untenstehende Tabelle gibt über die konkreten Werte für diesen Parameter zur Polygonzeichnung mehr Aufschluß.

Beispielprogramm:

```
100 rem ################################################
110 rem ###                                          ###
120 rem ###            circle-demo 1                 ###
130 rem ###            ----------------              ###
140 rem ###                                          ###
150 rem ################################################
160 :
170 color 4,1,1                  :rem rahmenfarbe
180 color 0,1,1                  :rem hintergrundfarbe
190 color 1,5,1                  :rem zeichenfarbe
200 graphic 1,1                  :rem hgr einschalten
210 :
220 for x=1 to 120 step 3
230 circle 1,160,100,100,100,,,x,120
240 circle 1,30,30,30,30,,,x,120
250 circle 1,30,170,30,30,,,x,120
260 circle 1,290,170,30,30,,,x,120
270 circle 1,290,30,30,30,,,x,120
280 circle 1,160,100,30,30,,,x,120
290 next x
```

```
100 rem ################################################
110 rem ###                                          ###
120 rem ###            circle-demo 2                 ###
130 rem ###            ----------------              ###
140 rem ###                                          ###
150 rem ################################################
160 :
170 color 4,1,1                  :rem rahmenfarbe
180 color 0,1,1                  :rem hintergrundfarbe
190 color 1,5,1                  :rem zeichenfarbe
```

```
200 graphic 1,1                    :rem hgr einschalten
210 :
220 for r=0 to 160 step 5
230 circle 1,160,100,r,,-(r-160)*.625,,,,12
240 next
```



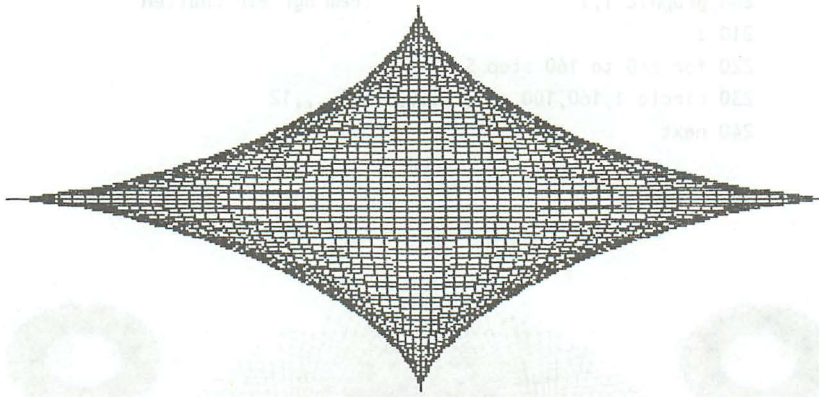


Tabelle für Polygone:

Polygon	Seitenwert
Dreieck	120
Viereck	90
Fünfeck	72
Sechseck	60
Zehneck	36
Zwölfeck	30

Allgemein können Sie die Anzahl der Ecken eines Polygons durch folgende Formel errechnen:

Anzahl der Ecken = 360 / sw

Der CHAR-Befehl

Format: CHAR fq,x,y,z$,iv

Parameter:

- fq — Farbquelle
  0=Hintergrundfarbe
  1=Vordergrundfarbe
  2=Zusatzfarbe 1
  3=Zusatzfarbe 2
- x,y — Koordinaten des Textanfangs
  x: 0-39
  y: 0-24
- z$ — Text, der ausgegeben werden soll
- iv — 0=Normale Darstellung
  1=reverse Darstellung

Beispiel: CHAR 1,3,20,"Der CHAR-Befehl"

Funktion: Plaziert einen Text auf dem Grafikbildschirm.

Erläuterungen:

Die Farbquelle gibt, wie der Name schon sagt, die Quelle an, aus der die zu benutzende Farbe stammt. Die Farbquelle wird bei dem COLOR-Befehl genauer erklärt.

Es gibt im Commodore-BASIC sogenannte Steuerzeichen. Das sind Zeichen, die nicht gedruckt werden, sondern irgend eine Funktion auslösen. Diese Zeichen, deren Funktion im Anhang übrigens aufgelistet ist, funktionieren im Grafikmodus nicht!.

Der DRAW-Befehl

Format: DRAW fq,x1,y1 TO x2,y2 .....

Parameter:
- fq Farbquelle (siehe BOX)
- x1,y1 Koordinaten des Startpunktes
- x2,y2 Koordinaten des Endpunktes

Beispiel: DRAW 1,10,10 TO 20,50

Funktion: Zeichnen einer Linie

Erläuterungen:

Mit dem DRAW-Befehl können beliebig viele Linien aneinandergefügt werden. Beispiel: DRAW 1,10,10 TO 20,50 TO 30,5. Sind Start- und Endpunkte gleich, so wird ein Punkt gezeichnet. Dieser Effekt wird auch erzielt, wenn Sie nur die ersten 3 Parameter, ohne "TO", angeben.

Wird das "TO" durch ein Komma ersetzt, so werden jeweils nur Punkte an die entsprechenden Koordinaten gesetzt und nicht mit Linien verbunden.

Weiterhin haben Sie die Möglichkeit, vom Graphic-Cursor (siehe LOCATE) bis zu einem Endpunkt eine Linie zu ziehen. In diesem Fall müssen die Anfangskoordinaten entfallen. Dann existiert noch die Möglichkeit, relative Koordinaten anzugeben. Das bedeutet daß eine Linie relativ zu den zuvor angegebene Koordinaten gezeichnet wird. Schließlich ist es noch möglich, die Endkoordinaten einer Linie als Polarkoordinaten anzugeben.

Damit existieren eine Reihe verschiedener Möglichkeiten, diesen Befehl anzuwenden.

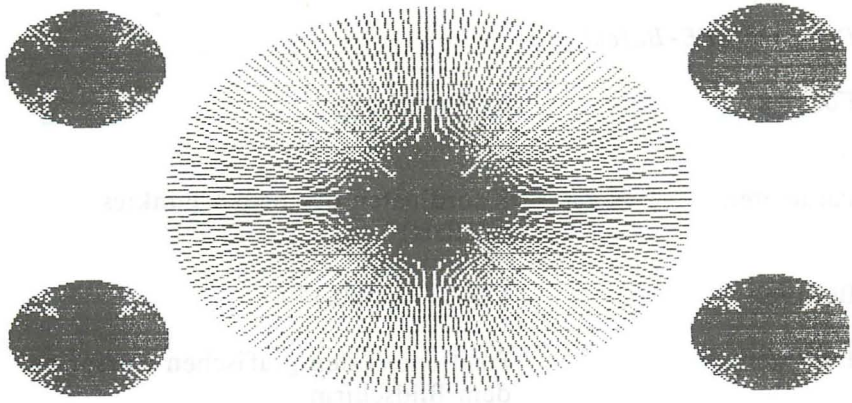
```
DRAW M,X1,Y1 TO X2,Y2   : Löschen oder Setzen einer Linie
DRAW M,X1,Y1 (, X2,Y2..): Setzen eines oder mehrerer Punkte
DRAW M       TO X1,Y1   : Zeichnen einer Linie vom
                          Grafikcursor nach x1,y1
DRAW M                  : Zeichnen oder Löschen eines
                        : Punktes an der Grafcur.-Position
DRAW M,X1,Y2 TO +X,+Y   : zeichnen mit relat. Koordinaten
DRAW M,X1,Y2 TO L;W     : Zeichnen des Radius (Polakoord.)
```

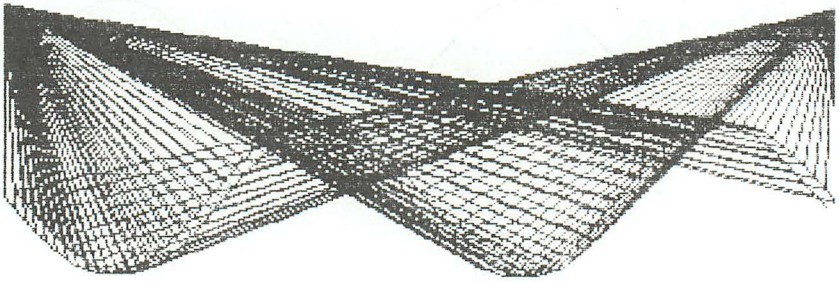
```
100 rem ################################################
110 rem ###                                            ###
120 rem ###              draw-demo 1                   ###
130 rem ###             ---------------                ###
140 rem ###                                            ###
150 rem ################################################
160 :
170 color 4,1,1                 :rem rahmenfarbe
180 color 0,1,1                 :rem hintergrundfarbe
190 color 1,5,1                 :rem zeichenfarbe
200 graphic 1,1                 :rem hgr einschalten
210 :
220 for x=0 to 319 step 4
230 y=40*sin(x/25)+100
240 if f=0 then f=1:locate x,y
250 draw 1 to x,y
260 next x
```

```
100 rem ################################################
110 rem ###                                            ###
120 rem ###              draw-demo 2                   ###
130 rem ###             ---------------                ###
140 rem ###                                            ###
150 rem ################################################
160 :
170 color 4,1,1                 :rem rahmenfarbe
180 color 0,1,1                 :rem hintergrundfarbe
190 color 1,5,1                 :rem zeichenfarbe
200 graphic 1,1                 :rem hgr einschalten
```

```
210 :
220 for x=0 to 319 step 4
230 y=40*sin(x/25)+100
240 if f=0 then f=1:xa=x:ya=y
250 draw ,x,y to xa,ya to 0,0 to 320-xa,200-ya to 320-x,200-y to
319,0
260 xa=x:ya=y
270 next
```

```
100 rem ##############################################
110 rem ###                                          ###
120 rem ###              draw-demo 3                 ###
130 rem ###             --------------               ###
140 rem ###                                          ###
150 rem ##############################################
160 :
170 color 4,1,1                 :rem rahmenfarbe
180 color 0,1,1                 :rem hintergrundfarbe
190 color 1,5,1                 :rem zeichenfarbe
200 graphic 1,1                 :rem hgr einschalten
210 :
220 for x=0 to 360 step 2
230 draw ,160,100 to 100;x
240 draw ,30,30 to 30;x
250 draw ,30,170 to 30;x
260 draw ,290,170 to 30;x
270 draw ,290,30 to 30;x
280 next x
```





Der LOCATE-Befehl

Format:		LOCATE x,y
Parameter:	- x,y	Koordinaten des neuen Punktes. (scaliert)
Beispiel:		LOCATE 100,130
Funktion:		Positioniert den grafischen Cursor auf dem Bildschirm.

Erläuterungen:

Der grafische Cursor ist nicht sichtbar und für alle grafischen Zeichenanweisungen der Ausgangspunkt. Nach der Ausführung einer Zeichenanweisung ist der letzte gezeichnete Punkt der neue Ausgangspunkt.

Im Klartext: Um eine bestimmte Figur mit Grafikbefehlen vom Grafikcursor aus zu zeichnen, müssen Sie einfach die ersten X- und Y-Koordinaten weglassen.

Der PAINT-Befehl

Format:		PAINT fq,x,y,mo
Parameter:	- fq	Farbquelle (siehe BOX)
	- x,y	Koordinaten des Startpunktes (scaliert)
	- mo	Legt fest, ob der auszumalende Bereich von der gewählten Farbe (0) oder einer anderen als der Hintergrundfarbe umgeben ist (1) (siehe Erläuterungen)
Beispiel:		PAINT 1,100,100,0
Funktion:		Füllen eines grafischen Bereiches mit einer wählbaren Farbe.

Erläuterungen:

Der PAINT-Befehl dient dem Ausfüllen beliebiger Flächen. Diese müssen eindeutig begrenzt sein. Das bedeutet, daß die Begrenzung lückenlos sein muß. Logischerweise muß der Startpunkt innerhalb der auszufüllenden Fläche sein. Nach soviel "muß" kann man nun auch wählen: Die Farbquellen sind die gleichen wie bei dem BOX-Befehl und werden natürlich auch mit dem COLOR-Befehl definiert (siehe dort). Nun gibt es aber noch zwei Möglichkeiten, wie die Fläche begrenzt ist: Die gleiche Farbe, wie die in der angegebenden Farbquelle, sowie eine fremde. Diese fremde Farbe kann nicht die Hintergrundfarbe sein, da die Begrenzung in einem solchen Fall nicht gegeben ist. Haben wir uns nun für eine Begrenzungsfarbe entschlossen, so geben wir den dazugehörenden Modus mit dem PAINT-Befehl an: 0 für die gleiche Farbe (wie in der angegebenden Farbquelle), sowie 1 für die fremde Farbe.

Ein Beispiel soll dieses erläutern:

```
100 rem ##############################################
110 rem ###                                          ###
120 rem ###              paint-demo                  ###
130 rem ###              --------------              ###
140 rem ###                                          ###
150 rem ##############################################
160 :
170 color 4,1,1              :rem rahmenfarbe
180 color 0,1,1              :rem hintergrundfarbe
190 color 1,5,1              :rem zeichenfarbe
200 graphic 1,1              :rem hgr einschalten
210 :
220 circle 1,160,100,50      :rem kreis zeichnen
230 paint 1,160,100          :rem kreis mit zeichen-
240 :                        :rem farbe ausmalen
250 paint 0,160,100          :rem kreis mit hinter-
260 :                        :rem grundfarbe ausmalen
```

Die RDOT-Funktion

Format: x=RDOT(n)

Parameter: - n Gibt an, welcher Wert mit der Funktion gelesen werden soll:
0=x-Position des grafischen Cursors.
1=y-Position des grafischen Cursors.
2=Farbquell-Code, aus dem die Farbe des Punktes bei der aktuellen Position des grafischen Cursors resultiert.

Beispiel: x=RDOT(0)

Funktion: Liefert die aktuelle Position des grafischen Cursors oder den für die Position gültigen Farbcode der Farbquelle.

Erläuterungen:

Dieser Befehl schließt eigentlich zwei verschiedene Möglichkeiten ein. Die erste Möglichkeit, die Sie durch diesen Befehl haben, ist die, daß Sie die aktuelle Position des Grafikcursors abfragen können. Die zweite ermöglicht es Ihnen festzustellen, ob oder bzw. in welcher Farbe ein Punkt unter dem Grafikcursor zu sehen ist. Das folgende Beispiel, das zunächst viele Punkte auf den Bildschirm malt und dann die am weitestenauseinanderliegenden miteinander verbindet, soll dieses verdeutlichen:

```
100 rem ##############################################
110 rem ###                                        ###
120 rem ###              rdot-demo                 ###
130 rem ###            --------------              ###
140 rem ###                                        ###
150 rem ##############################################
160 :
170 color 4,1,1                 :rem rahmenfarbe
180 color 0,1,1                 :rem hintergrundfarbe
190 color 1,5,1                 :rem zeichenfarbe
200 graphic 1,1                 :rem hgr einschalten
210 box ,0,0,319,199
220 for z=0 to 100              :rem 100 punkte
230 x=int(319*rnd(1))           :rem zufällig setzen
240 y=int(199*rnd(1))
250 draw ,x,y
260 next z
270 :
280 for y=0 to 199              :rem obersten punkt
290 for x=0 to 319              :rem suchen
300 locate x,y                  :rem GC setzen
310 if rdot(2)=1 then 340       :rem Punkt gefunden,=> 340
320 next x,y
330 :
340 x1=x:y1=y                   :rem obersten punkt merken
350 :
```

```
360 for y=199 to 0 step -1      :rem untersten punkt
370 for x=319 to 0 step -1      :rem suchen
380 locate x,y                  :rem GC setzen
390 if rdot(2)=1 then 420       :rem Punkt gefunden,=> 420
400 next x,y
410 :
420 draw 1 to x1,y1             :rem linie zeichnen
```

Der SCALE-Befehl

Format: SCALE mo

Parameter: - mo Modus.
0=Scalierung ausschalten
1=Scalierung einschalten

Beispiele: SCALE 1
SCALE 0

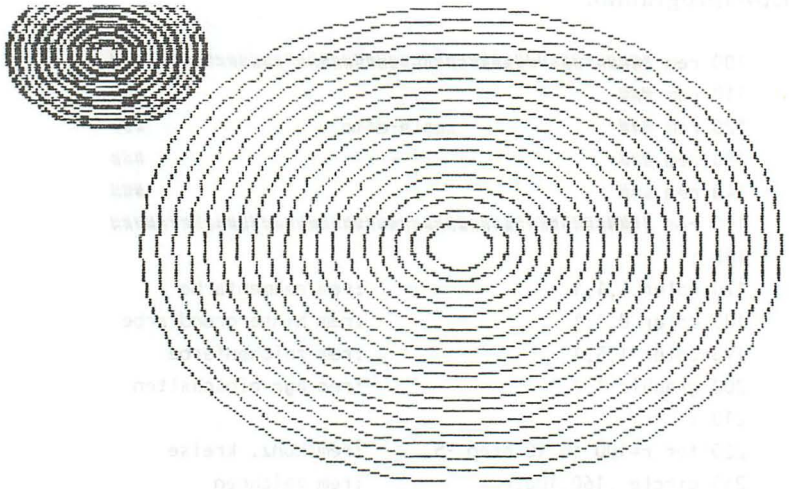
Funktion: Schaltet die Scalierung aus oder ein.

Erläuterungen:

Der SCALE-Befehl vermag den Wertebereich des Bildschirms in der Grafik zu erweitern. Dabei wird selbstverständlich die maximale Auflösung von 320*200 Punkten in der HGR und 160*200 in Multicolor beibehalten und nur die eigentliche Koordinate mit einem bestimmten Faktor multipliziert, so daß der Bildschirm nun eine Auflösung von 1024 mal 1024 Punkten erhaält. Dadurch werden alle Figuren auf dem Bildschirm logischerweise kleiner. Einen wesentlichen Vorteil hat dieser Befehl jedoch: Die lästige Umrechnung der Koordinaten bei diesem erweiterten Koordinatensystem (dieses ist manchmal erforderlich, z. B. in der Statistik) entfällt.

Beispielprogramm:

```
100 rem ##############################################
110 rem ###                                          ###
120 rem ###              scale-demo                  ###
130 rem ###             --------------               ###
140 rem ###                                          ###
150 rem ##############################################
160 :
170 color 4,1,1                  :rem rahmenfarbe
180 color 0,1,1                  :rem hintergrundfarbe
190 color 1,5,1                  :rem zeichenfarbe
200 graphic 1,1                  :rem hgr einschalten
210 :
220 for r=100 to 10 step -5      :rem konz. kreise
230 circle ,160,100,r            :rem zeichnen
240 next r
250 scale 1                      :rem skalierung ein
260 for r=100 to 10 step -5
270 circle ,160,100,r            :rem dieselben kreise
280 next r
290 scale 0                      :rem skalierung aus
```



1.1.2 Die Textgrafikbefehle

Die Textgrafik ist eigentlich nichts anderes als die Gestaltung des Bildschirms mit normalen Textzeichen, wie wir ja schon anfangs gehört haben. Durch die nun folgenden Befehle ist es uns möglich, den Bildschirmaufbau sinnvoller als nur durch POKE- oder PRINT-Befehle zu gestalten.

Der CHAR-Befehl (Textgrafik)

Format: CHAR fq,x,y,z$

Parameter: - fq : Farbquelle
0=Hintergrundfarbe
1=Vordergrundfarbe
2=Zusatzfarbe 1
3=Zusatzfarbe 2

- x,y Koordinaten des Textanfangs.
  x:0-39 bzw. 79
  y:0-24

- z$ Text, der ausgegeben werden soll.

Beispiel: CHAR 1,3,20,"Der CHAR-Befehl"

Funktion: Plaziert einen bestimmten Text an einer bestimmten Bildschirmstelle.

Erläuterungen:

Im Text enthaltene Steuerzeichen werden hier im Textmodus ausgeführt, jedoch können nur Strings, keine Fließkommata oder Integervariablen durch den CHAR-Befehl positioniert werden. Es gibt jedoch einen kleinen Trick, um dieses zu erreichen. Wandeln Sie zunächst Ihre Zahl mit dem STR$-Befehl in einen String um und positionieren Sie diesen dann durch den CHAR-Befehl.

Als Beispiel für die gerade für diesen Befehl typische Textgrafik haben wir für Sie zwei Spieleprogramme für Ihren Computer programmiert, welche im folgenden ausgedruckt sind. Es handelt sich bei dem ersten Spiel um das bekannte Hang-Man-Spiel, das früher auch als beliebtes "unter-der-Schulbank-Spiel" unter dem Namen "Galgenmännchen" bekannt war.

Beim zweiten Spiel handelt es sich um ein Spiel aus den Anfängen der Spielcomputer. Es nennt sich in unserem Fall "Squash" und ist auf Spielgeräte zurückzuführen, die damals unter so hochtrabenden Namen wie "Fußball", "Tennis" oder "Tontaubenschießen" verkauft wurden:

```
100 REM ***----------------------------***
110 REM **------------------------------**
120 REM *-                              -*
130 REM --     H A N G - M A N          --
140 REM --                              --
150 REM --                 (TF/03/86)   --
160 REM *-                              -*
170 REM **------------------------------**
180 REM ***----------------------------***
190 :
200 COLOR0,3,0
210 COLOR4,1
220 COLOR1,8,7
230 VOL 7
240 :
250 AZ = 5:REM ANZAHL DER WORTE PRO SCHWIERIGKEITSGRAD
260 F  =0 :REM ANZAHL DER FEHLER(MAX12)
270 RV$=CHR$(18)+" "+CHR$(146):REM REVERSES SPACE
280 MK$="":REM VARIABEL ZUM MERKEN DER BUCHSTABEN
290 RESTORE 1820:REM AUF FRAGEWOERTER STELLEN
300 :FOR J= 1 TO 8  :REM 8 SCWIERIGKEITSGRADE
310 :  FOR I= 1 TO AZ
320 :   READ F$(J,I)  :REM WOERTER EINLESEN
330 :  NEXT I
340 :NEXT J
350 :
360 :
370 SCNCLR
380 CHAR,8,2,"┌────────────────────┐"
390 CHAR,8,3,"  H  A  N  G   M  A  N  "
400 CHAR,8,4,"└────────────────────┘"
410 :
420 :   CHAR,16,10," ┌──┐":REM   O Y Y P
430 :   CHAR,16,11," ●  │":REM   Q     N
440 :   CHAR,16,12,"/│\ │":REM N - M   N
450 :   CHAR,16,13," │  │":REM   -     N
460 :   CHAR,16,14,"/ \ │":REM N   M   N
470 :   CHAR,16,15,"_____┘":REM P P P P @
480 CHAR,0,18,"":REM CURSOR POSITIONIEREN
490 :
500 PRINT" SIE HABEN BELIEBIG VIELE VERSUCHE EIN"
510 PRINT
520 PRINT"VOM COMPUTER GESTELLTES WORT ZU ERRATEN."
530 PRINT
540 PRINT" ES GELTEN DIE NORMALEN HANG-MAN REGELN"
550 CHAR,14,24,"<<< TASTE >>>"
560 :
570 GETKEY A$
580 :
590 SCNCLR
600 :
610 CHAR ,2,4,"WELCHER SCHWIERIGKEITSGRAD[1-8]"
620 INPUT SG
630 IF SG<1 OR SG>8 THEN 610
640 F$=F$(SG, INT(RND(1)*AZ+1))
650 :
660 RESTORE 1390 :REM AUF GALGEN STELLEN
```

```
670 CHAR,6,10,"ES KANN LOS GEHN."
680 GETKEY A$
690 :
700 REM ------------------------------
710 REM ---       SPIELBEGINN        ---
720 REM ------------------------------
730 :
740 SCNCLR
750 :
760 FL =LEN(F$)
770 :FOR I=1 TO FL
780 :  CHAR1,12+I,20,"."
790 :NEXT I
800 :
810 :DO
820 :   CHAR 1,5,0,"":PRINT"FALSCH:"F" RICHTIG:"R
830 :  CHAR 1,8,20,CHR$(18)+">>"+CHR$(146)
840 :  GETKEY BR$:IF BR$<"A"OR BR$>"Z"THEN LOOP:ELSE PRINT BR$;
850 IF INSTR(MK$,BR$)<>0THEN LOOP:ELSE MK$=MK$+BR$
860 :  IF INSTR(F$,BR$)=0 THEN EXIT
870 ::    FOR I=1TO FL
880 ::       IF MID$(F$,I,1)=BR$ THEN CHAR ,12+I,20,BR$:R=R+1
890 ::    NEXT I
900 :  IF R=FL THEN 1110
910 LOOP
920 :
930 :
940 REM ------------------------------
950 REM ---      GALGEN BAUEN        ---
960 REM ------------------------------
970 :
980 DO
990 :  READ X,Y
1000 :  IF X=-1 AND Y=-1THEN EXIT:ELSE READ GA$
1010 :  IF GA$="←"THEN GA$=RV$
1020 :  CHAR 1,X,Y,GA$
1030 LOOP
1040 F=F+1:IF F>11THEN 1230
1050 GOTO 810
1060 :
1070 REM ------------------------------
1080 REM ---      SPIEL GEWONNEN      ---
1090 REM ------------------------------
1100 :
1110 SCNCLR
1120 CHAR,10,10,"■■ S U P E R ■■■"
1130 CHAR,12,12,"NOCHMAL [J/N]?"
1140 FORI=900 TO 1015 STEP 10
1150 SOUND 1,I,6:NEXT
1160 GETKEY A$:IF A$<>"N" THEN RUN
1170 SCNCLR:END
1180 :
1190 REM ------------------------------
1200 REM ---     SPIEL VERLOREN       ---
1210 REM ------------------------------
1220 :
1230 CHAR,12,0,"":PRINTF
```

```
1240 FOR I=27 TO 35
1250 :   CHAR,I,9,RV$
1260 NEXT I
1270 :
1280 FOR I=6TO 16
1290 :   CHAR,31,I,RV$
1300 NEXT I
1310 :
1320 CHAR 1,4,18,"DAS GESUCHTE WORT WAR:"+F$
1330 CHAR1,13,24,"NOCHMAL [J/N]?"
1340 FOR I=200 TO 0 STEP -7
1350 SOUND 1,I,7:NEXT
1360 GETKEY A$:IF A$<>"N"THEN RUN
1370 SCNCLR:END
1380 :
1390 REM ------------------------------
1400 REM ---   HANG MAN'S GALGEN    ---
1410 REM ------------------------------
1420 :
1430 DATA9,15,"──────┘",-1,-1
1440 DATA17,14,"│"
1450 DATA17,13,"│"
1460 DATA17,12,"│"
1470 DATA17,11,"│"
1480 DATA17,10,"│"
1490 DATA17,9,"│"
1500 DATA17,8,"│"
1510 DATA17,7,"│"
1520 DATA17,6,"│",-1,-1
1530 DATA17,5,"┐"
1540 DATA11,5,"──────",-1,-1
1550 DATA15,15,"┘"
1560 DATA16,14,"/"
1570 DATA17,13,"┘",-1,-1
1580 DATA15,5,"┐"
1590 DATA16,6,"\"
1600 DATA17,7,"┐",-1,-1
1610 DATA9,7," ⌒"
1620 DATA9,8,"(  )"
1630 DATA9,9," ⌣",-1,-1
1640 DATA10,10,"←"
1650 DATA10,11,"←"
1660 DATA10,12,"←",-1,-1
1670 DATA9,13,"/"
1680 DATA8,14,"/",-1,-1
1690 DATA11,13,"\"
1700 DATA12,14,"\",-1,-1
1710 DATA9,10,"/"
1720 DATA8,11,"/",-1,-1
1730 DATA11,10,"\"
1740 DATA12,11,"\",-1,-1
1750 DATA10,5,"│"
1760 DATA10,6,"│",-1,-1
1770 :
1780 REM ------------------------------
1790 REM ---       DIE WOERTER        ---
1800 REM ------------------------------
```

```
1810 :
1820 DATA"BONN"
1830 DATA"MAUS"
1840 DATA"WELT"          :REM 4  BUCHSTABEN
1850 DATA"HANS"
1860 DATA"BUCH"
1870 :
1880 DATA"BASIC"
1890 DATA"STIFT"
1900 DATA"NATUR"         :REM 5  BUCHSTABEN
1910 DATA"HUNDE"
1920 DATA"KNALL"
1930 :
1940 DATA"FRISCH"
1950 DATA"TASCHE"
1960 DATA"GRAFIK"        :REM 6  BUCHSTABEN
1970 DATA"FLOPPY"
1980 DATA"LAUFEN"
1990 :
2000 DATA"POLIZEI"
2010 DATA"SPIELEN"
2020 DATA"LOGISCH"       :REM 7  BUCHSTABEN
2030 DATA"AMERIKA"
2040 DATA"LISTING"
2050 :
2060 DATA"SCHLUMPF"
2070 DATA"COMPUTER"
2080 DATA"ENGLISCH"      :REM 8  BUCHSTABEN
2090 DATA"JOYSTICK"
2100 DATA"BIRNBAUM"
2110 :
2120 DATA"BAUERNHOF"
2130 DATA"COMMODORE"
2140 DATA"MIKROSKOP"     :REM 9  BUCHSTABEN
2150 DATA"ASSEMBLER"
2160 DATA"FERNSEHER"
2170 :
2180 DATA"BUNDESWEHR"
2190 DATA"AUTORENNEN"
2200 DATA"DATABECKER"    :REM 10 BUCHSTABEN
2210 DATA"KOCHREZEPT"
2220 DATA"RADDAMPFER"
2230 :
2240 DATA"SCHULPAUSEN"
2250 DATA"FLUGBETRIEB"
2260 DATA"ANGELSCHEIN"   :REM 11 BUCHSTABEN
2270 DATA"SCHNELLKOCH"
2280 DATA"EISENBAHNEN"
2290 :
2300 DATA"TASCHENDIEBE"
2310 DATA"LASERDRUCKER"
2320 DATA"HUBSCHRAUBER"  :REM 12 BUCHSTABEN
2330 DATA"STROMLEITUNG"
2340 DATA"SIEGERPODEST"
```

```
100 REM ***--------------------------***
110 REM **----------------------------**
120 REM *-                            -*
130 REM --           SQUASH           --
140 REM --                            --
150 REM --                (TF/03/86)  --
160 REM *-                            -*
170 REM **----------------------------**
180 REM ***--------------------------***
190 :
200 COLOR0,4
210 COLOR4,6
220 COLOR1,7
230 VOL 5
240 SCNCLR
250 FOR I =0 TO 39
260 : CHAR,I,0,CHR$(18)+" "+CHR$(146)
270 NEXT
280 :
290 COLOR1,6
300 FORI=0TO23
310 : CHAR,39,I,CHR$(18)+" "+CHR$(146)
320 NEXT
330 :
340 SL$=" ▒▒          ▒ "
350 BL$="▒●"
360 SL=LEN(SL$)-3
370 SP=16:REM SCHLAEGERPOSITION
380 X=INT(RND(1)*38)+1:REM ERSTE X-POSITION BALL
390 Y=INT(RND(1)*20)+1:REM ERSTE Y-POSITION BALL
400 FX=-1.5:FY=-1
410 CHAR 1,SP,22,SL$
420 CHAR 1,X,Y,BL$
430 FORI=1000 TO 10 STEP -30:SOUND1,I,2:NEXT
440 :
450 DO
```

```
460 :    CHAR,10,0,"":PRINTCHR$(31);CHR$(18);P;CHR$(146)
470 :    GET T$
480 :    IF T$="A"THEN SP=SP-1
490 :    IF T$="L"THEN SP=SP+1
500 :    IF SP<0 THEN SP=0
510 :    IF SP>39-SL THEN SP=39-SL
520 :    CHAR,SP,22,SL$
530 REM                              *** BALLBEWEGUNG ***
540 :    CHAR 1,X,Y," "
550 :    IF X+FX<0THEN FX=-FX:SOUND1,RND(1)*500,5
560 :    IF X+FX>38THEN FX=-FX:SOUND1,RND(1)*500,5
570 :    IF Y+FY<1THEN FY=-FY:SOUND1,RND(1)*500,5
580 :    IF Y+FY>21THEN GOTO 670
590 :    X=X+FX:Y=Y+FY
600 :    CHAR 1,X,Y,BL$
610 LOOP
620 :
630 REM"------------------------------
640 REM"--- BALL NICHT GETROFFEN ??---
650 REM"------------------------------
660 :
670 IF SP<X+FX AND SP+SL>X+FX THENFY=-FY:P=P+1:SOUND1,RND(1)*500,5:GOTO590
680 F=F+1:IF F<3 THEN 380
690 :
700 REM ------------------------------
710 REM ---      SPIEL BEENDEN       ---
720 REM ------------------------------
730 :
740 SCNCLR
750 FORI=10TO900 STEP10 :SOUND 1,I,1:NEXT
760 POKE239,0
770 CHAR,8,10,"G A M E  O V E R "
780 CHAR,9,12,"TRY AGAIN Y/N?"
790 GETKEY A$:IF A$<>"N"THEN RUN
800 SCNCLR:END
```

Der PRINT USING-Befehl

Format: PRINT [#fn] USING v$;au(;)

Parameter:

- fn: Nummer der Ausgabedatei
  1-127:Carriage return
  128-255:Angefügter Zeilenvorschub (Linefeed).
  Für die normale Bildschirmausgabe ist dieser Parameter nicht nötig.
- v$: Definiert das Ausgabeformat
- au: Ausdrücke, die formatiert ausgegeben werden sollen. Wenn mehrere Ausdrücke behandelt werden sollen, so werden sie mit Kommata oder Semikolon getrennt. Hierbei ist zu beachten, daß ein Komma keine Tabulierung bewirkt, wie dies beim normalen PRINT-Befehl der Fall ist.

Beispiel: PRINT USING "#,###.##";a;b;c

Funktion: Formatierte Ausgabe eines oder mehrerer Ausdrücke.

Erläuterungen:

Die Definition des Ausgabebildes erfolgt mit Hilfe verschiedener Zeichen, die man beherrschen sollte:

String-Ausdrücke:

- # : Dieses Zeichen steht für ein Zeichen im Ausgabebild. Hat der zu behandelnde Ausdruck mehr Zeichen als diejenigen Stellen, die mit # definiert sind, so werden die überzähligen Zeichen bei der Ausgabe nicht berücksichtigt. Ansonsten erfolgt die Ausgabe linksbündig.

= : Die Ausgabe erfolgt zentriert.

> : Die Ausgabe erfolgt rechtsbündig.

Numerische Ausdrücke:

: Zeichen des Ausgabebildes. Die Ausgabe erfolgt rechtsbündig. Nicht benötigte Stellen werden mit Leerzeichen ausgefüllt. Wenn weniger |-Zeichen als Stellen im eingegebenen Ausdruck angegeben werden, so wird das Ausgabebild mit *-Zeichen gestaltet (siehe untenstehendes Beispiel).

+,- : Eines dieser Zeichen bewirkt eine Vorzeichenangabe bei der Ausgabe. Die Zeichen können entweder am Anfang oder Ende der Ausgabedefinition stehen (siehe untenstehendes Beispiel).

. : Dieser Punkt legt die Dezimalpositionen fest. Sind weniger Stellen als in der angebenen Zahl vorhanden, so wird die Zahl gerundet. Dieses Zeichen darf außerdem nur einmal in der Definition verwendet werden (siehe untenstehendes Beispiel).

, : Dient zur besseren Lesbarkeit von größeren Zahlen. Im Ausgabebild erscheint ein Komma, was nicht mit dem Dezimalkomma verwechselt werden darf. Außerdem muß vor dem ,-Zeichen in der Definition ein #-Zeichen stehen (siehe untenstehendes Beispiel).

$: Vor der ersten Zahl im Ausgabebild wird ein Dollar-Zeichen gesetzt. Auch hier muß vor diesem Zeichen in der Definition ein #-Zeichen stehen. Wie aus dem PUDEF-Befehl ersichtlich wird, bedeutet dieses Zeichen, daß es möglich ist, der formatierten Ausgabe ein beliebiges Zeichen voranzustellen. Ist dieses Zeichen noch nicht umdefiniert, so benutzt der Computer das Dollar-Zeichen (siehe untenstehendes Beispiel).

^^^^ : Erwirkt eine Ausgabe im Exponentialformat. Nach diesen vier Zeichen muß noch ein + oder - in der Definition folgen. Das Exponentialformat ist, wie mathematisch versierte Leser sicher wissen, eine besondere Schreibweise von Zahlen. So wird der Dezimalpunkt hinter das erste Zeichen gesetzt. Die Anzahl der Stellen, die hinter der ersten Zahl existieren, wird extra ausgedrückt. Hierdurch wird die Schreibweise mancher Zahlen wesentlich kürzer.

Beispiel:

```
Normalformat          Exponentialformat

100000000000          1.0 E + 11
                oder    1 E + 11
```

Doch auch kleinere Zahlen lassen sich so ausdrücken. Im Gegensatz zu der geraden benutzten Schreibweise wird das + durch ein - ersetzt. Wie Sie sicher schon festgestellt haben, wird bei der Benutzung des + der Dezimalpunkt nach rechts verschoben, um die alte Schreibweise zu erlangen. Jetzt ist es umgekehrt:

```
Normalformat          Exponentialformat
0.00000000001         1.0 E - 11
                oder    1 E - 11
```

Zur besseren Verständlichkeit nun noch ein Beispiel:

```
10 x=12345.12345
20 v$="+#$##,###,###.###"
30 for i=1 to 10
40 print using v$;x
50 v$=left$(v$,len(v$)-1)
60 next i
```

Der PUDEF-Befehl

Format: PUDEF v$

Parameter: - v$ Ein Stringausdruck, mit der Länge von höchstens 4 Zeichen. Jede Position stellt ein bestimmtes Steuerzeichen im PRINT USING-Befehl dar.

Dabei gilt:

Pos 1:Füllzeichen
Pos 2:Komma
Pos 3:Dezimalpunkt
Pos 4:Dollarzeichen

Funktion: Definiert bis zu 4 Steuerzeichen im PRINT USING-Befehl neu.

Erläuterungen:

Als Füllzeichen ist das Leerzeichen voreingestellt. Sonst jeweils das angegebene. Also wird der Dezimalpunkt auch als Punkt ausgegeben.

Beispiel: PUDEF " ., "
Funktion: Stellt die englische Schreibweise von Zahlen auf die deutsche um (Vertauschen von Punkt und Komma).

Beispiel: PUDEF "- "
Funktion: Das Schriftbild wird von links mit "-"-Zeichen anstatt mit Leerzeichen aufgefüllt.

1.1.3 Shape-Grafik

Die folgenden Befehle befassen sich mit der Shapegrafik. Shapes sind eigene kleine Grafiken, welche als Auszüge der hochauflösenden oder der multicolor Grafik in Stringvariablen gespeichernt werden können. Leider sind sie jedoch ziemlich langsam, und daher als Action-Figuren, in Spielen z.B., völlig ungeeignet. Jedoch erfüllen sie ihre Aufgabe in Strategie-Spielen, wo es mehr auf Schönheit als auf Schnelligkeit ankommt, voll und ganz.

Der GSHAPE-Befehl

Format:		GSHAPE zk,x,y,mo
Parameter:	- zk	Variable mit binärer Bildinformation (s. SSHAPE-Befehl)
	- x,z	Bildschirmkoordinaten der linken, oberen Ecke des darzustellenden Bildes (scaliert)
	- mo	Art der Bildschirmdarstellung: 0=Darstellung genau so, wie sie gespeichert wurden. 1=Invertierte Bildschirmausgabe 2=Überlagerung der Darstellung auf dem vorhandenen Bild 3=Darstellung nur auf schon benutzten Bildpunkten 4=Schon benutzte Bildpunkte,auf die die neue Darstellung übertragen werden soll, werden invertiert.
Beispiel:		GSHAPE z$,10,10,4
Funktion:		Überträgt ein gespeichertes Shape auf den Grafik-Bildschirm.

Bemerkungen:

Diese Eigenschaft kann man, wie in der Einleitung zu diesem Kapitel schon gesagt, für mehrere Dinge benutzen. Als Anregung möchte ich zum Beispiel ein Layout-Programm geben, bei dem Sie Ihre Bauteile in diesen Stringvariablen speichern und anschließend in den Bildschirm "stempeln" können.

Der SSHAPE-Befehl

Format: SSHAPE z$,x1,y1,x2,y2

Parameter:
- z$ Zeichenkettenvariable, die den gewählten Bildschirmbereich als Binärinformation übernimmt.
- x1,y1 Koordinaten des linken, oberen Eckpunktes des gewählten Bildschirmbereichs (scaliert)
- x2,y2 Koordinaten des rechten, unteren Eckpunktes des gewählten Bildschirmbereichs (scaliert). Voreingestellt ist die aktuelle Position des grafischen Cursors.

Beispiel: SSHAPE z$,10,10,20,20

Funktion: Übergibt den Inhalt des gewählten Bildschirmbereichs der Variablen z$ in binärer Form.

Erläuterungen:

Der SSHAPE-Befehl ermöglicht es uns, den Inhalt eines Bildschirmbereichs zu "lesen" und in eine Variable zu übergeben. Hierdurch wird uns die Möglichkeit gegeben, bestimmte Grafiken zu verschieben, da ja der GSHAPE-Befehl die Umkehrung des SSHAPE-Befehls ist und den Inhalt einer Variablen in einen

Bildschirmbereich übergibt (siehe auch GSHAPE-Befehl). Es ist allerdings nicht möglich, einen beliebig großen Bildschirmbereich in eine Variable zu übergeben. Die Länge der zu übergebenden Zeichenkette beschränkt sich auf 255 Zeichen. Um Komplikationen mit der maximalen Länge zu vermeiden, kann man zwei Formeln benutzen, die die benötigte Länge berechnen:

```
Hochauflösend: l=INT((ABS(x1-x2)+1)/8+.99)*(ABS(y1-y2)+1)+4

Multicolor: l=INT((ABS(x1-x2)+1)/4+.99)*(ABS(y1-y2)+1)+4
```

Der Vorteil dieser Formeln liegt klar auf der Hand: Hat man sich einen Bildschirmbereich ausgesucht, so ist es vorteilhaft zu wissen, ob die Größe des Bereichs nicht die Möglichkeiten des Befehls übersteigt. Benutzt man die Formeln, so entgeht man den Fehlermeldungen des Computers.

```
100 rem ##############################################
110 rem ###                                        ###
120 rem ###          shape-demo                    ###
130 rem ###         --------------                 ###
140 rem ###                                        ###
150 rem ##############################################
160 :
170 color 4,1,1                :rem rahmenfarbe
180 color 0,1,1                :rem hintergrundfarbe
190 color 1,5,1                :rem zeichenfarbe
200 graphic 1,1                :rem hgr einschalten
210 :
220 m=1:for z=1 to 20
230 y1=int(199*rnd(1))
240 x1=int(319*rnd(1))
230 y2=int(199*rnd(1))
240 x2=int(319*rnd(1))
250 box m,x1,y1,x2,y1,,1       :rem ausgefüllte Quadrate
260 next
270 :
```

```
280 rem bestimmten Bereich kopieren
290 :
300 sshape a$,160,100,180,120
310 :
320 for x=0 to 320 step 30
330 gshape a$,x,30,0
340 gshape a$,x,60,1
350 gshape a$,x,90,2
360 gshape a$,x,120,3
370 gshape a$,x,150,4
380 next x
```

1.1.4 Sonstige Grafikbefehle

In diesem Kapitel werden Sie Befehle finden, die entweder bei keinem der anderen Kapitel unterzuordnen waren, oder bei denen es sich nicht gelohnt hätte, extra ein eigenes Kapitel zu schreiben.

Der COLOR-Befehl

Format: COLOR fq,fc,fl

Parameter:

- fq — Farbquelle.
  0=Hintergrund
  1=grafischer Vordergrund
  2=grafischer Mehrfarbenmodus 1
  3=grafischer Mehrfarbenmodus 2
  4=Rand

- fc — Farbcode.
  Folgende Farbcodes kann man wählen:

1=schwarz	9=hellbraun
2=weiß	10=braun
3=rot	11=rosa
4=grün	12=dunkelgrau
5=violet	13=grau

```
6=dunkelgrün 14=hellgrün
7=blau       15=hellblau
8=gelb       16=hellgrau
```

- fl Farbhelligkeit (0-7)

Beispiel: COLOR 4,8

Funktion: Definiert für jede mögliche Farbquelle eine Farbe.

Erläuterungen:

COLOR 0,fc ändert die Hintergrundfarbe des Bildschirms nur mit anschließendem GRAPHIC-Befehl. Das bedeutet, daß dieser Befehl nicht gleich die sichtbare Hintergrundfarbe ändert.
Wie schon bei der Beschreibung von anderen Befehlen angedeutet, möchte ich denen, die nichts mit den Farbquellen anfangen können, diese etwas näher bringen. Zunächst einmal dienen die Farbquellen 0, 1 und 4 dazu, ganz bestimmte Bildschirmstellen farbig zu gestalten. Hierzu folgt im Anschluß an die Definition der GRAPHIC-Befehl (siehe dort). Neben dieser Aufgabe haben die Farbquellen auch noch eine andere, sehr wichtige. Man kann diese Farbquellen bei allen Befehlen, die etwas zeichnen, abrufen und deren Inhalt als Zeichenfarbe benutzen. So ist es möglich, die Zeichenfarbe in Übereinstimmung mit der anderen, bereits benutzten, zu wählen.

Der GRAPHIC-Befehl

Format: GRAPHIC mo,ls
GRAPHIC CLR

Parameter: - mo : Modus. Folgende 5 Modi sind möglich:
0=Text mit 40 Zeichen / Zeile
1=hochauflösende Grafik (320*200 Punkte)
2=hochauflösende Grafik mit Text

3=Multicolorgrafik (160*200 Punkte)
4=Multicolorgrafik mit Text

- ls Bestimmt, ob bei Aufruf der Modi 1-4 der Bildschirm gelöscht werden soll (1) oder nicht (0).

- GRAPHIC CLR: Gibt den reservierten Speicherplatz wieder frei.

Beispiel: GRAPHIC 1,1

Funktion: Aktiviert Grafikmodus und reserviert Speicherplatz (bzw. gibt ihn wieder frei).

Erläuterungen:

Bei den Modi 1-4 wird ein 8-KByte-Grafikspeicherplatz, der für die Darstellung der Grafik notwendig ist, ab dem Bereich 8192 (hex $2000), ein 2-KByte Farb- und Helligkeits-RAM ab dem Bereich 6144 (hex $1800) reserviert und wird erst wieder durch GRAPHIC CLR freigegeben.

Beachten Sie bitte, daß nach Einschalten der Grafik beim C16 und C116 in der Grundversion nur noch ca. 2 KByte für eigene Programme zur Verfügung stehen. Auch uns wurde diese Tatsache oft zum Verhängnis. Dies ist übrigens auch der Grund, wieso die Programme unseres CAD-Kapitels nur mit mindestens 32-KByte-RAM funktionieren.

Weiterhin wird der Grafikmodus aktiviert. Außerdem werden alle Farben, die mit dem COLOR-Befehl neu definiert wurden, sichtbar.

Die RCLR-Funktion

Format: x=RCLR(fq)

Parameter: fq Farbquelle
0=Hintergrund
1=Vordergrund
2=Multicolor 1
3=Multicolor 2
4=Rand

Beispiel: x=RCLR(1)

Funktion: Liefert den aktuellen Farbcode der angegebenen Farbquelle.

Erläuterungen:

Dieser Befehl ermöglicht es uns, die Inhalte der Farbquellen abzufragen. Das ist dann nützlich, wenn in einem Programm Bedingungen an diese geknüpft werden. Dadurch ist es möglich, wenn nötig, Änderungen vorzunehmen. Nach der Ausführung dieses Befehls trägt die Variable x den abgefragten Inhalt.

Die RGR-Funktion

Format: x=RGR(dummy)

Parameter: dummy: Pseudowert, dessen Wert beliebig zwischen 0 und 255 gewählt werden kann

Beispiel: x=RGR(1)

Funktion: Liefert den aktuellen Grafikmodus

Die RLUM Funktion

Format: x=RLUM(fq)

Parameter: fq Farbquelle
0=Hintergrund

1=Vordergrund
2=Multicolor 1
3=Multicolor 2
4=Rand

Beispiel: x=RLUM(1)

Funktion: Liefert den aktuellen Helligkeitswert der angegebenden Farbquelle.

Erläuterungen:

Dieser Befehl ermöglicht es uns, die Inhalte der Farbquellen auf deren Helligkeitswerte abzufragen. Das ist dann nützlich, wenn in einem Programm Bedingungen an diese geknüpft werden. Dadurch ist es möglich, wenn nötig, Änderungen vorzunehmen. Nach der Ausführung dieses Befehls trägt die Variable x den abgefragten Inhalt.

Die SCNCLR-Anweisung

Format: SCNCLR(n)

Parameter:

"n" ist ein ganzzahliger Wert mit folgender Bedeutung:

0 - Der Textbildschirm wird gelöscht.
1 - Der hochauflösende Grafikbildschirm wird gelöscht.
2 - Der hochauflösende geteilte Bildschirm wird gelöscht.
3 - Der Mehrfarben-Grafikbildschirm wird gelöscht.
4 - Der geteilte Mehrfarben-Grafikbildschirm wird gelöscht.

Erläuterungen:

Wird "n" weggelassen, so wird der jeweils aktivierte Bildschirm gelöscht. Das Weglassen von "n" kann also entgegen der Informationen des Commodore-Handbuches in allen Modi geschehen.

2. Grundlagen der Grafikprogrammierung

2.1 Theoretische Grundlagen

Das nun folgende Kapitel ist für diejenigen unter Ihnen gedacht, denen es nicht reicht, eine Linie zum Beispiel mit den uns bereits bekannten Hilfsmitteln zu zeichnen, die vielmehr etwas genauer ergründen wollen, wie man nun jeden einzelnen Punkt einer Linie errechnet:

2.1.1 Linienberechnung

Zunächst einmal müssen wir uns, um dieses Problem zu lösen, etwas mit der analytischen Geometrie befassen. Hinter diesem monströsen Begriff steckt in diesem Zusammenhang betrachtet, nichts anderes als die normierte Geradengleichung, von der Sie ja bestimmt auch schon einmal in der Schulmathematik gehört haben. Diese Gleichung hat die folgende Form:

$$y = mx + n$$

x und y sind hierbei die Koordinaten eines beliebigen Punktes der Geraden. m ist die Steigung der Geraden und n stellt ihren Schnittpunkt mit der y-Achse dar.

Sind nun zwei Punkte dieser Gerade bekannt, es wären ja in diesem Fall die Endpunkte unserer Linie (x1,y1) und (x2,y2), so können wir sofort die zwei nachstehenden Formeln ableiten, die wir anschließend gleichsetzen.

$$n = y1 - mx1 \quad \text{und zugleich} \quad n = y2 - mx2$$

$$\rightarrow \; y1 - mx2 = y2 - mx2$$

$$\Leftrightarrow \qquad m = \frac{y2 - y1}{x2 - x1}$$

Die letzte Formel ist die sogenannte Punkt-Steigungs-Formel und erlaubt uns, die Steigung m einer Geraden mit den zwei auf dieser Geraden liegenden Punkten (x1,y1) und (x2,y2) zu bestimmen. n gibt den Schnittpunkt der Geraden mit der y-Achse an, also die "Höhe" der Geraden. Wählen wir n=0, so geht die Gerade durch den Ursprung mit den Koordinaten (0,0). Verschieben wir diesen Startpunkt der Linie zu ihrem Endpunkt, so müssen wir entsprechend die beiden Koordinaten, in diesem Fall x2 und y2, zu X und Y addieren. Die folgende Formel gibt dann nun die endgültige Geradengleichung wieder, die bereits die verschobene Gerade angibt und in die m eingesetzt wurde:

$$y = \frac{y2 - y1}{x2 - x1} * (x - x2) + y2$$

Diese Formel ist die Grundlage des unten dargestellten Programms und wird stückweise in den Zeilen 380, 480 und 500 errechnet, wobei die X-Koordinate stets von x2 nach x1 läuft, und für jeden solchen X-Wert der entsprechnde Y-Wert bestimmt wird.

Ein einziges Problem, das diese Formel mit sich bringt, existiert jedoch, wenn wir eine Senkrechte zeichnen. In diesem Fall wird x1=x2 und damit der Nenner der Steigung gleich 0, was uns unser Computer mit einem "DIVISION BY ZERO ERROR" quittiert. Wir umgehen diesen Sonderfall, indem wir in der Unterroutine, die in Zeile 310 beginnt, eine Senkrechte zeichnen.

Natürlich kommt, wie Sie selbst nun festgestellt haben, ein BASIC-Programm noch lange nicht mit einer gleichwertigen Assemblerroutine mit, die wir im folgenden Kapitel vorstellen wollen.

```
100 rem ************************************************
110 rem ***                                          ***
120 rem ***         zeichnen einer linie             ***
130 rem ***       ----------------------             ***
140 rem ***                                          ***
150 rem ************************************************
160 :
170 color 4,1,4
180 color 0,1
190 color 1,6,4
200 :
210 graphic 1,1
220 x1=110:y1=120
230 x2=130:y2=140
240 :
250 gosub 380
260 :
270 getkey a$
280 graphic 0
290 list
300 end
310 :
320 rem ************************************
330 rem ***                                ***
340 rem *** unteroutine "linie zeichnen" ***
350 rem ***                                ***
360 rem ************************************
370 :
380 dy=y2-y1 : dx=x2-x1
390 yk=y2 : xk=x2
400 :
410 if dx<>0 then 480
420 for yk=y2 to y1 step sgn(-dy)
430 gosub 560
440 next yk
450 return
460
470
480 dd=dy/dx
490 for xk=x2 to x1 step sgn(-dx)
```

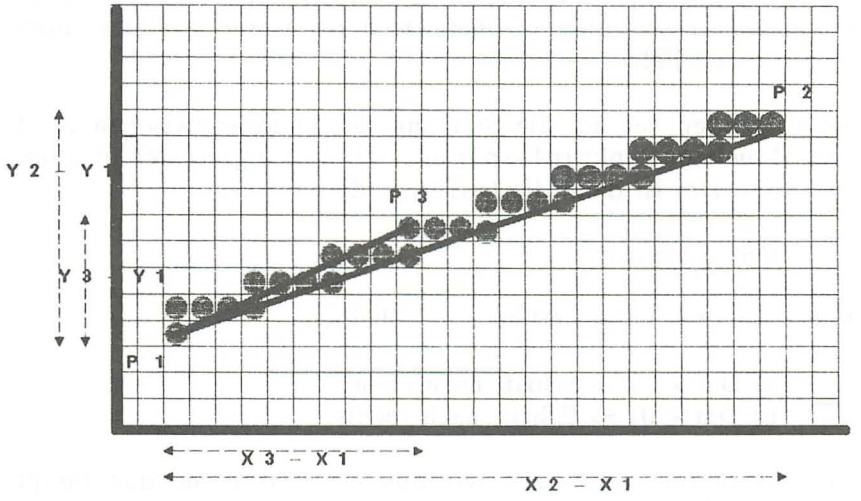
```
500 :  zk=int(dd*(xk-x2)+y2)
510 :  do while zk<>yk
512 ::     yk=yk+sgn(-dy)
514 ::     gosub 560
516 :  loop
520 :  gosub 560
530 next xk
540 return
550 :
560 draw ,xk,yk : xk=xk+1
570 draw ,xk,yk : xk=xk-1
580 return
```

2.1.2 Linienberechnung in Assembler

Wenn Sie das vorstehende Programm des öfteren für Ihre Linien und Zeichnungen verwenden wollen, so wird Ihnen die lange Wartezeit, die zum Zeichnen benötigt wird, recht bald unbequem. In diesem Fall sollte uns ein Blick hinter die Kulissen einer in Maschinensprache (Assembler) geschriebenen Routine nicht zu schade sein. Sie werden staunen, welchen Geschwindigkeitsvorteil eine solche Assemblerroutine bringen wird.

Gehen wir die Sache einmal ganz ruhig an: Natürlich werden wir hier versuchen, einen Algorithmus, also eine Rechenvorschrift, zu finden, in der nicht so viel zeitintensiv herumgerechnet werden muß. Dazu sehen wir uns einmal an, wie eine sogenannte Linie in Wirklichkeit auf dem Bildschirm ausschaut (s. Abbildung). Was uns da als schöne, gleichmäßige Linie erscheint, nimmt in Wahrheit einen recht wilden Zickzackverlauf. Wir werden nun nicht die Koordinaten jedes Punktes einzeln errechnen, sondern lediglich feststellen, in welche Richtung (hoch, runter, rechts oder links) der nächste Punkt gezeichnet werden soll. Dadurch muß jedesmal nur eine einfache Addition durchgeführt werden.

Linie in der hochauflösenden Grafik



Wie wir oben gesehen haben, können wir die Steigung einer Geraden durch die Formel m=(y2-y1)/(x2-x1) berechnen. Nehmen wir an, wir befinden uns während des Zeichnens unserer Linie gemäß obiger Abbildung im Punkt p3. Wie deutlich zu sehen ist, befindet sich dieser Punkt nicht auf, sondern vielmehr "über" unserer Ideallinie. Die Steigung der Geraden p1 nach p3 ist also größer, als die Idealgerade p1 nach p2. Aus diesem Grunde müssen wir den nächsten Punkt rechts neben dem zuletzt gezeichneten Punkt setzen, da sich nur so im nächsten Schritt die Steigung von p1->p3 wieder an die der Gerade p1->p2 annähert.

Wir können nun also unser Gesetz formulieren: Ist die Steigung der Geraden zwischen Start- und aktuellem Punkt größer als die der Idealgeraden, so zeichnen wir den nächsten Punkt rechts ne-

ben den vorherigen, ist sie dagegen kleiner oder gleich, so wird der nächste Punkt darüber gezeichnet. Dies gilt natürlich nur dann, wenn man grundsätzlich von unten nach oben eine Linie mit positiver Steigung zeichnet. Doch realisieren wir erst einmal diesen einen Fall.

Im folgenden sei ma die Steigung der Gerade zwischen Start- und aktuellem Punkt (p1->p3) und mi die Steigung der Ideallinie (p1->p2). Dm ist dann die Differenz dieser Werte:

Dm = mi-ma

Folgende zwei Fälle sind zu unterscheiden:

a) Dm > 0 => Schritt nach oben
b) Dm < 0 => Schritt nach rechts

Für die Steigungen setzen wir nun die entsprechenden Punkt-Steigungs-Formeln (s.o.) ein:

```
ma = (y3-y1)/(x3-x1)
mi = (y2-y1)/(x2-x1)
```

Dann ergibt sich:

```
Dm                = (y2-y1)/(x2-x1)-(y3-y1)/(x3-x1)  <=>
Dm*(x3-x1)*(x2-x1) = (y2-y1)*(x3-x1)-(y3-y1)*(x2-x1)
```

Da x3 und x2 immer größer sind als x1, ist auch der Ausdruck (x3-x1)*(x2-x1) immer größer Null, ändert also nichts an dem Vorzeichen der linken Seite der Gleichung. Da aber nun interessant ist, ob Dm größer oder kleiner Null ist, kann man die beiden Klammern auf der linken Gleichungsseite getrost wegfallen lassen und wir erhalten wieder:

Dm = (y2-y1)*(x3-x1) - (y3-y1)*(x2-x1)

Streng mathematisch ist diese Gleichung natürlich falsch. Gleich sind natürlich nur noch die Vorzeichen der beiden Gleichungsseiten. Für unsere Zwecke geht das aber in Ordnung.

Bewegen wir uns nun nach rechts, so erhöht sich x3, also (x3-x1), entsprechend um 1, bewegen wir uns dagegen nach oben, so erhöht sich y3, also (y3-y1), um 1. Das neue Dm (Kurz NDm) sieht für den ersten Fall dann so aus:

```
NDm = (y2-y1)*((x3-x1)+1) - (y3-y1)*(x2-x1)          <=>
NDm = (y2-y1)*(x3-x1) + (x3-x1) - (y3-y1)*(x2-x1)    <=>
NDm = Dm + (x3-x1)
```

Entsprechend für den zweiten Fall:

```
NDm = (y2-y1)*(x3-x1) - ((y3-y1)+1)*(x2-x1)          <=>
NDm = (y2-y1)*(x3-x1) - (y3-y1)*(x2-x1) - (y3-y1)    <=>
NDm = Dm - (y3-y1)
```

Damit kann jedes NDm durch eine einfache Addition aus dem alten Dm errechnet werden. Dieser neue Wert wird dann auf sein Vorzeichen getestet, um wiederum zu entscheiden, in welche Richtung nun gezeichnet werden soll, die Koordinaten werden erhöht, das neue Dm wird errechnet usw.

Bisher haben wir diese Ableitung nur für Geraden mit positiver Steigung dargestellt. Bei Geraden mit m<0 (negative Steigung), also y3 kleiner y1, gelten die gleichen Regeln und Formeln, wenn man die Vorzeichen von (y2-y1) und (x2-x1) umkehrt, also den Betrag dieser Werte berechnet. Gleichzeitig muß natürlich statt nach rechts nach links und statt nach oben nach unten gezeichnet werden.

Dieses ganze Verfahren wird solange wiederholt, bis x3=x2 und y3=y2. Um nicht dauernd diese Differenz zu berechnen, wird einfach die Anzahl der zu zeichnenden Punkte anz gezählt:

```
anz= abs(x2-x1) + abs(y2-y1)
```

abs ist die Absolutfunktion, d.h. Vorzeichen werden weggestrichen. Ist die Anzahl der zu zeichnenden Punkte erreicht, ist die Linie fertig.

Doch nun nach aller grauen Theorie hier das Assemblerprogramm mit BASIC-Lader und kleiner Demo. Die Funktion wird mit SYS linie,x1,y1,x2,y2 aufgerufen.

profi-ass 65c02 v1

```
110:  1630                .opt o1
120:              ;
130:  1630                *=   $1630
140:              ;
150:              ;
160:              ; ************************
170:              ; **                    **
180:              ; **  linien in assembler **
190:              ; **                    **
200:              ; ************************
210:              ;
220:              ;
230:              ;
240:
240:  1630
250:              ;*******************
260:              ;
270:  9d84        getbyt   =   $9d84   ;holt bytewert
280:  9491        chkcom   =   $9491   ;prueft auf komma
290:  9dd8        chkget   =   $9dd8   ;chkcom+getbyt
300:  9dd2        getcor   =   $9dd2   ;holt koordinaten
310:  9d7e        qerr     =   $9d7e   ;illegal quantity
320:              ;
330:              ;nullseitige register
330:  1630
340:              ;*******************
350:              ;
360:  0063        offx     =   $63
370:  00ab        msk      =   $ab     ;maske
380:  0069        dif0     =   $69
390:  006a        dif1     =   $6a
400:  006b        dif2     =   $6b
410:  006c        dif3     =   $6c
```

```
420:   006d          dif4     =   $6d
430:   006e          dif5     =   $6e
440:   006f          zwn      =   $6f     ;xk/8 (hline)
450:   0070          za       =   $70     ;zaehler (hline)
460:   00ac          a        =   $ac     ;punktadresse
470:   00ad          b        =   $ac+1
480:   0014          xk       =   $14     ;x-koordinate
490:   0097          flg      =   $97     ;unplot/plot-flag
500:   00fd          use      =   $fd     ;diverses
510:                 ;
520:                 ;feste werte
520:   1630
530:                 ;************
540:                 ;
550:   2000          graph    =   $2000   ;graphikstart
560:   0021          gstart   =   $21     ;gr-start+1 h-byte
570:   003e          gend     =   $3e     ;gr-ende-1 h-byte
580:                 ;
590:                 ;
600:                 ;Ansteueradressen
600:   1630
610:                 ;*****************
620:                 ;
630:   1630 4c 3f 16          jmp  plot    ;punkt setzen
640:   1633 4c 3c 16          jmp  unplot  ;punkt loeschen
650:   1636 4c 52 16          jmp  sline   ;linie zeichnen
660:   1639 4c 4f 16          jmp  clline  ;linie loeschen
670:                 ;
680:                 ;
690:                 ;punkt loeschen
690:   163c
700:                 ;**************
710:                 ;
720:   163c a2 00    unplot   ldx  #$00    ;loesch-flag
730:   163e 2c                .byt $2c
740:                 ;
750:                 ;
760:                 ;punkt setzen
760:   163f
770:                 ;*************
```

```
780:                    ;
790:  163f a2 80        plot    ldx  #$80    ;setz-flag
800:  1641 86 97        pl1     stx  flg
810:  1643 20 91 94             jsr  chkcom
820:  1646 20 60 16             jsr  tescor  ;koordinaten holen
830:  1649 20 7b 16             jsr  hposn   ;adresse errechnen
840:  164c 4c c9 16             jmp  plt     ;punkt setzen/loeschen
850:                    ;
860:                    ;
870:                    ;linie loeschen
870:  164f
880:                    ;***************
890:                    ;
900:  164f a2 00        clline  ldx  #$00    ;loesch-flag
910:  1651 2c                   .byt $2c
920:                    ;
930:                    ;
940:                    ;linie zeichnen
940:  1652
950:                    ;***************
960:                    ;
970:  1652 a2 80        sline   ldx  #$80    ;setz-flag
980:  1654 20 41 16             jsr  pl1     ;ersten punkt setzen
990:  1657 20 91 94             jsr  chkcom
1000: 165a 20 60 16             jsr  tescor  ;zweite koord. holen
1010: 165d 4c 3f 17             jmp  hline   ;linie zeichnen
1020:                   ;
1030:                   ;
1040:                   ;koordinaten testen
1040: 1660
1050:                   ;*******************
1060:                   ;
1070: 1660 20 d2 9d tescor  jsr  getcor  ;holen
1080: 1663 8a               txa
1090: 1664 a8               tay
1100: 1665 a6 15            ldx  xk+1
1110: 1667 c0 c8            cpy  #200    ;y-koord. >= 200print
1120: 1669 b0 0d            bcs  illff   ;ja!
1130: 166b a5 14            lda  xk
1140: 166d e0 01            cpx  #>320   ;x-koord. >= 320print
```

```
1150:  166f 90 06          bcc  t1      ;ja
1160:  1671 d0 05          bne  illff   ;a-> xk-low
1170:  1673 c9 40          cmp  #<320   ;x-> xk-high
1180:  1675 b0 01          bcs  illff   ;y-> yk
1190:  1677 60       t1    rts
1200:  1678 4c 7e 9d illff jmp  qerr    ;illegal quantity
1210:                ;
1220:                ;
1230:                ;adresse errechnen
1230:  167b
1240:                ;******************
1250:                ;
1260:  167b 8c c9 17 hposn sty  yk       ;y-k
1270:  167e 8d c7 17       sta  xkl      ;x-kl
1280:  1681 8e c8 17       stx  xkh      ;x-kh (zwischenspeichern)
1290:  1684 85 14          sta  xk
1300:  1686 86 15          stx  xk+1
1310:  1688 98             tya
1320:  1689 4a             lsr  a
1330:  168a 4a             lsr  a
1340:  168b 4a             lsr  a        ;int(y/8)
1350:  168c aa             tax
1360:  168d bd cc 17       lda  mul.h,x  ;320*int(y/8) (high-byte)
1370:  1690 85 ad          sta  b
1380:  1692 8a             txa
1390:  1693 29 03          and  #3       ;bits 0+1 isolieren
1400:  1695 aa             tax
1410:  1696 bd e6 17       lda  mul.l,x  ;320*int(y/8) (low-byte)
1420:  1699 85 ac          sta  a
1430:  169b 98             tya           ;y-koord.
1440:  169c 29 07          and  #7       ;(y and 7)
1450:  169e 18             clc
1460:  169f 65 ac          adc  a        ;offy=320*int(y/8)+(y and 7)
1470:  16a1 85 ac          sta  a        ;****************************
1480:  16a3 a5 14          lda  xk
1490:  16a5 29 f8          and  #$f8     ;offx=8*int(x/8)
1500:  16a7 85 63          sta  offx     ;***************
1510:  16a9 a9 20          lda  #>graph  ;graphikseite
1520:  16ab 05 ad          ora  b
1530:  16ad 85 ad          sta  b        ;+sa
```

```
1540:  16af 18              clc
1550:  16b0 a5 ac           lda  a
1560:  16b2 65 63           adc  offx     ;ad = offy + offx + sa
1570:  16b4 85 ac           sta  a        ;**********************
1580:  16b6 a5 ad           lda  b
1590:  16b8 65 15           adc  xk+1
1600:  16ba 85 ad           sta  b
1610:  16bc a5 14           lda  xk
1620:  16be 29 07           and  #7
1630:  16c0 49 07           eor  #7       ;7-(x and 7)
1640:  16c2 aa              tax
1650:  16c3 bd ea 17        lda  msktab,x ;maskentabelle
1660:  16c6 85 ab           sta  msk      ;2~(7-(x and 7))
1670:  16c8 60              rts
1680:                 ;
1690:                 ;
1700:                 ;punkt plotten
1700:  16c9
1710:                 ;**************
1720:                 ;
1730:  16c9 a0 00     plt   ldy  #0
1740:  16cb 08              php           ;carry-flag retten
1750:  16cc a5 ab           lda  msk      ;maske
1760:  16ce 24 97           bit  flg      ;plot/unplot-flag
1770:  16d0 30 05           bmi  pl2      ;plot
1780:  16d2 49 ff           eor  #$ff     ;unplot
1790:  16d4 31 ac           abd  (a),y
1800:  16d6 2c              .byt $2c      ;überspringe nächsten befehl
1810:  16d7 11 ac     pl2   ora  (a),y    ;plot
1820:  16d9 91 ac           sta  (a),y
1830:  16db 28              plp           ;carry-flag
1840:  16dc a4 6f           ldy  zwn
1850:  16de 60              rts
1860:                 ;
1870:                 ;
1880:                 ;
1890:                 ;vektorroutinen
1890:  16df
1900:                 ;***************
1910:                 ;
```

```
1920:  16df a5 ac    unten    and  #7
1940:  16e3 c9 07             cmp  #7
1950:  16e5 f0 05             beq  un1     ;packenrand!
1960:  16e7 38                sec
1970:  16e8 a9 00             lda  #0
1980:  16ea b0 04             bcs  un2     ;addiere 1 (c=1!)
1990:  16ec a9 38    un1      lda  #$38    ;addiere 320-7=313
2000:  16ee e6 ad             inc  b       ;c=1!
2010:  16f0 65 ac    un2      adc  a
2020:  16f2 85 ac             sta  a
2030:  16f4 a9 00             lda  #0
2040:  16f6 65 ad             adc  b
2050:  16f8 85 ad             sta  b
2060:  16fa 60                rts
2070:                ;
2080:  16fb 30 e2    u.o      bmi  unten
2090:                ;
2100:  16fd a5 ac    oben     lda  a       ;adresse
2110:  16ff 29 07             and  #7
2120:  1701 f0 05             beq  ob1     ;packenrand (oben)
2130:  1703 18                clc
2140:  1704 a9 ff             lda  #$ff
2150:  1706 90 04             bcc  ob2     ;subtrahiere 1
2160:  1708 a9 c7    ob1      lda  #$c7    ;subtahiere 320+7=327
2170:  170a c6 ad             dec  b
2180:  170c 65 ac    ob2      adc  a
2190:  170e 85 ac             sta  a
2200:  1710 a5 ad             lda  b
2210:  1712 e9 00             sbc  #0
2220:  1714 85 ad             sta  b
2230:  1716 60                rts
2240:                ;
2250:                ;
2260:  1717 46 ab    rechts   lsr  msk     ;maske verschieben
2270:  1719 90 0e             bcc  re2     ;noch innerhalb byte
2280:  171b 66 ab             ror  msk     ;ausserhalb byte
2290:  171d a5 ac             lda  a
2300:  171f c8                iny
2310:  1720 18                clc
2320:  1721 69 08             adc  #8
```

```
2330:  1723 85 ac          sta  a
2340:  1725 90 02          bcc  re2
2350:  1727 e6 ad          inc  b     ;8 addieren
2360:  1729 60       re2   rts
2370:                ;
2380:  172a 10 eb    r.l   bpl  rechts
2390:                ;
2400:  172c 06 ab    links asl  msk
2410:  172e 90 0e          bcc  li1
2420:  1730 26 ab          rol  msk
2430:  1732 a5 ac          lda  a
2440:  1734 88             dey
2450:  1735 38             sec
2460:  1736 e9 08    li3   sbc  #8
2470:  1738 85 ac          sta  a
2480:  173a b0 02          bcs  li1
2490:  173c c6 ad          dec  b     ;8 subtrahieren
2500:  173e 60       li1   rts
2510:                ;
2520:                ;
2530:                ;linie zeichnen
2530:  173f
2540:                ;***************
2550:                ;
2560:  173f 48       hline pha        ;a  -> x2-low-byte
2570:  1740 ad c8 17       lda  xkh   ;x  -> x2-high-byte
2580:  1743 4a             lsr  a     ;y  -> y2
2590:  1744 ad c7 17       lda  xkl   ;xkl-> x1-low-byte
2600:  1747 6a             ror  a     ;xkh-> x1-low-byte
2610:  1748 4a             lsr  a     ;yk -> y1
2620:  1749 4a             lsr  a
2630:  174a 85 6f          sta  zwn   ;x1 / 8 zwischensp.
2640:  174c 68             pla
2650:  174d 48             pha
2660:  174e 38             sec
2670:  174f ed c7 17       sbc  xkl
2680:  1752 48             pha
2690:  1753 8a             txa
2700:  1754 ed c8 17       sbc  xkh
2710:  1757 85 6c          sta  dif3  ;x2-x1
```

```
2720:  1759 b0 0a          bcs  l3     ;negativprint
2730:  175b 68             pla         ;ja
2740:  175c 49 ff          eor  #$ff
2750:  175e 69 01          adc  #1
2760:  1760 48             pha
2770:  1761 a9 00          lda  #0
2780:  1763 e5 6c          sbc  dif3   ;vorzeichenwechsel
2790:  1765 85 6a    l3    sta  dif1
2800:  1767 85 6e          sta  dif5   ;(x2-x1)-high
2810:  1769 68             pla
2820:  176a 85 69          sta  dif0
2830:  176c 85 6d          sta  dif4   ;(x2-x1)-low
2840:  176e 68             pla
2850:  176f 8d c7 17       sta  xkl
2860:  1772 8e c8 17       stx  xkh
2870:  1775 98             tya
2880:  1776 18             clc
2890:  1777 ed c9 17       sbc  yk     ;y2-y1
2900:  177a 90 04          bcc  l4     ;negativprint
2910:  177c 49 ff          eor  #$ff
2920:  177e 69 fe          adc  #$fe   ;vorzeichenwechsel
2930:  1780 85 6b    l4    sta  dif2   ;(y2-y1)
2940:  1782 8c c9 17       sty  yk
2950:  1785 66 6c          ror  dif3   ;(x2-x1)/2
2960:  1787 38             sec
2970:  1788 e5 69          sbc  dif0   ;(y2-y1)-(x2-x1)
2980:  178a aa             tax         ;low-byte nach x-reg.
                                       ;(punkte-zaehler-low)
2990:  178b a9 ff          lda  #$ff
3000:  178d e5 6a          sbc  dif1
3010:  178f 85 70          sta  za     ;high-byte nach za (punkte-
                                       ;zaehler-high)
3020:  1791 a4 6f          ldy  zwn
3030:  1793 b0 05          bcs  l5     ;unbedingt
3040:  1795 0a       l1    asl  a
3050:  1796 20 2a 17       jsr  r.l    ;rechts/links
3060:  1799 38             sec
3070:  179a a5 6d    l5    lda  dif4
3080:  179c 65 6b          adc  dif2
3090:  179e 85 6d          sta  dif4
```

```
3100:  17a0 a5 6e          lda  dif5
3110:  17a2 e9 00          sbc  #0      ;(x2-x1)-(y2-y1)nach(x2-x1)
3120:  17a4 85 6e    l2    sta  dif5
3130:  17a6 84 6f          sty  zwn
3140:  17a8 20 c9 16       jsr  plt     ;punkt zeichnen
3150:  17ab e8             inx
3160:  17ac d0 05          bne  l6
3170:  17ae e6 70          inc  za
3180:  17b0 d0 01          bne  l6      ;dec zaehler
3190:  17b2 60             rts
3200:  17b3 a5 6c    l6    lda  dif3
3210:  17b5 b0 de          bcs  l1
3220:  17b7 20 fb 16       jsr  u.o     ;unten/oben
3230:  17ba 18             clc
3240:  17bb a5 6d          lda  dif4
3250:  17bd 65 69          adc  dif0
3260:  17bf 85 6d          sta  dif4
3270:  17c1 a5 6e          lda  dif5
3280:  17c3 65 6a          adc  dif1
3290:  17c5 50 dd          bvc  l2      ;unbedingt
3300:                ;
3310:                ;
3320:                ;interne speicher
3330:  17c7
3340:                ;*****************
3350:                ;
3360:  17c7 00       xkl     .byt 0      ;xk-low-zwischenspeicher
3370:  17c8 00       xkh     .byt 0      ;xk-high-zwischenspeicher
3380:  17c9 00       yk      .byt 0      ;yk-zwischenspeicher
3390:  17ca 00       zwis    .byt 0      ;zwischenspeicher
3400:  17cb 00       flg2    .byt 0      ;zwischenspeicher
3410:                ;
3420:                ;
3430:                ;tabellen
3430:  17cc
3440:                ;*********
3450:                ;
3460:                ;
3470:                ;multiplikationstabelle
3470:  17cc
```

```
3480:                ;(n*320 fuer n=0 bis n=24)
3490:                ;
3500:                ;high-bytes
3510:                ;
3520:  17cc 00 01 02 mul.h    .byt 0,1,2,3,5,6,7,8,10,11,12,13,15,16
3530:  17da 11 12 14          .byt 17,18,20,21,22,23,25,26,27,28,30,31
3540:                ;
3550:                ;low-bytes
3560:                ;
3570:  17e6 00 40 80 mul.l    .byt $00,$40,$80,$c0
3580:                ;
3590:                ;
3600:                ;maskentabelle
3600:  17ea
3610:                ;
3620:  17ea 01 02 04 msktab .byt %00000001,%00000010,%00000100,%00001000
3630:  17ee 10 20 40          .byt %00010000,%00100000,%01000000,%10000000
3640:                ;
}1630-17f2
no errors
```

Und hier der BASIC-Lader:

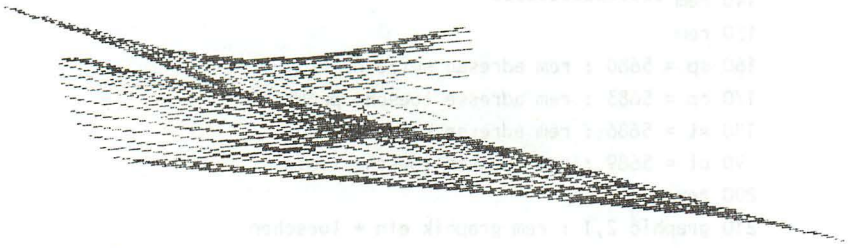
```
100 for i = 5680 to 6130
110 read x : poke i,x : s=s+x : next
120 data  76, 63, 22, 76, 60, 22, 76, 82, 22, 76, 79, 22
130 data 162,  0, 44,162,128,134,151, 32,145,148, 32, 96
140 data  22, 32,123, 22, 76,201, 22,162,  0, 44,162,128
150 data  32, 65, 22, 32,145,148, 32, 96, 22, 76, 63, 23
160 data  32,210,157,138,168,166, 21,192,200,176, 13,165
170 data  20,224,  1,144,  6,208,  5,201, 64,176,  1, 96
180 data  76,126,157,140,201, 23,141,199, 23,142,200, 23
190 data 133, 20,134, 21,152, 74, 74, 74,170,189,204, 23
200 data 133,173,138, 41,  3,170,189,230, 23,133,172,152
210 data  41,  7, 24,101,172,133,172,165, 20, 41,248,133
220 data  99,169, 32,  5,173,133,173, 24,165,172,101, 99
230 data 133,172,165,173,101, 21,133,173,165, 20, 41,  7
240 data  73,  7,170,189,234, 23,133,171, 96,160,  0,  8
250 data 165,171, 36,151, 48,  5, 73,255, 49,172, 44, 17
```

```
260 data 172,145,172, 40,164,111, 96,165,172, 41,  7,201
270 data   7,240,  5, 56,169,  0,176,  4,169, 56,230,173
280 data 101,172,133,172,169,  0,101,173,133,173, 96, 48
290 data 226,165,172, 41,  7,240,  5, 24,169,255,144,  4
300 data 169,199,198,173,101,172,133,172,165,173,233,  0
310 data 133,173, 96, 70,171,144, 14,102,171,165,172,200
320 data  24,105,  8,133,172,144,  2,230,173, 96, 16,235
330 data   6,171,144, 14, 38,171,165,172,136, 56,233,  8
340 data 133,172,176,  2,198,173, 96, 72,173,200, 23, 74
350 data 173,199, 23,106, 74, 74,133,111,104, 72, 56,237
360 data 199, 23, 72,138,237,200, 23,133,108,176, 10,104
370 data  73,255,105,  1, 72,169,  0,229,108,133,106,133
380 data 110,104,133,105,133,109,104,141,199, 23,142,200
390 data  23,152, 24,237,201, 23,144,  4, 73,255,105,254
400 data 133,107,140,201, 23,102,108, 56,229,105,170,169
410 data 255,229,106,133,112,164,111,176,  5, 10, 32, 42
420 data  23, 56,165,109,101,107,133,109,165,110,233,  0
430 data 133,110,132,111, 32,201, 22,232,208,  5,230,112
440 data 208,  1, 96,165,108,176,222, 32,251, 22, 24,165
450 data 109,101,105,133,109,165,110,101,106, 80,221,  0
460 data   0,  0,  0,  0,  0,  1,  2,  3,  5,  6,  7,  8
470 data  10, 11, 12, 13, 15, 16, 17, 18, 20, 21, 22, 23
480 data  25, 26, 27, 28, 30, 31,  0, 64,128,192,  1,  2
490 data   4,  8, 16, 32, 64,128,  0
500 if s <> 47183 then print "fehler in datas !!" : end
510 print ok
520 rem
530 rem basic-ende unter programmstart setzen:
540 hi = 5679                  : rem programm-startadresse-1
550 poke 55,hi-int(hi/256)*256 : rem low-byte end-adresse
560 poke 56,int(hi/256)        : rem high-byte
570 clr                        : rem initialisieren
```

Das versprochene kleine Demo:

```
100 rem ***************
110 rem **             **
120 rem ** line-demo **
130 rem **             **
```

```
140 rem ***************
150 rem
160 sp = 5680 : rem adresse setze punkt
170 cp = 5683 : rem adresse loesche punktstep
180 sl = 5686 : rem adresse setze linie
190 cl = 5689 : rem adresse loesche linie
200 rem
210 graphic 2,1 : rem graphik ein + loeschen
220 rem
230 modus = sl : rem modus: zeichne linie
240 gosub 340
250 modus = cl : rem modus: loesche linie
260 gosub 340
270 modus = sp : rem modus: setze punkt
280 gosub 410
290 modus = cp : rem modus: loesche punkt
300 gosub 410
310 goto 230
320 rem
330 rem
340 rem zeichenroutine 1:
350 for x=0 to 319 step 5
360 sys modus,80*sin(x/50)+100,30*sin(x/20)+60,x,x/2
370 next x
380 return
390 rem
400 rem
410 rem zeichenroutine 2:
420 for x=0 to 319
430 sys modus,x,60*sin(x/20)+100
440 next x
450 return
```



Wie Sie sehen, bietet Ihnen diese kleine BASIC-Erweiterung insgesamt 4 verschiedene Befehle mit folgender Syntax:

sys sp,x,y	- setze Punkt an der Stelle x,y
sys cp,x,y	- lösche Punkt an der Stelle x,y
sys sl,x1,y1,x2,y2	- zeichne Linie von (x1,y1) nach (x2,y2)
sys cl,x1,y1,x2,y2	- lösche Linie von (x1,y1) nach (x2,y2)

wobei für die Variablen sp, cp, sl und cl folgende Adressen einzusetzen sind:

```
sp = 5680
cp = 5683
sl = 5686
cl = 5689
```

Anhand dieses Assemblerprogrammes wird Ihnen demonstriert, wie einfach eine BASIC-Erweiterung auf Ihrem Computer realisiert werden kann, wie Werte vom BASIC-Programm an ein Maschinenspracheprogramm übergeben werden können und na-

türlich nicht zuletzt, wie eine schnelle Line- oder Punkt-Routine in Assembler verwirklicht werden kann.

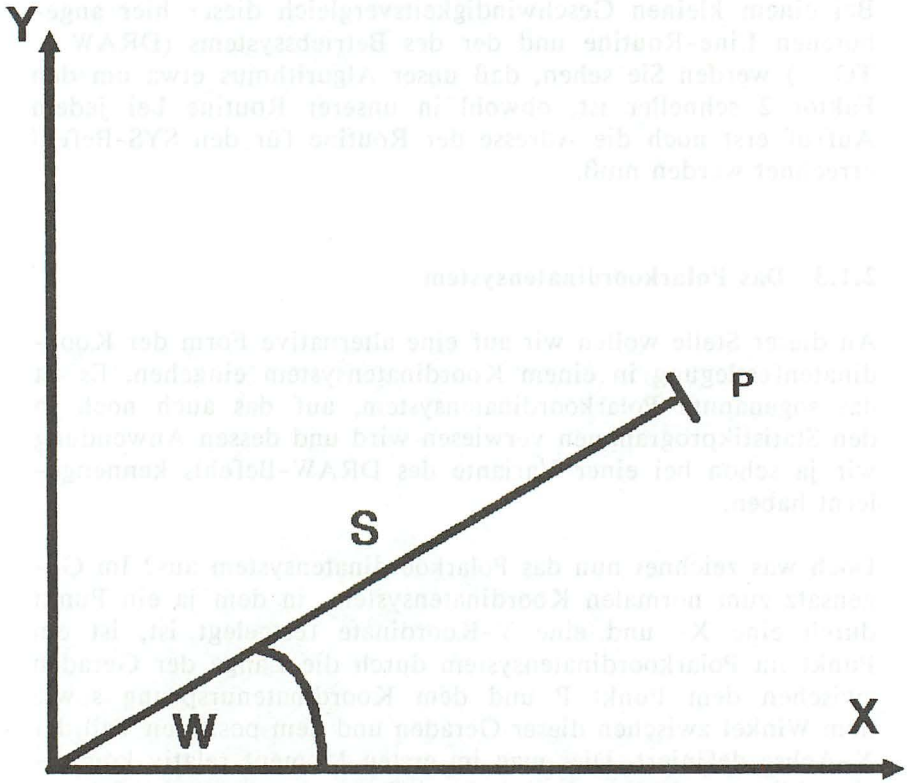
Bei einem kleinen Geschwindigkeitsvergleich dieser hier angebotenen Line-Routine und der des Betriebssystems (DRAW ... TO ...) werden Sie sehen, daß unser Algorithmus etwa um den Faktor 2 schneller ist, obwohl in unserer Routine bei jedem Aufruf erst noch die Adresse der Routine für den SYS-Befehl errechnet werden muß.

2.1.3 Das Polarkoordinatensystem

An dieser Stelle wollen wir auf eine alternative Form der Koordinatenfestlegung in einem Koordinatensystem eingehen. Es ist das sogenannte Polarkoordinatensystem, auf das auch noch in den Statistikprogrammen verwiesen wird und dessen Anwendung wir ja schon bei einer Variante des DRAW-Befehls kennengelernt haben.

Doch was zeichnet nun das Polarkoordinatensystem aus? Im Gegensatz zum normalen Koordinatensystem, in dem ja ein Punkt durch eine X- und eine Y-Koordinate festgelegt ist, ist ein Punkt im Polarkoordinatensystem durch die Länge der Geraden zwischen dem Punkt P und dem Koordinatenursprung sowie dem Winkel zwischen dieser Geraden und dem positiven Teil der X-Achse definiert. Dies mag im ersten Moment relativ kompliziert klingen, doch es ist relativ einfach, es anhand der folgenden Zeichnung zu verdeutlichen:

Das Polarkoordinatensystem



In einem Polarkoordinatensystem wird ein Punkt durch die Länge der Strecke S und durch den Winkel W bestimmt.

Doch was haben wir nun von dieser, im Gegensatz zum normalen Koordinatensystem relativ komplizierten Bestimmung eines Punktes? Nun, es sprechen mehrere Grunde dafür:

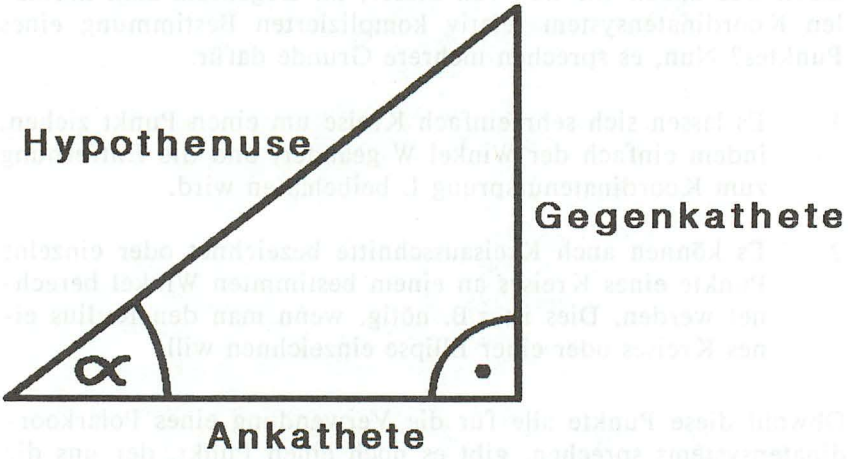
1. Es lassen sich sehr einfach Kreise um einen Punkt ziehen, indem einfach der Winkel W geändert und die Entfernung zum Koordinatenursprung L beibehalten wird.

2. Es können auch Kreisausschnitte bezeichnet oder einzelne Punkte eines Kreises an einem bestimmten Winkel berechnet werden. Dies ist z.B. nötig, wenn man den Radius eines Kreises oder einer Ellipse einzeichnen will.

Obwohl diese Punkte alle für die Verwendung eines Polarkoordinatensystems sprechen, gibt es noch einen Punkt, der uns die Anwendung des Polarkoordinatensystems etwas erschwert. Es ist die Umrechnung dieser Koordinaten in die normalen X- und Y-Angaben, denn der Computer versteht ja nur diese Koordinaten. Um die Koordinaten umzurechnen, müssen wir uns etwas mehr mit den Winkelsätzen im rechtwinkligen Dreieck beschäftigen. Für den Fall, daß Sie sich in diesem Fachbereich der Mathematik nicht so gut auskennen sollten, haben wir an dieser Stelle noch einmal die Winkelsätze im rechtwinklingen Dreieck beschrieben:

Die Definition des Sinus

Die Definition des Sinus besagt, daß in einem rechtwinkligen Dreieck der Sinus des Winkels Alpha gleich dem Quotient von Gegenkathete und Hypotenuse ist. Anhand der folgenden Zeichnung, können Sie sich diese Begriffe vergegenwärtigen:



Folgende Gesetze gelten in einem rechtwinktigen Dreieck:

$$\sin \text{alpha} = \frac{\text{Gegenkathete}}{\text{Hypothenuse}}$$

$$\cos \text{alpha} = \frac{\text{Ankathete}}{\text{Hypothenuse}}$$

In diesem Fall wäre also sin (wa) = $\frac{a}{c}$

Die Definition des Kosinus

Diese zweite Winkelfunktion erklärt nun ein weiteres Verhältnis in einem rechtwinkligen Dreieck. Sie besagt nämlich, daß der

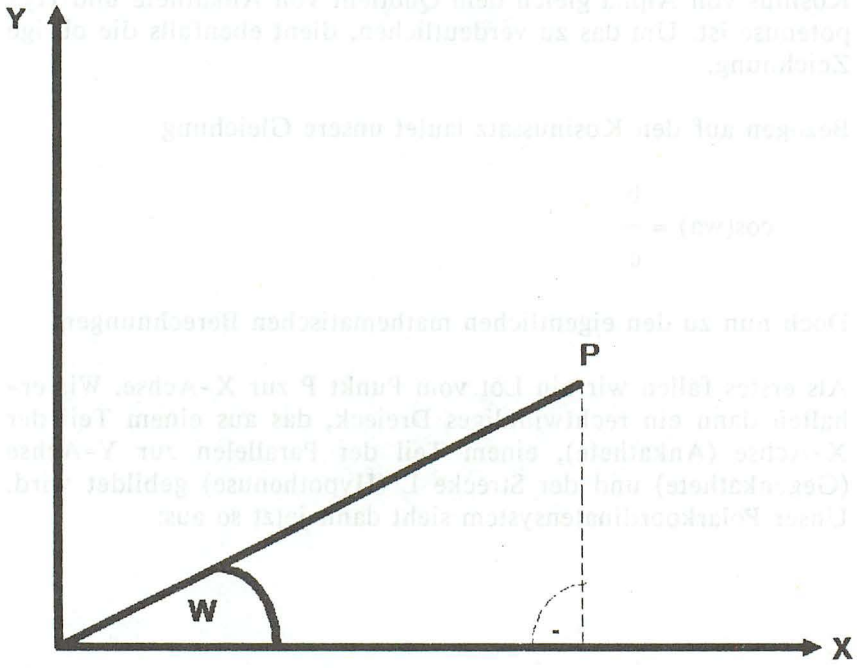
Kosinus von Alpha gleich dem Quotient von Ankathete und Hypotenuse ist. Um das zu verdeutlichen, dient ebenfalls die obrige Zeichnung.

Bezogen auf den Kosinussatz lautet unsere Gleichung

$$\cos(wa) = \frac{b}{c}$$

Doch nun zu den eigentlichen mathematischen Berechnungen:

Als erstes fällen wir ein Lot vom Punkt P zur X-Achse. Wir erhalten dann ein rechtwinkliges Dreieck, das aus einem Teil der X-Achse (Ankathete), einem Teil der Parallelen zur Y-Achse (Gegenkathete) und der Strecke L (Hypothenuse) gebildet wird. Unser Polarkoordinatensystem sieht dann jetzt so aus:



Bekannt sind uns die Strecke L und der Winkel W. Mit diesen beiden Angaben können wir nun die Kathete und die Gegenkathete des Dreiecks berechnen: Für die Berechnung der Gegenkathete, also des Y-Wertes, gilt:

Y = L * SIN (W)

während für die Berechnung des X-Wertes folgende Formel gilt:

X = L * COS (W)

Wollen Sie nun eine Ellipse zeichnen, so können Sie in beiden Formeln auch verschiedene Werte für L einsetzen. Das könnte dann beispielsweise so aussehen:

Y = YR * SIN (W) und X = XR * COS (W)

wobei außer den genannten Parametern bedeuten:

XR: Radius der Ellipse in X-Richtung
YR: Radius der Ellipse in Y-Richtung

Nun kommt es aber selten vor, daß wir einen Kreis oder eine Ellipse direkt um den Koordinatenursprung zeichnen wollen. Aus diesem Grund verschieben wir unseren Koordinatenursprung zum gewünschten Kreismittelpunkt. Die endgültigen Formeln sehen dann so aus:

X = MX + XR * COS (W) und Y = MY + YR * SIN (W)

wobei MX und MY jetzt die Koordinaten des Kreismittelpunktes darstellen. Da der Winkel W in Radiant angegeben werden muß, wir aber in Grad rechnen, hier noch die Formeln zur Umrechnung von Grad in Radiant und umgekehrt:

Radiant = Grad * (3.1415 / 180)
Grad = Radiant / (3.1415 / 180)

Probieren Sie doch nun einmal das oben erlangte Wissen selbständig in einem kleinen Programm aus! Setzen Sie z.B. den X-Radius auf 50, den Y-Radius auf 40 und erhöhen Sie in einer Schleife den Winkel W. Nachdem Sie dann die Koordinaten umgerechnet haben, können Sie mittels DRAW den Punkt setzen.

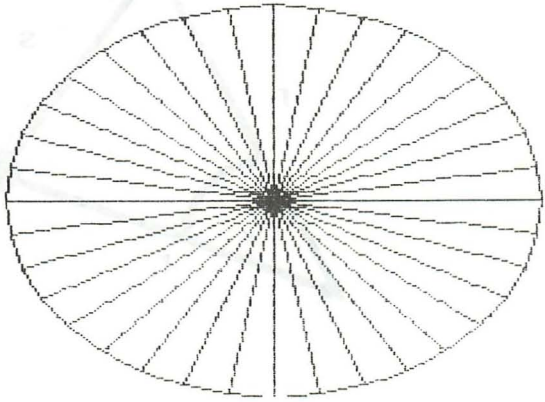
2.1.4 Kreis und Ellipsenberechnung im Polarkoordinatensystem

Auch das Problem des Ellipsen und Kreisberechnung ist jetzt, da wir ja das Verfahren im Polarkoordinatensystem schon kennen, recht einfach zu lösen. Wir errechnen unseren Punkt, indem wir jeweils 360mal unseren Winkel erhöhen, und jedesmal nachdem dieses geschehen ist, eine "unsichtbare Linie" der Länge R von unserem Ursprungspunkt ziehen. Das entstehende Ende der Linie merken wir uns nun, und verbinden anschließend das Ende der nächsten Linie mit unserem vorherigen Ende. Ist das 360mal geschehen, so erhalten wir einen vollständigen Kreis.

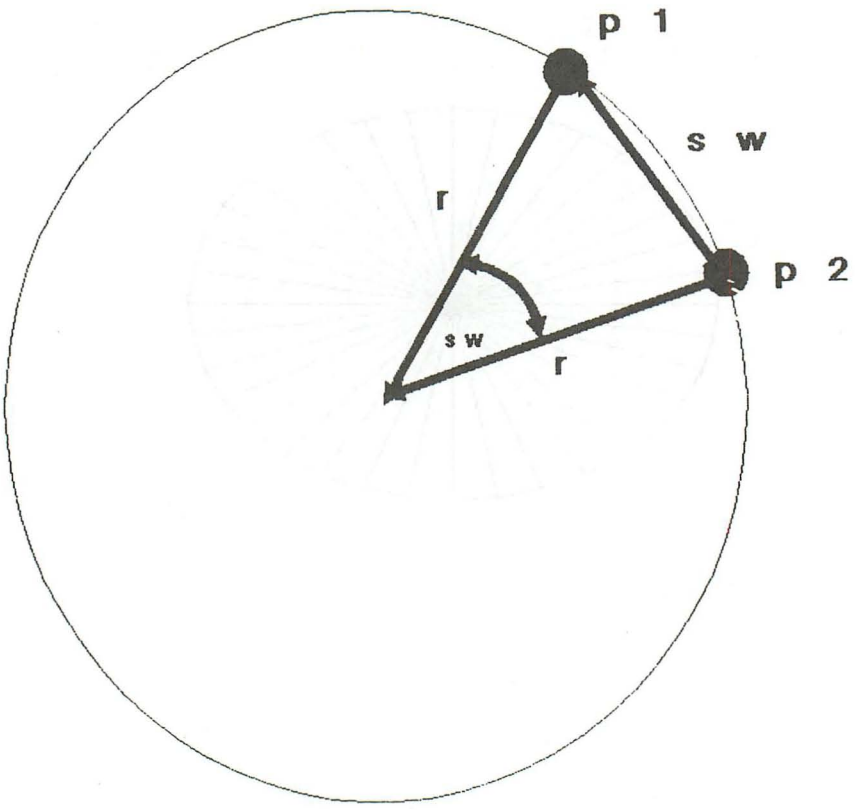
In dem folgenden Demoprogramm ist dieses Verfahren unter Ausnutzung des Polarkoordinatensystems des Draw-Befehls durchgeführt worden. Um die Anwendung noch einmal zu verdeutlichen, habe ich jedoch die einzelnen Radien in dieses Programm einzeichnen lassen.

Unter dem Programm finden Sie dann zwei Verdeutlichungsgrafiken, bei denen die erste daraus entstanden ist, das die "0" beim Draw-Befehl durch eine "1" ersetzt wurde. Damit wird die sonst "blind" gezeichnete Linie sichtbar. Die sich daran anschließende Schaugrafik zeigt dann, wie genau jeder einzelne Punkt dieses Kreises berechnet wird.

```
100 rem ################################################
110 rem ###                                          ###
120 rem ###        zeichnen eines kreises            ###
130 rem ###        ------------------------          ###
140 rem ###                                          ###
150 rem ################################################
160 :
170 for w=0 to 360
180 draw ,160,100 to 100;w
190 x=rdot(0)
200 y=rdot(1)
210 if fl=0 then fl=1:xa=x:ya=y
220 draw ,xa,ya to x,y
230 next x
```



Prinzip der Kreisberechnung im Polarkoordinatensystem



Wie Sie hier sehr gut sehen können, wird der Kreis eigentlich aus einer Vielzahl von Linien gebildet. Es werden nämlich die Eckpunkte eines in einem bestimmten Abstand vom Ausgangspunkt stehenden Radius jeweils mit einer Linie verbunden, so wie das in unserem Beispiel mit den Punkten p1 und p2 geschieht.

Weitaus schwieriger ist schon das Zeichnen von Ellipsen, zumal wir hier nicht den Draw-Befehl zu Hilfe nehmen können, da der uns ja nur einen Radius erlaubt. Also müssen wir nun das oben angesprochene Polarkoordinatensystem sozusagen "zu Fuß" programmieren. Aus unserer Umrechnung vom Polarkoordinatensystem in das normale, kartesische Koordinatensystem ergaben sich wie sich sich erinnern, folgende Formeln:

```
x = l*cos(w)
y = l*sin(w)
```

wobei wir nun die Längenvariable, die die Ordinate (die Y-Koordinate) ergibt, durch den Y-Radius und die Abszisse (die X-Koordinate) durch den X-Radius ersetzen.

Jetzt müssen wir nur noch den Mittelpunkt unseres Kreises oder unserer Ellipse errechnen. Das erreichen wir durch Addieren der Koordinaten des Mittelpunktes zu den entsprechenden Koordinaten des Kreispunktes. Wir erhalten also folgende Formeln:

```
x = xr*cos(w)+xmitte
y = yr*sin(w)+ymitte
```

Nun brauchen wir nur noch in einer bestimmten Schleife die einzelnen Winkelabschnitte durchzugehen und unsere Endpunkte, die wir ja mit diesen Formeln errechnen, mit einer Linie, ähnlich wie wir das ja schon beim obengenannten Verfahren durchgeführt haben, zu verbinden. Indem wir nun noch die Schrittweite der Schleife variieren, haben wir noch zu guter letzt die Möglichkeit, verschiedenartige Polygone zu zeichnen, deren Eckenzahl wir, ähnlich der Formel, die wir beim CIRCLE-Befehl kennengelernt haben, mit folgender Formel errechnen können:

```
anzahl der ecken = 360/schrittweite
```

Damit stehen uns alle benötigten Mittel zur Verfügung, um nun unser Wollen in die Tat umzusetzen, und die Ellipse zu zeichnen:

```
100 rem ################################################
110 rem ###                                            ###
120 rem ###      zeichnen einer ellipse                ###
130 rem ###      ------------------------              ###
140 rem ###                                            ###
150 rem ################################################
160 :
170 xr=160:yr=100              : rem die radien
180 xm=160:ym=100              : rem der mittelpunkt
190 sw=4      * (pi/180)       : rem die schrittweite
200 :
210 pi=3.1415926               : rem pi
220 eb=0      * (pi/180)       : rem ellipsenanfang
230 ee=360    * (pi/180)       : rem ellipsenende
240 :
250 rem ### ellipse zeichnen ###
260 :
270 for z=eb to ee step sw
280 x=xr*cos(w)+xm
290 y=yr*sin(w)+ym
300 if fl=0 then fl=1:xa=x:ya=y
310 draw ,x,y to xa,ya
320 xa=x:ya=y
330 next
```

2.1.5 Die Achtelkreismethode

An dieser Stelle möchte ich noch einen Nachtrag zu den Zeilen 190, 220 und 230 anführen. Wie Sie ja wissen, rechnet Ihr Computer nicht, wie wir dies wohl gewohnt sind, in Altgrad, dessen Einteilung von 0 bis 360 reicht, sondern in Radiant (0 bis 2*pi). In diesen Zeilen werden nun die Angaben, die in Altgrad angegeben sind, nach Radiant umgerechnet.

Es gibt nun noch eine zweite, sehr schnelle Methode, einen Kreis zu zeichnen, die sogenannte Achtelkreismethode. Hierbei werden nach jeder Berechnung insgesamt 8 verschiedene Punkte des Kreises gezeichnet. So reduziert sich die Zeit zum Zeichnen des Kreises auf etwa ein achtel. Ausgehend von den 4 Vierteln

(0, 90, 180 und 270 Grad) wird der Kreis so jeweils in und gegen den Uhrzeigersinn gezeichnet. Dabei geht man folgendermaßen vor:

Angenommen, man hat die Koordinaten eines Kreispunktes bei 1 Grad nach dem alten Algorithmus (s.o.) berechnet und den entsprechenden Punkt des Kreises gezeichnet. Dieser befindet sich nun im rechten oberen Viertel des Kreises. Da ein Kreis nun ein sehr symmetrisches Objekt ist, kann dieser Punkt nun an dem senkrechten Durchmesser des Kreises gespiegelt werden. Die y-Koordinate dieses dadurch entstehenden neuen Punktes (im linken oberen Viertel) ist identisch mit der des Ursprungpunktes. Lediglich die x-Koordinate ist nach folgender Formel umzurechnen:

```
x2 = xmitte-(x1-xmitte)
x2 = 2*xmitte-x1
```

Dabei ist xmitte die x-Koordinate des Kreismittelpunktes. Da man zu den Koordinaten ja sowieso erst zum Schluß kurz vor dem Zeichnen die Mittelpunktkoordinaten hinzuaddiert (s.o.), braucht man in der obigen Gleichung die Rechnerei mit xmitte gar nicht, sondern rechnet einfach:

```
x2U = -x1U
```

Dabei sind x2U und x1U die Koordinaten des Kreises mit dem Mittelpunkt im Ursprung.

Damit kommt man sehr "billig" an einen zweiten Kreispunkt. Doch nicht genug. Spiegelt man diese beiden Punkte um den waagerechten Durchmesser, so erhält man wieder zwei neue Punkte. Diese beiden Punkte errechnet man folgendermaßen:

```
x3U =     x1U und x4U =     x2U = -x1U
y3U = -   y1U und y4U = -   y2U = -y1U
```

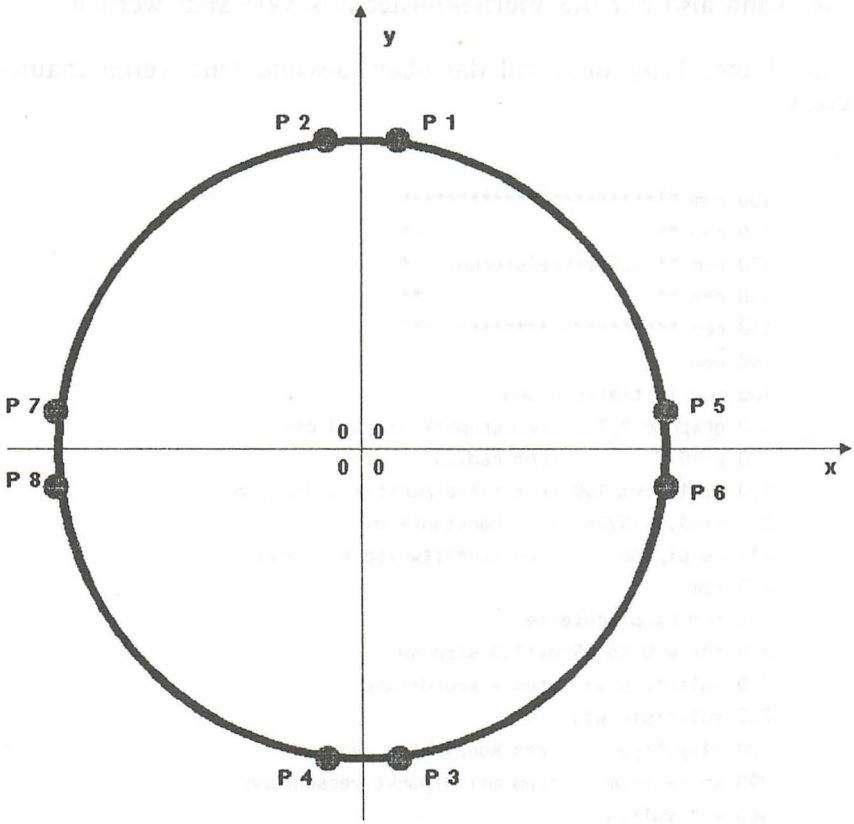
Auch hier sind (x3U,y3U) und (x4U,y4U) die Koordinaten der beiden in der unteren Hälfte des Kreises (um den Nullpunkt) liegenden Punkte.

Durch eine Drehung dieser nunmehr 4 Punkte um den Nullpunkt erhält man 4 weitere Punkte, die jeweils rechts und links vom Kreismittelpunkt liegen. Diese Punkte p5,p6,p7,p8 erhält man durch folgende Rechnungen:

```
x5U = y1U          und   y5U =  x1U
x6U = y1U          und   y6U = -x1U
x7U = y3U = -y1U   und   y7U =  x1U
x8U = y3U = -y1U   und   y8U = -x1U
```

Die untenstehende Zeichnung mag diesen Sachverhalt noch erläutern. Man kann also durch einfache Vorzeichenwechsel 8 verschiedene Koordinaten des Kreises berechnen. Nach diesen Spiegelungen und Drehungen addieren wir jeweils noch die Mittelpunktskoordinaten hinzu und schon liegen wir wieder richtig.

Achtelkreistechnik



Der Nachteil dieser Kreisberechnung: Es werden stets nur volle Kreise gezeichnet. Einfache Bögen sind dagegen nur mit viel Aufwand zu realisieren. Weiterhin fällt die Möglichkeit flach, die Routine auch für Polygone zu verwenden. Bei Ellipsen ist

der letzte Schritt (die Drehung) nicht ohne weiteres möglich, hier kann also nur die Viertelkreistechnik verwandt werden.

Ein kleines Programm soll das oben Gesagte kurz veranschaulichen:

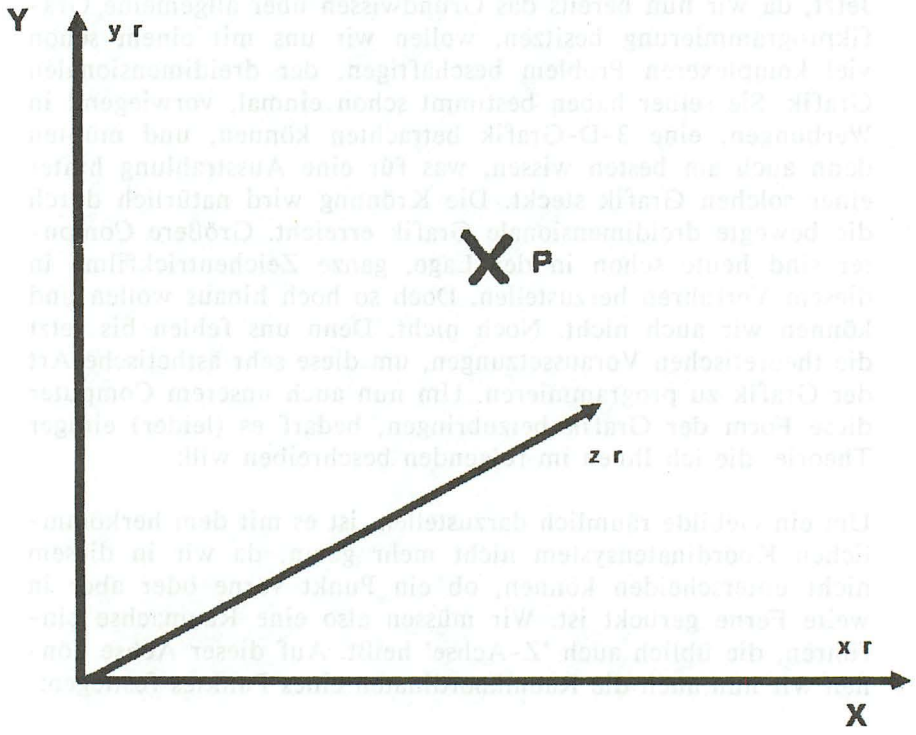
```
100 rem ************************
110 rem **                      **
120 rem ** achtelkreistechnik **
130 rem **                      **
140 rem ************************
150 rem
160 rem initialisierung
170 graphic 2,1    :rem graphik ein und clr
180 r=50           :rem radius
190 xm=160:ym=100 :rem mittelpunktskoordinaten
200 pi=3.1415926   :rem konstante pi
210 sw=pi/180      :rem schrittweite = 1 grad
220 rem
230 rem hauptschleife
240 for w=0 to 45*pi/180 step sw
250 xu1=r*cos(w)   :rem x-koordinate
260 yu1=r*sin(w)g
280 y1=yu1+ym      :rem koorinaten gemaess
290 xn=-xu1+xm     :rem mittelpunkt verschoben
300 yn=-yu1+ym
310 xo=-xu1+ym
320 yo=-yu1+xm
330 xs=xu1+ym
340 ys=yu1+xm
350 draw,x1,y1     :rem 8 punkte zeichnen
360 draw ,ys,xs
370 draw ,ys,xo
380 draw ,x1,yn
390 draw ,xn,yn
400 draw ,yo,xo
410 draw ,yo,xs
420 draw ,xn,y1
430 next
```

2.1.6 Die Paralellprojektion

Jetzt, da wir nun bereits das Grundwissen über allgemeine Grafikprogrammierung besitzen, wollen wir uns mit einem schon viel komplexeren Problem beschäftigen, der dreidimensionalen Grafik. Sie selber haben bestimmt schon einmal, vorwiegend in Werbungen, eine 3-D-Grafik betrachten können, und müßten dann auch am besten wissen, was für eine Ausstrahlung hinter einer solchen Grafik steckt. Die Krönung wird natürlich durch die bewegte dreidimensionale Grafik erreicht. Größere Computer sind heute schon in der Lage, ganze Zeichentrickfilme in diesem Verfahren herzustellen. Doch so hoch hinaus wollen und können wir auch nicht. Noch nicht. Denn uns fehlen bis jetzt die theoretischen Voraussetzungen, um diese sehr ästhetische Art der Grafik zu programmieren. Um nun auch unserem Computer diese Form der Grafik beizubringen, bedarf es (leider) einiger Theorie, die ich Ihnen im folgenden beschreiben will:

Um ein Gebilde räumlich darzustellen, ist es mit dem herkömmlichen Koordinatensystem nicht mehr getan, da wir in diesem nicht unterscheiden können, ob ein Punkt vorne oder aber in weite Ferne gerückt ist. Wir müssen also eine Raumachse einführen, die üblich auch 'Z-Achse' heißt. Auf dieser Achse können wir nun auch die Raumkoordinaten eines Punktes festlegen:

Das 3-D-Koordinatensystem

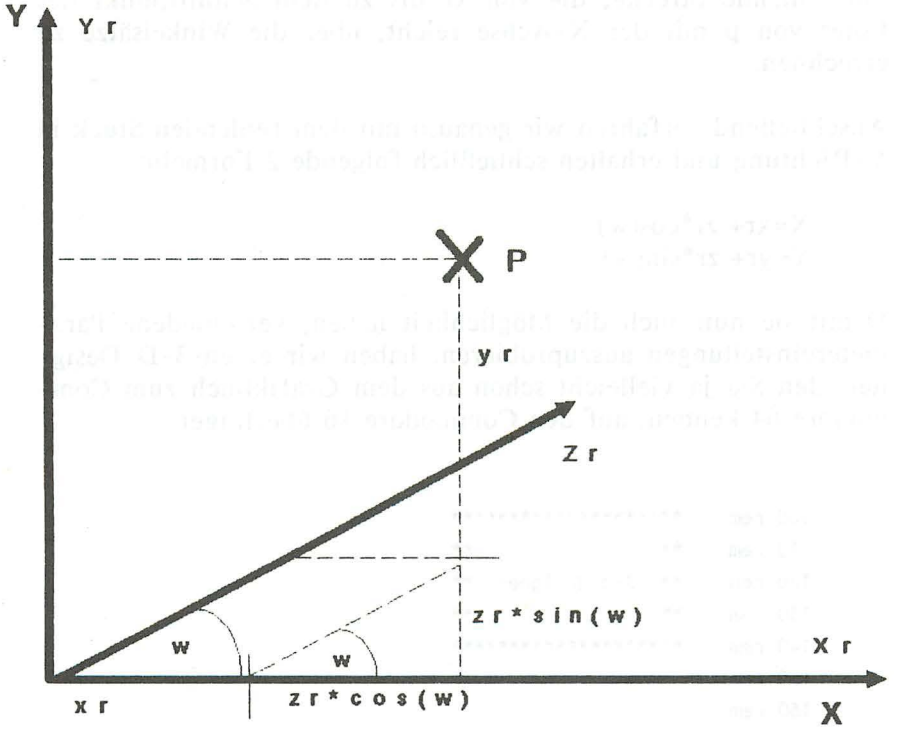


Wie zeichnet man nun einen Punkt in ein solches Koordinatensystem ein? - Nun, zunächst einmal verfahren wir genau so, wie wir es von dem zweidimensionalen Koordinatensystem gewohnt sind. Danach zeichnen wir eine Parallele zur Z-Achse, die durch den entstandenen Punkt, nennen wir ihn 'p', verlaufen muß. Auf dieser Parallelen tragen wir nun den Abstand der Z-Koordinate ab und erhalten unseren neuen dreidimensionalen Punkt.

Das Problem, das sich ergibt, ist folgendes. Wir müssen einen Rechnenalgorithmus finden, der es uns ermöglicht, einen Punkt mit 3-D-Koordinaten in einen gleichen mit 2-D-Koordinaten umzurechnen.

Eine Zeichnung soll uns dabei helfen:

Das 3-D-Koordinatensystem



Wie aus der Zeichnung ersichtlich, ergeben sich folgende Formeln zur Umrechnung eines Punktes im 3-D-Koordinatensystem in das zweidimensionale:

$$x = xr + zr * \cos(w)$$
$$y = yr + zr * \sin(w)$$

Die Vorgehensweise ist nun folgende: Zunächst müssen wir das Lot vom Punkt P fällen. Nun bildet sich ein rechtwinkliges Dreieck mit den Punkten xr,p und dem Schnittpunkt des Lotes von P mit der X-Achse. Hier haben wir nun die Möglichkeit, die fehlende Strecke, die von xr bis zu dem Schnittpunkt des Lotes von p mit der X-Achse reicht, über die Winkelsätze zu errechnen.

Anschließend verfahren wir genauso mit dem fehlenden Stück in Y-Richtung und erhalten schließlich folgende 2 Formeln:

$$X=xr+ zr*\cos(w)$$
$$Y=yr+ zr*\sin(w)$$

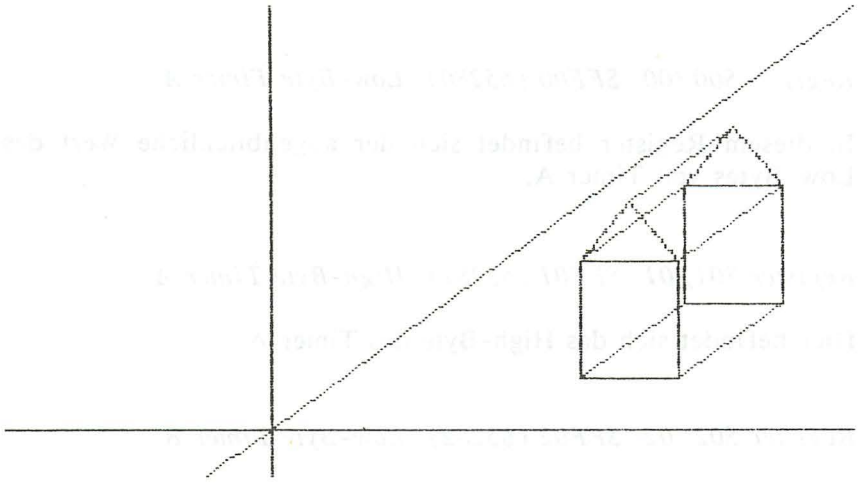
Damit sie nun auch die Möglichkeit haben, verschiedene Parametereinstellungen auszuprobieren, haben wir einen 3-D-Designer, den Sie ja vielleicht schon aus dem Grafikbuch zum Commodore 64 kennen, auf den Commodore 16 übertragen:

```
100 rem    ********************
110 rem    **                  **
120 rem    **  3-d-designer    **
130 rem    **                  **
140 rem    ********************
150 rem
160 rem
270 rem
280 rem          *********
290 rem    ****  parameter   ***
300 rem          *********
310 rem
320 f1 = 5 : f2 = 5 : f3 = 3 : rem zerrung
```

```
330 ar = 15 : br =  0 : cr = 10 : rem verschiebung der raumkoord.
340 w = 3.1415926/4: rem sichtwinkel (in rad)
350 si = sin(w) : co = cos(w) : rem konstanten
360 v1 = 100 : v2 = 180 : rem verschiebung der ebenenkoord.
370 rem
380 rem          **********
390 rem   ****  umrechnung  ***
400 rem          **********
410 rem
420 color 0,1    : rem hintergrundfarbe
430 color 1,6    : rem zeichenfarbe
440 color 4,1    : rem rahmenfarbe
450 graphic 1,1 : rem graphik init
460 fg=0:if v1<0 or v1>319 then fg=1:goto 480 : rem flag
470 draw ,v1,0 to v1,199 : rem raum-z-achse
480 if v2<0 or v2>200 then fg=1:goto 500
490 draw ,0,v2 to 319,v2 : rem raum-x-achse
500 z2 = v1 - (199-v2)/si*co : z1 = v1 + v2/si*co
510 if fg=0 and z1>=0 and z1<320 then draw ,v1,v2 to z1,0: rem rau
m-y-achse oben
520 if fg=0 and z2>=0 and z2<320 then draw ,v1,v2 to z2,199: rem r
aum-y-achse unten
530 read ap : dim x%(ap),y%(ap) : rem anzahl punkte
540 for za=1 to ap : rem punktezahl
550 read xr,zr,yr
560 x%(za) =  f1*(xr+ar) + f3*(yr+cr)*co+v1
570 y%(za) = -f2*(zr+br) - f3*(yr+cr)*si+v2
580 next za : rem naechster punkt
590 rem
600 rem          *******
610 rem    ****  striche  ***
620 rem          *******
630 rem
640 read al : rem anzahl linien
650 for za=1 to al
660 read p1,p2 : rem punktnummern lesen
670 if x%(p1)<0 or y%(p1)<0 or x%(p2)<0 or y%(p2)<0 then goto
700:rem ausserhalb
680 ifx%(p1)>319ory%(p1)>199orx%(p2)>319ory%(p2)>199then goto 700:
rem ausserhalb
```

```
690 draw ,x%(p1),y%(p1) to x%(p2),y%(p2) : rem verbinden
700 next za : rem naechste linie
710 getkey a$
720 graphic 0,1 : list : rem graphik aus
730 rem
740 rem        ***********
750 rem ****  koordinaten  ****
760 rem        ***********
770 data 10 : rem anzahl punkte
780 data  0, 0, 0,  6, 0, 0,  6,10, 0
790 data  0,10, 0,  3,15, 0,  3,15,15
800 data  6,10,15,  6, 0,15,  0, 0,15
810 data  0,10,15
820 rem
830 rem        ************
840 rem ****  verbindungen  ****
850 rem        ************
860 data 17 : rem anzahl linien
870 data  1, 2,  2, 3,  3, 4,  4, 1
880 data  4, 5,  5, 3,  5, 6,  6, 7
890 data  7, 3,  7, 8,  8, 2,  8, 9
900 data  9, 1,  9,10, 10, 4, 10, 6
910 data 10, 7
```

Das Ergebnis



2.2 Hardwaregrundlagen

Bekommen Sie keinen Schreck, wenn Sie diese Überschrift lesen! Ich will in diesem Abschnitt keineswegs Ihren oder meinen Computer "auseinandernehmen". Mir geht es in diesem Kapitel nur darum, verständlich zu machen, wie in groben Abläufen das Darstellen von Zeichen oder Grafik auf dem Grafik- und Textbildschirm überhaupt funktioniert. Erst durch diese Grundlagen, die ja damit geschaffen werden, ist ein wirklich sinnvolles Arbeiten (natürlich in Bezug auf Grafik) erst möglich.

2.2.1 Die Register des TED

TED, so nennt sich der universale Controller des C16. Universal deshalb, weil er nicht nur für die Grafik, sondern auch für den Sound und andere I/O-Funktionen zuständig ist. Dieser TED,

dessen 32 Register ab der Adresse $FF00 liegen, werden nun im folgenden beschrieben, weil sie die Grundlagen für viele, oder mehr noch, sogar für aller Grafikanwendungnen sind. Die Funktionen dieser Register sind nun im folgenden beschrieben:

Register $00/00 $FF00 (65280) Low-Byte Timer A

In diesem Register befindet sich der augenblickliche Wert des Low-Bytes von Timer A.

Register $01/01 $FF01 (65281) High-Byte Timer A

Hier befindet sich das High-Byte des Timer A.

Register $02/02 $FF02 (65282) Low-Byte Timer B

Dieses Register enthält das Low-Byte des Timers B entsprechend zu Register 0.

Register $03/03 $FF03 (65283) High-Byte Timer B

In diesem Register befindet sich, analog zu Register 1, das High-Byte des Timers B.

Register $04/04 $FF04 (65284) Low-Byte Timer C

Dieses Timer-Register enthält das Low-Byte des Timers C.

Register $05/05 $FF05 (65285) High-Byte Timer C

Dieses nun letzte Timer-Register enthält den High-Byte des Timers C.

Register $06/06 $FF06 (65286) Multi-Function-Register I

Bits 0-2: Diese Bits bewirken eine vertikale Verschiebung des Bildschirms um jeweils ein Pixel.

Bit 3: Um die entstehenden Lücken beim vertikalen Smooth-Scrolling zu "vertuschen", kann mit diesem Bit die Länge des Bildschirms von 25 auf 24 Zeilen verkürzt werden.

Bit 4: Screen enable - dieses Bit schaltet den Bildschirm an und aus. Wenn der Bildschirm abgeschaltet ist, führt der Computer alle Routinen wesentlich gleichmäßiger aus, da der Prozessor des C16 nicht mehr durch den TED gestoppt wird, damit dieser Zeit hat, das Videobild aufzubauen.

Bit 5: =0: Graphikmodus aus / =1: Graphikmodus ein

Bit 6: Cursor enable - ist dieses Bit gesetzt, so verschwindet der Cursor, bei gelöschtem Bit erscheint der Cursor, an der vorgegebenen Position (s.u.).

Bit 7: Funktion unbekannt

Register $07/07 $FF07 (65287) Multi-Function-Register II

Bits 0-2: Diese Bits bewirken eine horizontale Verschiebung des Bildschirms um jeweils ein Pixel.

Bit 3: Dieses Bit ist dafür zuständig, daß der Bildschirm 38 oder 40 Spalten breit ist. Dieses kommt dem Programmierer ebenfalls bei der Durchführung des Smooth-Scrollings zu gute.

Bit 4: Multicolor enable - wenn dieses Bit gesetzt ist, so erscheinen die Punkte der hochauflösenden Grafik doppelt so breit, allerdings kann dann ein Punkt mehr als 4 Farben annehmen.

Bit 5: Funktion unbekannt

Bit 6: Dieses Bit entscheidet, ob PAL oder NTSC-Norm ausgewählt ist.

Bit 7: Invert off - ist dieses Bit gesetzt, wird die hardwaremäßige Invertierung aller Zeichen, die einen größeren Bildschirmcode als 128 haben, ausgeschaltet. Statt dessen können Sie nun den Groß/Grafik und den Klein/Groß-Zeichensatz darstellen.

Register $08/08 $FF08 (65288) Tastaturregister

Dieses Register ist für die Auswertung der Tastatur zuständig. Die genaue Funktion dieses Registers würde an dieser Stelle den Rahmen des Buches sprengen.

Register $09/09 $FF09 (65289) Interrupt Register (IRR)

Dieses Register ist für das Interrupt-Handling zuständig. Soll ein Interrupt bei einem bestimmten Ereignis ausgelöst werden, so ist die dazu korrespondierende Bitnummer zusammen mit dem 7. gesetztem Bit in dieses Register zu schreiben. Den einzelnen Bitnummern entsprechen folgende Ereignisse:

Bit 0: keine Funktion
Bit 1: Rasterzeileninterrupt.
Bit 2: Light-Pen-Interrupt
Bit 3: Interrupt ausgelöst durch Timer A
Bit 4: Interrupt ausgelöst durch Timer B
Bit 5: keine Funktion
Bit 6: Interrupt ausgelöst durch Timer C
Bit 7: Interrupt-Flag

Beim Lesen dieses Registers können Sie durch das Bitmuster, welches sich ergibt, herausfinden, welcher Interrupt eingetreten ist. Falls ein Interrupt eingetreten ist, ist in jedem Fall zusätzlich Bit 7 gesetzt.

Register $0a/10 $FF0A (65290) Interrupt Register (IMR)

Bit 0: keine Funktion
Bit 1: Rasterinterrupt erlauben/nicht erlauben
Bit 2: Light-Pen-Interrupt erlauben/nicht erlauben
Bit 3: Timer 1 - Interrupt ein/aus
Bit 4: Timer 2 - Interrupt ein/aus
Bit 5: keine Funktion
Bit 6: Timer 3 - Interrupt ein/aus
Bit 7: -

Dieses Register ist die Maske des IRR. Ist hier ein Bit gelöscht, so ist der entsprechende Interrupt eingeschaltet, d.h. durch das entsprechende Ereignis kann zukünftig ein Interrupt ausgelöst werden. Ist das Bit gesetzt, so ist der Interrupt gesperrt.

Register $0b/11 $FF0B (65291) Rasterinterrupt

In diesem Register geben Sie die Rasterzeile an, bei deren Strahldurchlauf des Bildschirm-Elektronenstrahls ein Interrupt ausgelöst werden kann.

Register $0c/12 $FF0C (65292) Cursorposition High-Byte

In den unteren beiden Bits dieses Registers geben Sie die Bildschirmposition des hardwaremäßig darstellbaren Cursors an.

Register $0d/13 $FF0D (65293) Cursorposition Low-Byte

Dieses Register enthält die unteren 8 Bits der Bildschirmposition des Hardwarecursors.

Register $0e/14 $FF0E (65294) Tonhöhe - TG 1 (Bits 0-7)

In diesem Register können Sie die untersten 8 Bits der Tonhöhe unterbringen, wenn Sie über den Tongenerator 1 einen Ton ausgeben wollen.

Register $0f/15 $FF0f (65295) Tonhöhe - TG 2 (Bits 0-7)

In diesem Register können Sie die untersten 8 Bits der Tonhöhe unterbringen, wenn Sie über den Tongenerator 2 einen Ton ausgeben wollen.

Register $10/16 $FF10 (65296) Tonhöhe - TG 2 (Bits 8/9)

Dieses Register beherbergt die obersten zwei Bits der Tonhöhe von Tongenerator 2.

Register $11/17 $FF11 (65297) Multi-Function-Register III

Bit 0-3: Lautstärke (Werte von 0-8)
Bit 4: Tongenerator 1 ein/aus
Bit 5: Tongenerator 2 ein/aus
Bit 6: Tongenerator 2: Rauschen ein/aus
Bit 7: keine Funktion bekannt

Register $12/18 $FF12 (65298) Multi-Function-Register IV

Bit 0/1: Tonhöhe - Tongenerator 1 (Bits 8/9)
Bit 2: Zeichensatz in ROM oder RAM
Bit 3-7: Bits 11-15 der Basisadresse der hochauflösenden Graphik

Register $13/19 $FF13 (65299) Zeichensatz-Adresse

Bits 2-7 dieses Registers enthalten die Adreßbits 10-15 der Zeichengenerator-Basisadresse.

Register $14/20 $FF14 (65300) Basisadresse Farb-/Videoram

In diesem Register ist das High-Byte der Basisadresse des ganz normalen Farbspeichers und des Video-RAM niedergelegt, die bei der normalen Textdarstellung und der Graphik verwendet werden.

Register $15/21 $FF15 (65301) Hintergrundfarbe

Register $16/22 $FF16 (65302) Zeichenfarbe

Register $17/23 $FF17 (65303) Multicolor-Farbe 1

Register $18/24 $FF18 (65304) Multicolor-Farbe 2

Register $19/25 $FF19 (65305) Rahmenfarbe

Register $1a/26 $FF1A (65306) Bit-Mapping 1

Register $1b/27 $FF1B (65307) Bit-Mapping 2

Register $1c/28 $FF1c (65308) Aktuelle Rasterzeile Bit 8

Dieses Register gibt Ihnen Bit 8 der momentanen Position des Elektronenstrahls an, der das Bild auf dem Monitor/Fernseher aufbaut.

Register $1d/29 $FF1d (65309) Aktuelle Rasterzeile Bits 0-7

Wie Register 28, nur Bits 0-7

Register $1e/30 $FF1E (65310) Rasterspalte

Register $1f/31 $FF1F (65311) Vertikal-Adresse

2.2.2 Der Textbildschirm

Fangen wir zunächst mit der Textgrafik an, dem Grafikmodus also, den Sie sehen können, wenn Sie den Computer frisch eingeschaltet haben. Für jederman ist es fast normal, daß ein Drücken auf eine Buchstabentaste ein Darstellen dieses entsprechenden Zeichens auf dem Bildschirm zur Folge hat. Doch niemand macht sich so recht darüber Gedanken, wie diese Darstellung überhaupt zustande kommt.

Dazu müssen Sie nun folgendes wissen: Die Zeichen, die Sie auf Ihrem Bildschirm sehen können, werden in einem bestimmten Teil des Speichers Ihres Computer abgelegt. Dabei besitzt jedes Zeichen einen anderen Code. Dieser Speicher wird auch Bildschirmspeicher, oder Video-RAM bezeichnet, und ist wie folgt aufgebaut:

3072 3073 3074 3075 3076 3077 3078 3079 3080 ... bis 3111
3112
3152
3192
3232
3272
3212
3252
.
.
.
bis 4032

Die einzelnen Bildschirmcodes (die Codes also, die den einzelnen Zeichen entsprechen) können Sie übrigens in unserem Anhang nachschlagen.

Soll nun in der linken, oberen Ecke des Bildschirms ein 'A' erscheinen, so müssen Sie den Code, der diesem 'A' entspricht in den Bildschirmspeicher bringen. Das geschieht in BASIC mit dem POKE-Befehl. Um nun also ein 'A' in der linken, oberen Ecke erscheinen zu lassen, ist also folgender Befehl notwendig:

```
POKE 3072,1
```

Ähnlich wie die Speicherung der Zeichen vonstatten geht, funktioniert nun auch die Speicherung der Farbe und der Helligkeitswerte: Diese sind nämlich in dem sogenannten Farb-RAM gespeichert, das genauso angeordnet ist wie der Bildschirmspeicher, nur, daß es die Startadresse 2048 trägt. Doch in Codezuweisung der einzelnen Farbe gibt es Unterschiede. Wie Sie vielleicht wissen, besteht ein einziger dieser Codes aus einer Zahl der Wert zwischen 0 und 255 liegt. Da wir einen 8-Bit-Computer besitzen, ist dies ja auch nur logisch, den 2^8 ergibt = 256, also existieren auch 256 mögliche Zahlenkombinationen (die '0' mit eingeschlossen). In den unteren 4 Bits dieses Codes wird nun die Farbe abgelegt, in den folgenden 3 Bits die Helligkeitswerte und schließlich wird im 7. Bit auch noch gespeichert, ob das Zeichen blinken soll oder nicht. Ein Schaubild mag dieses verdeutlichen:

```
Bitnummer: 0  1  2  3  4  5  6  7
-----------------------------------
           F  F  F  F  H  H  H  F
           a  a  a  a  e  e  e  l
           r  r  r  r  l  l  l  a
           b  b  b  b  l  l  l  s
           e  e  e  e  i  i  i  h
                       g  g  g
                       k  k  k
                       e  e  e
                       i  i  i
                       t  t  t
```

Somit können wir eine Formel aufstellen, die die richtigen Bitkombinationen für Farbe und Helligkeit errechnet, und das Farb-RAM schreibt:

```
poke 2048+xk+40*yk,farbe+lum*16
```

Folgende Variablen werden hierbei gebraucht:

xk: X-Koordinate des Zeichens, das geändert werden soll
yk: Y-Koordinate des Zeichens
farbe: Farbe des Zeichens (0-15)
lum: Helligkeit (0-7)

Hinweis: Achten Sie stets darauf, daß der Helligkeitswert die zulässige Größe nicht überschreitet, da das Zeichen sonst anfängt zu blinken.

Damit wären nun alle Speicher, die für den Aufbau des Textbildschirms zuständig sind, erklärt. Doch halt! Woher weiß der Computer den überhaupt wie ein einziges Zeichen genau aussehen soll. Durch den Bildschirmcode wird ihm doch gezeigt, welches Zeichen er aus einer Vielzahl von möglichen auswählen soll. Es muß also noch einen Speicher geben, der das Aussehen aller Zeichen beinhaltet. Und in der Tat, ein solcher existiert wirklich. Dieser Speicher wird Zeichensatzspeicher, oder, das ist die wohl gebräuchlichere Wendung, Chargenerator genannt, und ist folgendermaßen aufgebaut:

2.2.3 Der Charactergenerator

```
...**...
..****..
.**..**.
.******.
.**..**.
.**..**.
.**..**.
........
```

Wie Sie sehen, ist die untere Matrixreihe beim 'A' frei. Das muß so sein, denn Ihr Computer kennt ja auch noch Zeichen, deren unteres Ende tiefer reicht als das des "A". Beispiele sind: "p q , ; ". Die Matrix des Buchstaben "p" sieht also so aus:

```
........
........
.******..
.**..**.
.**..**.
.******..
.**.....
.**.....
```

Nun wissen Sie also, wie die Zeichen im Zeichensatz Ihres Computers aufgebaut sind. Wir werden Ihnen allerdings noch ein kleines Programm vorstellen, das Ihnen jede Zeichenmatrix vergrößert und ausgibt:

```
100 rem ------------------------------
110 rem -- auslesen und vergroessern --
120 rem ---                        ---
130 rem --   der gesuchten zeichen  ---
140 rem ------------------------------
150 :
160 ad=8192 :rem startadresse der zeichen
170 :
180 scnclr
190 color 4,1:color0,1
200 color 1,7
210 :
220 print" welches zeichen?"
230 getkey a$
232 :
237 gosub 380
```

```
238 :
239 print
240 :
250 rem ------------------------------
260 rem --- zeichenmatrix berechnen ---
270 rem ------------------------------
280 :
290 :for t=0to7                          :rem 8 zeilen
300 : for i=0to7                         :rem 8 spalten
310 :  locate i,t:b=rdot(2)              :rem auf punkt testen
320 :   ifb=0then print".";:else print"*";:rem ausdruck
330 :  next i                            :rem spaltenschleife
340 : print                              :rem zeilenvorschub
350 :next t                              :rem zeilenschleife
360 :
370 getkey a$:rem auf taste warten
371 run      :rem und neu starten
372 :
373 rem ------------------------------
374 rem ----   zeichen "kopieren"  ----
375 rem ------------------------------
376 :
380 graphic1    :rem grafik anschalten
385 char1,0,0,a$:rem buchstaben in grafik schreiben
386 graphic0    :rem grafik ausschalten
390 return      :rem weiter im hauptprogramm
```

Nun auch zu diesem Programm die Erklärung der einzelnen Zeilen:

160	Festlegen der Startadresse der "Zeichensatzkopie".
180-200	Bilschirm löschen, Farben setzen.
220	Frage nach dem gewünschten Zeichen stellen.
230	Auf eingabe des Zeichens warten.
237	Unterprogramm aufrufen.
239	Eine neue Zeile anfangen.
290	Zeilen-Schleife.
300	Spalten-Schleife.

```
310         Graficursor auf Buchstaben positionieren und
            Punkt testen.
320         Fals der Punkt nicht gesetzt ist, einen Punkt (.),
            andernfals einen Stern (*) ausgeben.
330         Zur nächsten Spalte
340         Zeilenvorschub
350         Zur nächsten Zeile
370         Auf Tastendruck warten
371         RUN ausführen
            Unterprogramm zum Darstellen des gewünschten
            Zeichens in der HiRes-Grafik
380         Einschalten der Grafik
390         Schreiben des gewünschten Zeichens an die Posi-
            tien (0,0); entspricht der Adresse 8192
400         Ausschalten der Grafik
410         Rückkehr zum Hauptprogramm
```

Der Char-Zoomer ist sicherlich so einfach zu verstehen, daß ich hier auf eine Bedienungsanleitung verzichten möchte.

Dieses Programm kopiert den Zeichensatz natürlich nicht richtig, sondern wendet einen kleinen Trick an, es stellt das gewünschte Zeichen in der HiRes-Grafik dar, die, wie wir noch sehen werden, vom Aufbau her dem Chargenerator gleicht. Um den Zeichensatz zu kopieren, damit wir ihn anschließend für unsere eigenen Belange modifizieren können, müssen wir erst einmal wissen, wo dieser liegt und wohin wir ihn kopieren wollen. Frage 1 ist nach einem kurzen Blick ins C16-Handbuch beantwortet. Auf der Seite 228 finden wir eine grobe übersicht über das C16 ROM. Wir entdecken die Character-Tabelle (Zeichensatz) an der Adresse $D000 oder 53248. Dieser erstreckt sich bis zur Adresse $D400 oder 54272.

Nun zur Frage 2: Um diese Frage zu beantworten, müssen wir ein wenig überlegen. Der Zeichensatz besteht aus 128 Zeichen. An dieser Stelle ein Hinweis: die reversen Zeichen erzeugt der Videocontroller selber. Das sind die Zeichen die die Bildschirmcodes von 128-255 tragen. Jedes dieser 128 Zeichen braucht 64 Bit, also 8 Byte. 128*8 ergibt 1024. Wir brauchen

also 1 KByte Speicherplatz für die Zeichensatzkopie. Aus Adressierungsgründen kann dieser Zeichensatz nur alle 1024 Bytes im Speicher liegen. Mögliche Adressen sind also:

0- 1023	(VORSICHT, dient als Systemspeicher)
1024 - 2047	(VORSICHT, dient ebenfals als Systemspeicher)
2048 - 3071	(VORSICHT, enthält Farbwerte des Textschirm
3072 - 4095	(VORSICHT, Textschirm)
4096 - 5119	(BASICSPEICHER)
5120 - 6143	(BASICSPEICHER)
6144 - 7167	(BASIC- oder HELLIGKEITSSPEICHER der Grafik)
7168 - 8191	(BASIC- oder FARBSPEICHER der Grafik)
8192 - 9215	(BASIC- oder GRAFIKSPEICHER)
9216 - 10239	(BASIC- oder GRAFIKSPEICHER)
10240 - 11263	(BASIC- oder GRAFIKSPEICHER)
11264 - 12287	(BASIC- oder GRAFIKSPEICHER)
12288 - 13311	(BASIC- oder GRAFIKSPEICHER)
13312 - 14335	(BASIC- oder GRAFIKSPEICHER)
14336 - 15359	(BASIC- oder GRAFIKSPEICHER)
15360 - 16383	(BASIC- oder GRAFIKSPEICHER)

(Sollten Sie glücklicher Besitzer eines Plus/4, oder eines erweiterten C16 sein, so reicht diese Tabelle natürlich noch weiter.)

Wie Sie sehen, müssen Sie bei den oben genannten Adressen einige Einschränkungen in Kauf nehmen. Da aber wohl niemand Hochauflösende Grafik und selbstdefinierten Zeichensatz in einem Programm zusammen gebraucht, bietet sich der Bereich von 15360 bis 16383 für die Zeichensatzkopie an.
Nun aber endlich zum Zeichensatzkopierprogramm :

```
10 for i= 0 to 1023     :rem 1024 Werte kopieren
20 : alt=53248+i        :rem Start des Zeichensatzes im ROM
30 : neu=15360+i        :rem Start der Zeichensatzkopie
```

```
40 : poke neu,peek(alt):rem kopieren
50 next i              :Schleife schließen
```

Doch schon wieder erscheint ein Problem.

1. Dieses Programm dauert zu lange (18 sec.)
2. Dieses Programm funktioniert nicht, da der C16 und Plus/4 durch ein sog. BANK-SWITCHING beim PEEK-Befehl immer auf das RAM zugreift. Das heißt, daß der Chargenerator sozusagen 'ausgeblendet' wird.

Wir müssen also wohl oder übel auf die Maschinensprache zugreifen.
Hier zuerst das Assemblerlisting für die Kenner unter Ihnen:

2.2.3.1 Eine Charcopyroutine in Assembler

```
10:   0332                *=   $0332   ;routine in den
                                       ;kassettenpuffer
20:   0332                .opt p1,oo   ;in den speicher assemblieren
22:   0332                .sym 2
23:   0332                .tit "char-copy"
30:             ;
40:             ;**********************************
50:             ;***                             ***
60:             ;***   char-copy routine v1.0     ***
70:             ;***                             ***
80:             ;***  copy rom ==> ram (15360)   ***
90:             ;**********************************
100:            ;
130:  d000      char    =    $d000   ;start char-generator
140:  3c00      neu     =    15360   ;neuer speicherbereich des
                                       ;char-generators
150:  07f8      steuer  =    $7f8    ;umschalten rom/ram
160:  00f6      merk1   =    $f6     ;zeiger auf zeichen
170:  00f7      merk2   =    $f7     ;zeiger auf zeichen
180:  00f8      merk3   =    $f8     ;zeiger auf zeichen
```

```
190:   00f9          merk4    =   $f9      ;zeiger auf zeichen
200:                 ;
210:                 ;***** programm *****
220:                 ;
230:   0332 a2 00            ldx  #<char  ;startadresse in
240:   0334 a0 d0            ldy  #>char  ;lo- & hi-byte
250:   0336 86 f6            stx  merk1   ;merken
260:   0338 84 f7            sty  merk2   ;merken
270:                 ;
280:   033a a2 00            ldx  #<neu   ;neu startadresse in
290:   033c a0 3c            ldy  #>neu   ;lo- & hi-byte
300:   033e 86 f8            stx  merk3   ;merken
310:   0340 84 f9            sty  merk4   ;merken
320:                 ;
330:   0342 78               sei          ;interrupt verbieten
340:   0343 ad f8 07         lda  steuer  ;alten wert
350:   0346 48               pha          ;auf stack legen
360:   0347 a9 00            lda  #00     ;auf rom
370:   0349 8d f8 07         sta  steuer  ;schalten
380:                 ;
390:   034c a2 04            ldx  #4      ;schleife 4 mal
400:   034e a0 00            ldy  #0      ;y-reg. loeschen
410:   0350 b1 f6    copy    lda (merk1),y;charrom lesen
420:   0352 91 f8            sta (merk3),y;und speichern
430:   0354 c8               iny          ;y-reg. erhoehen
440:   0355 d0 f9            bne  copy    ;<>0; dann nach copy
450:   0357 e6 f7            inc  merk2   ;lo-byte erhoehen
460:   0359 e6 f9            inc  merk4   ;lo-byte erhoehen
470:   035b ca               dex          ;x-reg. erniedriegen
480:   035c d0 f2            bne  copy    ;noch nicht 0; dann copy
490:                 ;
500:   035e 68               pla          ;alter steuerwert vom stack
510:   035f 8d f8 07         sta  steuer  ;und zurueckschreiben
520:   0362 58               cli          ;interrupt freigeben
530:   0363 60               rts          ;zurueck zum basic
]0332-0364
no errors
```

```
symboltable:
copy      0350   merk4     00f9
merk3     00f8   merk2     00f7
merk1     00f6   steuer    07f8
neu       3c00   char      d000

8 symbols defined
```

Und hier nun der BASIC-Lader:

```
100 rem ********************************
110 rem ***                            ***
120 rem ***        char-copy           ***
130 rem ***        ---------           ***
140 rem ***                            ***
150 rem ***   rom --> ram (15360)      ***
160 rem ***                            ***
170 rem ***       (basiclader)         ***
180 rem ********************************
190 :
200 input"startadresse";sa
210 :
220 if sa =0 then sa=dec("0332"):rem kassettenpuffer
230 :
240 print"data werden gelesen !!"
250 :
260 for i= sa to sa+dec("0031")
270 ::: read a:poke i,a
280 ::: pr=pr+a
290 next
300 :
310 if pr<> 7710 then print"data o.k. aufruf mit sys";sa
320 :
330 print"data error!!!!":end
340 :
350 :
360 rem **** assembler data ****
370 :
```

```
380 data 162,000,160,208,134,246,132,247
390 data 162,000,160,060,134,248,132,249
400 data 120,173,248,007,072,169,000,141
410 data 248,007,162,004,160,000,177,246
420 data 145,248,200,208,249,230,247,230
430 data 249,202,208,242,104,141,248,007
440 data 088,096,000,000,000,000,000,000
```

Wenn Sie den BASIC-Lader nun abgetippt, gespeichert und gestartet haben, sollte der Zeichensatz nun ab der Adresse 15360 im RAM stehen. Überprüfen Sie dies, indem Sie folgenden Befehl eingeben:

```
GRAPHIC 1,0
```

Sie sehen im unteren Teil des Bildschirms nun die Zeichensatzkopie. Jetzt sollten Sie dem Computer wieder mitteilen, daß er den Grafikspeicher als BASIC-Speicher ansehen soll. Die geschieht mit dem GRAPHIC-CLR-Befehl.
Bevor Sie sich jedoch daranmachen, ein Programm zu schreiben, sollten Sie den Zeichensatz vor dem Überschreiben durch Variablen schützen. Dies erledigen diese 3 Befehle:

```
POKE 55,246:POKE 56,59:CLR.
```

Damit legen Sie das Ende des BASIC-Speichers nach 15350.

Nun geht es mit PEEK & POKE dem alten Zeichensatz an den Kragen.

Das erste, was wir nun machen wollen, ist uns eine Skizze des neuen Zeichen anzufertigen. Diese Skizze sollte wie folgt aussehen.

```
        BIT      :
    7 6 5 4 3 2 1 0 :Nummer
    ----------------:
                    :
                    : -1-
                    :
  *   * *   *       : -2-
                    :
      * *           : -3-
                    :
  * *     * *       : -4-
                    :
  * *     * *       : -5-
                    :
      * *           : -6-
                    :
  *   * *   *       : -7-
                    :
                    : -8-
```

Dieses Zeichen müssen wir jetzt computergerecht zubereiten. Das funktioniert wie folgt: Für jeden gesetzten Punkt setzen wir ein BIT, für jeden gelöschten Punkt löschen wir ein BIT. Die nun erstellten Bitmuster wandeln wir dann in Dezimalzahlen um.

```
1.Reihe : 00000000 =0*128 +0*64 +0*32 +0*16 +0*8 +0*4 +0*2 +0*1 =   0
2.Reihe : 01011010 =0*128 +1*64 +0*32 +1*16 +1*8 +1*4 +0*2 +1*1 =  93
3.Reihe : 00011000 =0*128 +0*64 +0*32 +1*16 +1*8 +0*4 +0*2 +0*1 =  24
4.Reihe : 01100110 =0*128 +1*64 +1*32 +0*16 +0*8 +1*4 +1*2 +0*1 = 102
5.Reihe : 01100110 =0*128 +0*64 +0*32 +1*16 +1*8 +0*4 +0*2 +0*1 = 102
6.Reihe : 00011000 =0*128 +0*64 +0*32 +1*16 +1*8 +0*4 +0*2 +0*1 =  24
7.Reihe : 01011010 =0*128 +1*64 +0*32 +1*16 +1*8 +1*4 +0*2 +1*1 =  93
8.Reihe : 00000000 =0*128 +0*64 +0*32 +0*16 +0*8 +0*4 +0*2 +0*1 =   0
```

Jetzt sollten wir uns entscheiden, welches Zeichen des alten Zeichensatzes verändert werden soll. Ich werde hier den "Pfeil nach oben" (^) verändern. Die Zeichensatzkopie liegt ab 15360

im Speicher und in der Kode-Tabelle Ihres Handbuches finden Sie auf Seite 213 den Poke-Kode für "^"(30). Da jeder Buchstabe 8 Bytes belegt, beginnt das alte Bitmuster des "^" ab 15360 + 8 * 30, also 15600. Endlich können wir ein neues Zeichen definieren.

```
10 for i= 15600 to 15600+7:rem 8 Werte
20 :    read a             :rem Bitmuster lesen
30 :    poke i,a           :rem und 'einpoken'
40 next i                  :rem schleife schließen
50 end
60 rem data fuer das neue Zeichen
70 data  0, 93, 24,102,102, 24, 93,  0
```

Nun ist der große Augenblick gekommen, das neue Zeichen sollte beim Druck auf die "^"-Taste erscheinen. Sollte - denn es erscheint nicht. Ihr Computer greift immer noch auf den ROM-Zeichensatz zurück. Wir müssen dem Rechner erst mitteilen, daß wir jetzt gern unseren Zeichensatz benutzt hätten. Dies wiederum erreichen wir durch Ändern zweier Speicherstellen, welche Register des TED-Videochips sind und die Basisadresse des Zeichengenerators enthalten:

```
v=65280:pokev+18,peek(v+18) and not 4:poke v+19,15360/256.
```

Nun erscheint auch beim Druck auf die "^"-Taste das neue Zeichen.

Anwendungen für einen veränderbaren Zeichensatz gibt es viele:

- Eine deutsche Tastatur
- Die von Druckern bekannte IC-SCHRIFT auf Ihrem C16 und Plus/4
- Und natürlich auch das Erfinden eigener Spielhelden, die eventuell auch aus mehreren Zeichen bestehen können.

2.2.3.2 Ein komfortabler Chargenerator

Damit Sie nicht ständig selbst zu Blatt und Stift greifen müssen, wenn Sie Ihren persönlichen Zeichensatz kreieren wollen, haben wir zu guter letzt noch ein Programm für Sie entwickelt, das es Ihnen erlaubt, Ihren eigenen Zeichensatz direkt am Bildschirm zu entwerfen. Die Anleitung zu diesem Programm, finden Sie dann im Anschluß daran. Doch nun zu unserem Chardesigner:

```
10 poke 55,0:poke 56,46:clr:rem zeichensatzkopie schuetzen
20 gosub  2460:rem maschinenprogramm
30 sys dec("3000"):sys dec("3000"):rem starten
40 :
50 i$=chr$(125):h$=chr$(96)
60 e1$=chr$(176):e2$=chr$(174)
70 e3$=chr$(173):e4$=chr$(189)
80 r1$=chr$(117):r2$=chr$(105)
90 r3$=chr$(106):r4$=chr$(107)
100 fori =1 to 27:f1$=f1$+(h$):next
110 f2$=left$(f1$,8)
120 :
130 scnclr:printchr$(142)chr$(8)
140 color0,1:color1,6,4:color4,3,5
150 :
160 char1,6,0,r1$+f1$+r2$
170 char1,6,1,i$+" c h a r - d e s i g n e r "+i$
180 char1,6,2,i$:char1,34,2,i$
190 char1,6,3,i$+"      c-16  /  plus 4      "+i$
200 char1,6,4,r3$+f1$+r4$
210 :
220 for i=0to 2
230 : char1,i*13,6 ,    "feld"+str$(i+1)
240 : char1,i*13, 8,    r1$+f2$+r2$
250 :: for t=0 to 7
260 ::  char1,i*13, 9+t,i$+"........"+i$
270 :: nextt
280 : char1,i*13,17,    r3$+f2$+r4$
290 next
300 :
```

```
310 char1, 0,7,chr$(18)+"design"
320 char1,13,7,"zur zeit"
330 char1,26,7,"orginal"+chr$(146)
340 :
350 for i=7 to 0 step -1
360 : char 1,10, 9+i,str$(i)
370 : char 1,23, 9+i,str$(i)
380 : char 1,36, 9+i,str$(i)
390 : char 1, 0+i,18,str$(i)
400 : char 1,13+i,18,str$(i)
410 : char 1,26+i,18,str$(i)
420 next
430 :
440 char1,2,20,"aktueller bildschirmcode:   0"
450 char1,32,19,e1$+h$+e2$
460 char1,32,20,i$+"@"+i$
470 char1,32,21,e3$+h$+e4$
480 clr  :rem platz schaffen
490 x=1:y=1:x1=1:y1=2:cu=46
500 :
510 poke 1319,19:poke 239,1:rem home bei naechster getkey
    anweisung
520 goto 1810:rem zeichen auslesen
530 :
540 getkey a$
550 :
560 : if a$=" "then cu=81:goto 1350
570 : if a$=chr$(160) then cu=46:goto 1350
580 : if a$=chr$(157) then 780 :rem c links
590 : if a$=chr$( 29) then 820 :rem c rechts
600 : if a$=chr$(145) then 860 :rem c oben
610 : if a$=chr$( 17) then 900 :rem c unten
620 : if a$=chr$( 13) then 940 :rem return
630 : if a$=chr$(147) then 970 :rem clr
640 : if a$=chr$( 19) then 1030:rem home
650 : if a$="2"       then 1070:rem kopieren feld 2
660 : if a$="3"       then 1100:rem kopieren feld 3
670 : if a$="s"       then 2130:rem save
680 : if a$="l"       then 2220:rem load
690 : if a$=chr$( 27) then 2000:rem uebernehmen
```

```
700 : if a$="i"       then 1220:rem invertieren
710 : if a$="n"       then 1670:rem neues zeichen
720 : if a$="h"       then 1390:rem spiegel hor
730 : if a$="v"       then 1530:rem spiegel ver
740 :
750 goto 540
760 :
770 rem -------- cursor left  --------
780 x=x-1:if x<1 then x=8:goto 860
790 goto 2360    :rem cursor setzen
800 :
810 rem -------- cursor right --------
820 x=x+1:if x>8 then x=1:goto 900
830 goto 2360    :rem cursor setzen
840 :
850 rem -------- cursor up    --------
860 y=y-1:if y<1 then y=8
870 goto 2360    :rem cursor setzen
880 :
890 rem -------- cursor down  --------
900 y=y+1:if y>8 then y=1
910 goto 2360    :rem cursor setzen
920 :
930 rem -------- return       --------
940 x=1:goto 900
950 :
960 rem -------- clr / home   --------
970 :for i=1to8
980 ::for h=1to8
990 :: char1,h,i+8,"."
1000 ::next h
1010 :next i:cu=46
1020 :
1030 x=1:y=1
1040 goto 2360
1050 :
1060 rem -------- f.2 kopieren --------
1070 x2=13:goto 1120
1080 :
```

```
1090 rem -------- f.3 kopieren --------
1100 x2=26
1110 :
1120 poke 3072+x1+40*(y1+8),cu
1130 :fort=9 to 16
1140 ::fori=x2+1 to x2+8
1150 :: b=peek(3072+i+40*t)
1160 :: poke 3072+i-x2+40*t,b
1170 ::next
1180 :next
1190 :
1200 goto 2370
1210 :
1220 rem -------- invertieren  --------
1230 :
1240 poke 3072+x1+40*(y1+8),cu
1250 :for i=9to16
1260 ::for t=1to8
1270 :: b=(3072+i*40+t)
1280 :: if peek(b)=46 then poke b,81:else poke b,46
1290 ::nextt
1300 :nexti
1310 :
1320 goto 2370
1330 :
1340 rem -------- punkt s/l   ---------
1350 x1=x:y1=y
1360 goto 820
1370 :
1380 rem -------- spiegeln h  ---------
1390 poke 3072+x1+40*(y1+8),cu
1400 :fori=1 to 4
1410 ::fort=9 to 16
1420 :: a=3072+i+40*t
1430 :: a2=3072+9-i+40*t
1440 :: b=peek(a):b2=peek(a2)
1450 :: poke a,b2:pokea2,b
1460 ::next t
1470 :next i
1480 cu=peek(3072+x1+40*(y1+8))
```

```
1490 :
1500 goto 2360
1510 :
1520 rem -------- spiegeln v ---------
1530 poke 3072+x1+40*(y1+8),cu
1540 :fori=1 to 8
1550 ::fort=9 to 12
1560 :: a=3072+i+40*t
1570 :: a2=3072+i+40*(25-t)
1580 :: b=peek(a):b2=peek(a2)
1590 :: poke a,b2:pokea2,b
1600 ::next t
1610 :next i
1620 cu=peek(3072+x1+40*(y1+8))
1630 :
1640 goto 2360
1650 :
1660 rem -------- neues zeichen--------
1670 color 4,3,5
1680 char 1,28,20,right$("   "+str$(z),3):poke3905,z
1690 getkey a$
1700 :  if a$=chr$(13) then 1810
1710 :  if a$=chr$(43) then z=z+1
1720 :  if a$=chr$(45) then z=z-1
1730 :  if a$=chr$(219)then z=z+10
1740 :  if a$=chr$(221)then z=z-10
1750 :  if z>127 then z=z-128
1760 :  if z<0   then z=z+128
1770 :
1780 goto 1670
1790 :
1800 rem ------- vergroessern----------
1810 ad=15360
1820 :for t=0 to7
1830 ::for i=0 to7
1840 :: x2=14+i:y2=9+t
1850 :: b=(peek(ad+z*8+t)and 2~(7-i))
1860 :: if b=0 then char 1,x2,y2,".":else char1,x2,y2,chr$(113)
1870 :next:next
1880 :
```

```
1890 ad=14336
1900 :for t=0 to7
1910 ::for i=0 to7
1920 :: x2=27+i:y2=9+t
1930 :: b=(peek(ad+z*8+t)and 2^(7-i))
1940 :: if b=0 then char 1,x2,y2,".":else char1,x2,y2,chr$(113)
1950 :next:next
1960 color4,1:goto 540
1970 :
1980 rem ------- z. uebernehmen----
1990 color 4,3,5
2000 color 4,3,5
2010 poke 3072+x1+40*(y1+8),cu
2020 :for t=0to 7:e=0
2030 ::for i= 7to0 step -1
2040 :: if (peek(3072+i+1+(t+9)*40))=81 then e=e+2^(7-i)
2050 ::next
2060 : poke 15360+z*8+t,e
2070 :next
2080 :
2090 color 4,1
2100 goto 2360
2110 :
2120 rem ------ save        ---------
2130 gosub 2310
2140 scnclr
2150 print"monitor"
2160 char1,0,6,"s"+chr$(34)+na$+chr$(34)+"08 3c00 3fff"
2170 char1,0,9,"x"
2180 char1,0,11,"goto 50"
2190 fori=1to4:poke1319+i,13:next:poke1319,19:poke239,5:end
2200 :
2210 rem ------ load         ---------
2220 gosub 2310
2230 scnclr
2240 print"monitor
2250 char1,0,6,"l"+chr$(34)+na$+chr$(34)+"08"
2260 char1,0, 9,"x"
2270 char1,0,11,"goto 50"
2280 fori=1to4:poke1319+i,13:next:poke1319,19:poke239,5:end
```

```
2290 :
2300 rem ----- filename --------
2310 scnclr
2320 input"filename";na$
2330 return
2340 :
2350 rem ------ cursor setzen -------
2360 poke 3072+x1+40*(y1+8),cu
2370 c = (3072+x+40*(y+8))
2380 cu=peek(c)
2390 poke c,cu or 128
2400 x1=x:y1=y
2410 goto 540
2420 :
2430 rem ----------------------------
2440 rem ---- char-copy routine -------
2450 rem ----------------------------
2460 :
2470 restore 2530
2480 for i=dec("3000") to dec("302b")
2490 : read a$:poke i,dec(a$)
2500 next
2510 return
2520 :
2530 data 78,a0,00,b9,00,d0,99,00,38
2540 data 99,00,3c,b9,00,d1,99,00,39
2550 data 99,00,3d,b9,00,d2,99,00,3a
2560 data 99,00,3e,b9,00,d3,99,00,3b
2570 data 99,00,3f,88,d0,d9,58,60,00
```

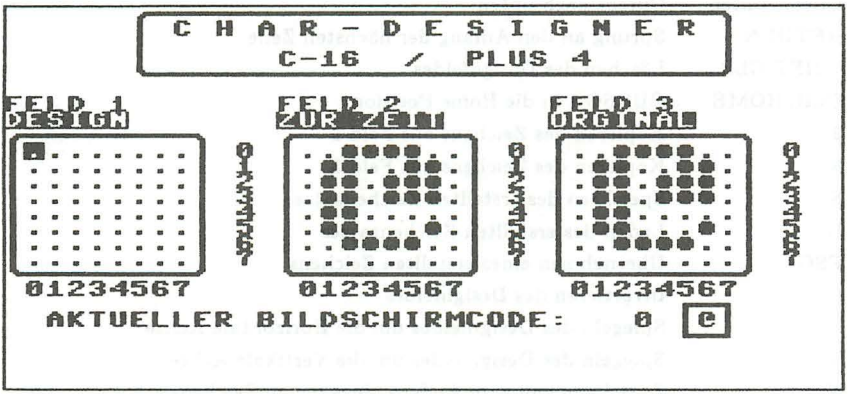
Hier nun wieder die Dokumentation des vorgestellten Programmes:

10	Schützen des Zeichensatzkopierprogrammes und der beiden Zeichensatzkopien
20	Assembler-Routine einlesen
30	Und starten
50-110	Festlegen einiger Strings für den Bildschirmaufbau

130	Löschen des Bildschirm; umschalten auf Großschrift; diese fixieren
140	Farben setzen
160-470	Bildschirm aufbauen
480	Platz schaffen durch löschen aller bisher definierten Variablen (Strings)
490	Festlegen der Variablen für die Cursorsteuerung
510	HOME in den Tastatur-Puffer, wird dann bei der nächsten GETKEY Anweisung ausgeführt
520	Unterprogramm zum Auslesen der Zeichen aufrufen
540	Auf Befehl Warten
560-750	Auswertung der gedrückten Taste
770-790	Cursor nach links
810-830	Cursor nach rechts
850-870	Cursor nach oben
890-910	Cursor nach unten
930-950	Return ausgeben
960-1040	Löschen des Designfeldes
1030-1040	Cursor in die HOME-Position
1070	Variablen für Kopierroutine Feld 2 nach Feld 1
1100	Variablen für Kopierroutine Feld 3 nach Feld 1
1120-1180	Eigentliche Felder-Kopierroutine
1240-1320	Invertierungs-Routine
1350,1360	Punkt setzen oder Löschen (je nachdem ob in der Befehlsauswertungsschleife der Variable CU der Wert 81 [setzen] oder 46 [löschen] zugewiesen worden ist)
1390-1500	Spiegeln des Designfeldes um die Horizontale
1500-1640	Spiegeln des Designfeldes um die Vertikale
1660-1780	Auswahl eines neuen Zeichens
1800-1960	Auslesen und vergrößern des gewählten Zeichens. Einmal aus dem Veränderten Zeichensatz und ein zweitesmal aus dem Orginalzeichensatz.
1990-2100	Übernehmen des erstellten Zeichens durch POKEn der BITmuster in die zweite Zeichensatzkopie
2120-2190	Speichern des zweiten Zeichensatzes durch aufrufen des MONITORS mit Hilfe der Tastaturpuffermethode.

2220-2280 Laden des Zeichensatzes durch aufrufen des Monitors mit Hilfe der Tastaturpuffermethode.
2310-2330 Eingeben des Filenamens für LOAD und SAVE
2360-2410 Setzen des Cursors durch Invertieren der Cursorposition
2430-2570 Einlesroutine der Char-Copy Routine und Assembler DATA

Kurz noch ein Wort zur Zeichensatz-Kopierroutine. Diese ist etwas anders aufgebaut als die Char-Copy-Routine aus dem Grundlagenkapitel. Da der ROM-Zeichensatz zweimal kopiert werden muß, andrerseits die Kopierroutine nicht zu lang werden sollte (wer will schon gerne 100 nichtssagende DATA abtippen), wurde die indirekt indizierte Adressierungsmethode durch eine simple X-indizierte Schleife ersetzt.



Auch hier wieder die Programmbeschreibung:

Nach dem Start des Programmes baut sich zuerst der Bildschirm auf und das erste Zeichen (der "Klammeraffe") wird vergrößert auf den Bildschirm gebracht. Sollten Sie dabei in einem der beiden Kopierfelder ein Durcheinander entdecken, welches dem "Klammeraffen" nicht ähnelt, ist das Programm nochmals zu starten.

Schließlich schaltet die Rahmenfarbe auf Schwarz, ein Zeichen dafür, daß der Computer für weitere Befehle bereit ist. Sie können nun durch Drücken verschiedener Tasten verschiedene Funktionen aufrufen. Dies möchte ich hier kurz vorstellen:

SPACE	Setzen eines Punktes
SHIFT-SPACE	Löschen eines Punktes
Cursor links	Cursor nach links
Cursor rechts	Cursor nach rechts
Cursor oben	Cursor nach oben
Cursor unten	Cursor nach unten
RETURN	Sprung an den Anfang der nächsten Zeile
SHIFT CLR	Löschen des Designfeldes
CLR/HOME	CURSOR in die Home Position
2	Kopieren des Zeichens aus Feld 2
3	Kopieren des Zeichens aus Feld 3
S	Speichern des erstellten Zeichensatzes
L	Laden des erstellten Zeichensatzes
ESC	Übernehmen eines erstellten Zeichens
I	Invertieren des Designfeldes
H	Spiegeln des Designfeldes um die Horizontale Achse
V	Spiegeln des Designfeldes um die Vertikale Achse
N	Zwischenmenü zum Ändern eines neuen Zeichens.
+	Erhöhen des Zeichencodes um 1 (nur wenn vorher N gedrückt war).
-	Erniedrigen des Zeichencodes um 1 (nur wenn vorher N gedrückt war).
SHIFT +	Erhöhen des Zeichencodes um 10 (nur wenn vorher N gedrückt war).
SHIFT -	Ernidriegen ds Zeichencodes um 10 (nur wenn vorher N gedrückt war).

RETURN Übernahme des neue Zeichencodes (nur wenn vorher N gedrückt war).

Bei der Ausführung der meisten dieser Befehle schaltet der Computer die Rahmenfarbe auf Rot um. Sie müssen dann warten, bis der Computer wieder Schwarz als Rahmenfarbe ausgibt.

Danach können Sie das Zeichen verändern oder mittels der ESC-Taste übernehmen. Das kreierte Zeichen wird nun in den Zeichensatz übernommen. Durch Druck auf die N-Taste besteht die Möglichkeit, ein anderes Zeichen zu verändern. Auswählen können Sie dieses Zeichen mit + , - , SHIFT + , SHIFT -. Wenn Sie sich also für das Fragezeichen entschieden haben (Code 63) können Sie dieses Zeichen dann durch Drücken von RETURN in die Kopierfelder bringen. Alle Änderungen beziehen sich dann auf das Fragezeichen, dieses wird also durch Ihr neues Zeichen beim Drücken von ESC überschrieben.

Die weiteren Befehle erklären sich von selbst, auf sie möchte ich hier nicht weiter eingehen. Probieren Sie einfach alles aus. Und zur Not gibt es ja immer noch den Resetknopf...

Wollen Sie den geänderten Zeichensatz dann in Ihren Programmen verwenden, müssen Sie diesen entweder mit

LOAD"Name",8,1 bzw. LOAD"Name",1,1

oder im MONITOR mit

L"Name"08 bzw. L"Name"01

einladen. Wie Sie auf den neuen Zeichensatz umschalten haben Sie sicherlich schon auf den vorherigen Seiten gelesen. Noch einmal zur Erinnerung:

```
POKE 65298,PEEK(65298) AND NOT 4:POKE 65299,15360/256
```

Sie sollten allerdings niemals bei verlegtem Zeichensatz auf die hochauflösende Grafik schalten, da dieses den TED (Videchip des C 16 und Plus/4) zum Absturz bringt. Dieses macht sich dadurch bemerkbar, daß beim Umschalten auf den Textbildschirm eine fernsehähnliche Bildstörung vorliegt. Abgeschaltet wird diese durch Eingabe der oben erwähnten POKEs.

Nun wünsche ich Ihnen viel Spaß beim Erschaffen neuer Zeichensätze.

2.2.4 Die hochauflösende Grafik

Zum Schluß sollten wir noch über die letzte Art der Bildschirmdarstellung reden, über die hochauflösende Grafik. Hier wird das ganze etwas kompliziert, weil ja nun nicht mehr ganze Zeichen auf einmal, sonder nur einzelne Punkte dargestellt werden.

Nun, auch dieses Problem hat man ziemlich rationell gelöst, denn bei dieser Art der Grafikdarstellung werden jeweils 8 Pixel zu einem Byte zusammengefaßt. Ein Schaubild soll die genaue Speichereinteilung des Video-RAM in der hochauflösenden Grafik verdeutlichen:

```
        Reihe 1  76543210 76543210 76543210 76543210 ...
        Reihe 2  76543210 76543210 76543210 76543210 ...
        Reihe 3  76543210 76543210 76543210 76543210 ...
        Reihe 4  76543210 76543210 76543210 76543210 ...
Zeile 1 Reihe 5  76543210 76543210 76543210 76543210 ...
        Reihe 6  76543210 76543210 76543210 76543210 ...
        Reihe 7  76543210 76543210 76543210 76543210 ...
        Reihe 8  76543210 76543210 76543210 76543210 ...

        Reihe 1  76543210 76543210 76543210 76543210 ...
        Reihe 2  76543210 76543210 76543210 76543210 ...
        Reihe 3  76543210 76543210 76543210 76543210 ...
```

```
        Reihe 4  76543210 76543210 76543210 76543210 ...
Zeile 2 Reihe 5  76543210 76543210 76543210 76543210 ...
        Reihe 6  76543210 76543210 76543210 76543210 ...
        Reihe 7  76543210 76543210 76543210 76543210 ...
        Reihe 8  76543210 76543210 76543210 76543210 ...
```

Während im vorhergehenden Schaubild gezeigt wurde, wie die einzelnen Punkte bitweise abgelegt wurde, verdeutlichts das nächste Schaubild, wie die einzelnen Bytes hintereinander angeordnet sind:

```
Spalte:   0     1     2     3  .....          39

          8192  8200  8207  8215 .....        8504
          8193  8201  8208  8216 .....        8505
          8194  8202  8209  8217 .....        8506
          8195  8203  8210  8218 .....        8507
Zeile 1   8196  8203  8211  8219 ..... b i s  8508
          8197  8204  8212  8220 .....        8509
          8198  8205  8213  8221 .....        8510
          8199  8206  8214  8222 .....        8511

          8512  8520  8528  8536 .....        8824
          8513  8521  8529  8537 .....        8825
          8514  8522  8530  8538 .....        8826
          8515  8523  8531  8539 .....        8827
Zeile 2   8516  8524  8532  8540 ..... b i s  8828
          8517  8525  8533  8541 .....        8829
          8518  8526  8534  8542 .....        8830
          8519  8527  8535  8543 .....        8831

Zeile 3          8832 - 9151

Zeile 4                .
Zeile 5                .
Zeile 6                .
                     b i s
```

```
Zeile 25          15872 - 16383
```

Ziel ist es nun, eine gute Methode zu finden, dem Computer zu sagen, an welchen Stellen er welche Figuren setzen soll. Nun, verwenden wir doch die uns schon bekannte Art, über ein Koordinatensystem dessen Einteilung auf der X-Achse von 0 bis 319, auf der Y-Achse von 0 bis 199 reicht. Jedoch, einfach ist eine solche Umrechnung nicht - aber zu schaffen. Wir gehen dabei wie folgt vor:

Zunächst einmal ermitteln wir die Zeilenadresse, also die Adresse der Zeile (0-25), in dem unserer Punkt liegt. Das können wir dadurch bewerkstelligen, daß wir die Y-Koordinate durch 8 teilen, denn wir haben ja schon in den Schaubildern gesehen, daß eine Zeile jeweils aus einer Aneinanderreihung von 8er-Päckchen besteht. Wir schreiben also:

```
zeile=int(y/8)
```

Der Vorkommateil des Ergebnisses gibt nun die Zeile an, während über den Nachkommateil (wie, sehen wir später) die Reihe innerhalb einer Zeile bestimmt werden kann.
Da jede Zeile nun aus 40 nebeneinanderliegenden Achter-Päckchen besteht, müssen wir das Ergebnis des oben stehenden Ausdrucks noch einmal 40*8 (= mal 320) nehmen:

```
zeile=int(y/8)*320
```

Der Divisionsrest, das entspräche den unteren drei Bits des Y-Wertes, gibt nun die Reihe innerhalb der Zeile an und muß dann natürlich auch noch zu dem oberen Ausdruck hinzuaddiert werden:

```
zeile=int(y/8)*320+(y and 7)
```

Nun müssen wir nur noch herausfinden, welches Byte wir in der schon ermittelten Zeile ansprechen müssen, und anschließend noch die Bitnummer in diesem Byte errechnen:

```
bytenr=int(x/8)*8

bitnummer=7-(x and 7)
```

Um also einen Punkt mit unserem Verfahren an die Stelle 130 und 70 zu setzen, müssen wir folgendes Programm ausführen. Zur Kontrolle wird der von gesetzte Punkt von einem BASIC-Befehl, dem Draw-Befehl wieder gelöscht:

```
100 rem ********************************
110 rem ***        punkt-demo          ***
120 rem ***      ------------          ***
130 rem ********************************
140 :
150 graphic 1,1           :rem grafik einschalten & loeschen
160 x=130:y=70            :rem x und y Koordinaten
170 adr=8192+int(y/8)*320+(y and 7)+(int(x/8)*8)
180 bit=7-(x and 7)
190 a=peek(adr)           :rem wert auslesen
200 poke adr,a or bit     :rem mit neuem punkt verknuepfen
210 :
220 getkey a$             :rem auf taste warten
230 draw 0,x,y            :rem punkt loeschen
```

3. Anwendungen der Grafikprogrammierung

3.1 Computer Aided Design

3.1.1 "Mini-Painter" für C16, C116 und Plus/4

Als erstes wollen wir in die wohl einfachste Form von CAD - Programmen Einblick nehmen, in die Layout-Programme. Dies sind Programme, welche es erlauben, einfache kleine Zeichnungen auf dem Bildschirm zu entwerfen und anschließend eine Hardcopy davon anzufertigen. Anwendungsbeispiele für diese Programme gibt es massenhaft - um ein solches Beispiel zu finden, brauchen wir gar nicht weit zu schauen: Ihr Grafikbuch selbst ist das beste Beispiel dafür, denn sämtliche Abbildungen, die Sie in diesem Buch finden, sind mit einem solchem Programm hergestellt worden. Um Ihnen auch die Möglichkeit zu geben, in den Genuß eines solchen Programms zu kommen, haben wir nun im folgenden ein solches Programm für den C16, C116 und Plus/4 abgedruckt. Zugegeben, dieses Programm erreicht lange nicht den Komfort, den Sie vielleicht von den großen 16- und 32-Bit-Rechnern kennen, aber vielleicht haben Sie selbst einmal Lust, ein solches, viel besseres Programm für den C16 zu programmieren?

Doch kommen wir nun zu dem Listing unseres Layout-Programms mit der daran anschließenden Erklärung der einzelnen Befehle:

```
100 rem ------------------------------
110 rem --*                         *--
120 rem -**      mini-painter       **-
130 rem -**    c16, c116, plus/4    **-
140 rem -** --------------------    **-
150 rem -** (min. 32k)              **-
160 rem -**                         **-
170 rem -** (kl/07/86)              **-
180 rem --*                         *--
```

```
190 rem ------------------------------
200 :
210 data "  draw  "," line  "," circle "," disk  "," rec   "," box   "
220 data " paint ","floppy "
230 :
240 color 0,1:color 4,1:color 1,5,3
250 scnclr
260 print ">>>  mini-painter c-16/c-116 plus 4 <<<"
270 print ">>> written '86 by klaus loeffelmann <<<"
280 print "-------------------------------------"
290 graphic 1,1:graphic 0:s=3
300 for x=1 to 8:read t$(x):char ,15,4+x*2,t$(x):next
310 :
320 x=1:char ,15,6,chr$(18)+t$(x)+chr$(18+128)
330 :
340 :
350 cr$=chr$(29) : rem cursor nach rechts
360 cl$=chr$(157) : rem cursor nach links
370 cd$=chr$(17) : rem cursor nach unten
380 cu$=chr$(145) : rem cursor nach oben
390 :
400 getkey a$
410 if a$<>cd$ and a$<>cu$ and a$<>chr$(13) and a$<>chr$(27) then
400
420 if a$=chr$(27) then gosub 540:f=1
430 if f=1 and m=1 then run
440 if f=1 and m=0 then f=0:goto 400
450 char ,15,4+x*2,t$(x)
460  if a$=chr$(13) then on x goto 630,1150,1320,1400,1500,1590,
1700,1750
470 if a$=cd$ then x=x+1
480 if a$=cu$ then x=x-1
490 if x<1 then x=8
500 if x>8 then x=1
510 char ,15,4+x*2,chr$(18)+t$(x)+chr$(18+128)
520 goto 400
530 :
540 rem --- sicherheitsabfrage
550 :
```

```
560 char ,5,24,"sind sie sicher ?  "+chr$(18)+"j"+chr$(18+128)+" ,
n":m=1
570 getkey a$
580 if a$=chr$(13) then char ,5,24,"                              ":
return
590 if a$=cr$ then char ,24,24,"j , "+chr$(18)+"n"+chr$(18+128):
m=0
600 if a$=cl$ then char ,24,24,chr$(18)+"j"+chr$(18+128)+" , n":
m=1
610 goto 570
620 :
630 rem --- draw
640 :
650 graphic 1
660 x=160:y=100:locate x,y
670 xa=x:ya=y:locate x,y
680 if rdot(2)=1 then ps=1:draw 0,x,y:else ps=0:draw 1,x,y
690 get a$:if a$=chr$(27) then draw ps,xa,ya:goto 1240
700 if a$=cr$ then j=3
710 if a$=cl$ then j=7
720 if a$=" " then j=0:ff=(ff or1)andnot(ffand1)
730 if a$=cu$ then j=1
740 if a$=cd$ then j=5
750 if a$<>"" then 770
760 j=joy(2):if j>127 then j=j-128:ff=(ff or1)andnot(ffand1)
770 if j=0 and ff<>1 then 690
780 if j=1 then y=y-s
790 if j=2 then y=y-s:x=x+s
800 if j=3 then x=x+s
810 if j=4 then y=y+s:x=x+s
820 if j=5 then y=y+s
830 if j=6 then y=y+s:x=x-s
840 if j=7 then x=x-s
850 if j=8 then x=x-s:y=y-s
860 draw ps,xa,ya
870 if ff=1 then draw 1,x,y to xa,ya
880 goto 670
890 :
900 rem --- get dotposition
910 :
```

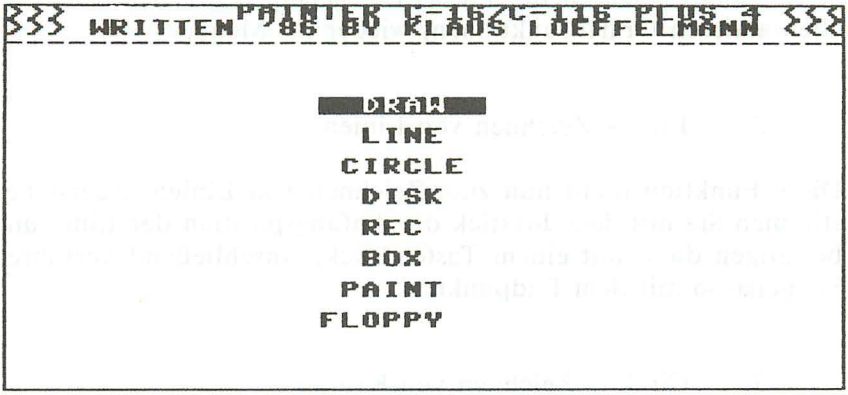
```
920 xa=x:ya=y:locate x,y
930 if rdot(2)=1 then ps=1:draw 0,x,y:else ps=0:draw 1,x,y
940 get a$
950 if a$=chr$(27) then draw ps,xa,ya:return
960 if a$=cr$ then j=3
970 if a$=cl$ then j=7
980 if a$=" " then draw ps,xa,ya:return
990 if a$=cu$ then j=1
1000 if a$=cd$ then j=5
1010 if a$<>"" then 1030
1020 j=joy(2):if j=128 then draw ps,xa,ya:return
1030 if j=0 then 940
1040 if j=1 then y=y-s
1050 if j=2 then y=y-s:x=x+s
1060 if j=3 then x=x+s
1070 if j=4 then y=y+s:x=x+s
1080 if j=5 then y=y+s
1090 if j=6 then y=y+s:x=x-s
1100 if j=7 then x=x-s
1110 if j=8 then x=x-s:y=y-s
1120 draw ps,xa,ya
1130 goto 920
1140 :
1150 rem --- linie
1160 :
1170 graphic 1:x=160:y=100
1180 gosub 920:if a$=chr$(27) then 1240
1190 draw ,x,y:xz=x:yz=y
1200 gosub 920:if a$=chr$(27) then 1240
1210 draw ,xz,yz to x,y
1220 goto 1180
1230 :
1240 rem --- nach einem ruecksprung
1250 :
1260 graphic 0:x=1
1270 char ,15,6,chr$(18)+t$(x)+chr$(18+128)
1280 goto 400
1290 :
1300 rem --- kreis
1310 :
```

```
1320 graphic 1:x=160:y=100
1330 gosub 920:if a$=chr$(27) then 1240
1340 draw ,x,y:xz=x:yz=y
1350 gosub 920:if a$=chr$(27) then 1240
1360 r=xz-x:if r<0 then r=-r
1370 circle ,xz,yz,r:draw 0,xz,yz
1380 goto 1330
1390 :
1400 rem --- ausgefuellter kreis (disk)
1410 :
1420 graphic 1:x=160:y=100
1430 gosub 920:if a$=chr$(27) then 1240
1440 draw ,x,y:xz=x:yz=y
1450 gosub 920:if a$=chr$(27) then 1240
1460 r=xz-x:if r<0 then r=-r
1470 circle ,xz,yz,r:draw 0,xz,yz:paint ,xz,yz
1480 goto 1430
1490 :
1500 rem --- rechteck
1510 :
1520 graphic 1:x=160:y=100
1530 gosub 920:if a$=chr$(27) then 1240
1540 draw ,x,y:xz=x:yz=y
1550 gosub 920:if a$=chr$(27) then 1240
1560 box ,xz,yz,x,y
1570 goto 1530
1580 :
1590 rem --- ausgefuelltes rechteck
1600 :
1610 graphic 1:x=160:y=100
1620 gosub 920:if a$=chr$(27) then 1240
1630 draw ,x,y:xz=x:yz=y
1640 gosub 920:if a$=chr$(27) then 1240
1650 box ,xz,yz,x,y,0,1
1660 goto 1620
1670 :
1680 rem --- flaechen ausfuellen
1690 :
1700 graphic 1:x=160:y=100
1710 gosub 920:if a$=chr$(27) then 1240
```

```
1720 paint ,x,y
1730 goto 1710
1740 :
1750 rem --- floppy-operationen
1760 :
1770 scnclr
1780 print "1....laden"
1790 print "2....speichern"
1800 print "3....ende"
1810 :
1820 print:print:print "bitte floppyoperationen nur mit dem"
1830 print "zusatzprogramm aus kapitel 5.1 benutzen!"
1840 :
1850 getkey a$
1860 :
1870 on val(a$) goto 1900,1970,1140
1880 goto 1850
1890 :
1900 rem --- bild laden
1910 :
1920 scnclr
1930 input "name des bildes:";a$
1940 open 1,8,0,a$
1950 sys 1539,1:goto 1240
1960 :
1970 rem --- bild speichern
1980 :
1990 scnclr
2000 input "name des bildes:";a$
2010 open 1,8,1,a$
2020 sys 1536,1:goto 1240
```

Und nun die Erklärung des Programms:

Nach dem Start unseres Mini-Painters erhalten Sie folgenden Menü-Aufbau:



Sie können nun mit den Cursortasten die entsprechende Befehlsauswahl treffen und dann mit einem Return auslösen. Zurück kommen Sie jederzeit mit ESCape. Diese Taste hat übrigens im Menü eine Sonderfunktion. Mit ihr können Sie nämlich nach Druck derselben Ihr gezeichnetes (und hoffentlich schon abgespeichertes) Bild löschen. Quittieren Sie die dann noch folgende Sicherheitsabfrage mit "JA" (ebenfalls mit den Cursortasten auszuwählen), und Ihr schönes Grafikbild wird für immer aus den RAMs Ihres Computers verschwinden.

Kommen wir zu den Befehlserklärungen im einzelnen:

1. Draw - freihandzeichnen

Nach Aufruf dieser Funktion befinden Sie sich im Grafikmodus. Sie können dort (zugegeben: nach einigem Suchen) den Grafikcursor finden, der Ihnen anzeigt, wo Sie sich auf Ihrem Bild befinden. Leider mußten wir den Grafikcursor aus Geschwindigkeitsgründen so klein halten. Nach einem Druck auf die Feuertaste des Joysticks befinden Sie sich im Draw-Modus. Ab sofort hinterläßt der Grafikcursor beim Zeichnen eine Spur. Ein nochmaliger Druck auf die Feuertaste bringt Sie wieder in den

Normalmodus. Durch die ESC-Taste gelangen Sie, wie auch bei allen anderen Grafikfunktionen, wieder ins Menü.

2. Line - Zeichnen von Linien

Diese Funktion dient nun zum Zeichnen von Linien. Zuerst bestimmen Sie mit dem Joystick die Anfangsposition der Linie und bestätigen diese mit einem Tastendruck. Anschließend verfahren Sie genauso mit dem Endpunkt.

3. Circle - Zeichnen von Kreisen

Hiermit haben Sie nun die Möglichkeit, Kreise auf den Bildschirm zu zeichnen. Wählen Sie zuerst, genau so wie bei der Linien-Funktion, den Mittelpunkt Ihres Kreises aus. Anschließend müssen Sie, ebenfalls durch die nun schon bekannte Technik, den Radius Ihres Kreises wählen. Der Kreis wird nun mit den vorgegebenen Parametern gezeichnet.

4. Disk - Zeichnen von gefüllten Kreisen

Diese Funktion entspricht weitgehend der vorherigen. Sie unterscheidet sich nur in der Tatsache, daß sie anschließend den Kreis noch ausfüllt.

5. Rec - Zeichnen von Rechtecken

Um Rechtecke auf den Bildschirm zu zeichnen, können Sie sich dieser Funktion bedienen. Zuerst bestimmen Sie den Eckpunkt der linken oberen, anschließend den Eckpunkt der linken, unteren Ecke.

6. Box - Zeichnen von gefüllten Rechtecken

Diese Funktion verhält sich äquivalent zur vorherigen, nur daß durch diese Funktion ausgefüllte Rechtecke gezeichnet werden.

7. Paint - Ausfüllen beliebiger Flächen

Diese, schließlich letzte, Grafikfunktion ermöglicht das Ausfüllen beliebiger Flächen. Positionieren Sie dazu den Grafikcursor irgendwo im Inneren der Fläche und betätigen anschließend den Feuerknopf. Die Fläche (die wirklich lückenlos umschlossen sein muß) wird nun ausgefüllt.

8. Floppy - Diskoperationen

Um Ihre Grafiken später noch weiter verwenden zu können, gelangen Sie durch diesen Menüpunkt in das Disketten-Untermenü. Hier haben Sie nun die Möglichkeit, Disketten zu laden und abzuspeichern. Achtung! Grundlagen hierfür sind jedoch unsere Routinen zum Speichern und Laden der Grafik aus dem I/O-Kapitel.

3.1.2 Konstruktionsprogramme

Bestimmt standen auch Sie schon einmal vor dem Problem, irgendeine technische Zeichnung anzufertigen. Sie wollten nun Ihren Computer mit seiner herrlichen Grafik für die Aufgabe einsetzen. Doch schon bald haben Sie bestimmt gemerkt, daß dem Commodore 16 irgendwie bestimmte Voraussetzungen fehlen, um diese Aufgabe zufriedenstellend zu lösen.

An dieser Stelle möchten wir nun an Sie appellieren, sich einmal mit dem "Computer Aided Design" (computerunterstütztes Konstruieren) zu beschäftigen.

Die Hauptaufgabe eines solchen CAD-Systems besteht darin, ein bestimmtes Teil mit Hilfe des Computers auf dem Bildschirm zu

konstruieren und später dann auf einem Drucker oder Plotter ausgeben zu lassen. Dabei arbeitet der Computer natürlich um ein Vielfaches schneller, als es ein Mensch jemals schaffen könnte.

3.1.2.1 CADOMAT Plus/4

Stellen wir uns also der Aufgabe, ebenfalls für unser Computersystem ein solches Wunderwerk des Programmierens zu schaffen!

In unserem Falle ist das eigentliche Problem die Kommunikation mit dem Rechner. Wir müssen dem Rechner natürlich auf irgendeine Weise mitteilen, an welcher Stelle wir welches Teil positioniert haben möchten. Da bestehen für uns, die wir ja bestimmt nicht Unsummen für irgendwelche komplizierten Eingabegeräte ausgeben möchten, leider nicht so viele Möglichkeiten.

Die erste Möglichkeit wäre der Lightpen, der jedoch die Nachteile hat, daß er 1. nicht sehr verbreitet ist und 2. zu ungenau arbeitet.

Zweite Möglichkeit wäre ein Joystick, so, wie wir es bei unserem Mini-Painter kennengelernt haben, mit dem wir jedoch auf die Dauer nicht sehr rentabel arbeiten können, weil ein solcher Joystick viel zu langsam ist.

Die dritte Möglichkeit wäre ein sogenanntes Grafiktablett, das für zweidimensionale Zeichnungen bestimmt am allerbesten geeignet ist. Kommen wir jedoch einmal zu dreidimensionalen Funktionen, so fehlt uns dann die Möglichkeit, die Raumkkoordinaten einzelner Objekte einzuzeichnen.

Bliebe uns als letzte Möglichkeit nur noch die Tastatur, die zwar den Nachteil hat, daß es mitunter eine ganze Zeit braucht, bis man eine geometrische Grundfigur im Bildschirm positioniert hat, jedoch hat sie den Vorteil, daß jedermann über sie verfügt und man mit ihr auch Raumkoordinaten und andere Parameter, wie zum Beispiel den Drehwinkel eines Gebildes, angeben kann.

3.1.2.2 Einteilung in verschiedene Ebenen

Das nächste, eigentlich nicht grafisch orientierte Problem ist die Einteilung in verschiedene Ebenen. Dazu muß zunächst einmal der Begriff "Ebene" geklärt werden. In bezug auf CAD ist eine Ebene nichts anderes als ein Teil der Gesamtfigur, die wir erstellen wollen. Sie können sich den Begriff "Ebene" am besten vorstellen, wenn Sie ihn mit einer Klarsichtfolie vergleichen, auf der nur ein einziges Stück der Gesamtfigur zu sehen ist. Legt man nun mehrere solcher Einzelfiguren übereinander, so ergibt sich aus diesen die Gesamtfigur.

Große CAD-Systeme haben nun den Vorteil, über sehr viel mehr Speicherplatz zu verfügen, als wir ihn zur Verfügung haben. Sie können einen einzigen Teil, ähnlich wie unsere Shapes, die jeweils einzeln berechnet werden, zwischenspeichern, und brauchen bei Bedarf diese nur noch in das Gesamtbild zu kopieren.

Wir haben, wie schon gesagt, wegen akutem Speicherplatzmangel diese Möglichkeit nicht und müssen uns deswegen etwas anderes einfallen lassen.

Eine Möglichkeit wäre, bei Bedarf jeweils die gesamte Grafik neu zeichnen zu lassen. Damit wäre unser CAD-System natürlich nicht sonderlich schnell, aber wir hätten auf der anderen Seite eine fast beliebige Menge Platz für unsere einzelnen Ebenen.

Wenn wir nun einen "Grafikablauf", also eine Zeichnung, anfertigen lassen, so machen wir im Prinzip nichts anderes, als eine Folge von untergeordneten Zeichnungen zu bestimmen. Auch hierzu wieder ein Vergleich, es ist dies das normale BASIC, oder besser, der BASIC-Interpreter. Dieser macht auch nichts anderes, als aufgrund mehrerer Teilinformationen das eigentliche große Programm auszuführen.

Wir müssen natürlich jetzt auch die Möglichkeit haben, Teile aus dem Gesamt-Zeichenablauf herauszunehmen.

Nun würde sich am besten ein Array als Programmspeicher für unser CAD-System eignen, weil dieser vom BASIC her am einfachsten zu bearbeiten ist.

In diesem Array ist auch das Einfügen oder Löschen von Zeichenschritten nicht mehr sonderlich schwierig.
Wir haben für Sie nun ein kleines CAD-System erarbeitet, dessen Funktionsweise wir Ihnen nun nach und nach vorstellen möchten:

3.1.2.3 Das Menü

Zunächst geht es also um das Erstellen eines Gesamt-Menüs, mit dem wir später alle Unterfunktionen aufrufen können:

```
100 rem ********************************
110 rem ***                            ***
120 rem ***          caddimat          ***
130 rem ***          ---------         ***
140 rem ***        (min. 32 K)         ***
150 rem ********************************
160 :
170 dim cm(60,10),p(8) : rem befehlsarrays vorbereiten
180 :
190 rem *** anzahl der parameter festlegen ***
200 :
210 p(1)=2 : p(2)=4 : p(3)=8
215 print chr$(27)+"m";:rem scrollen verhindern
220 p(4)=6 : p(5)=2 : p(6)=4
230 p(7)=6 : p(8)=2
240 :
270 color 0,1          : rem hintergrundfarbe bestimmen
280 color 1,6          : rem zeichen farbe bestimmen
290 color 4,1          : rem rahmenfarbe bestimen
300 :
310 graphic 1,1        : rem grafik loeschen
320 :
```

```
330 rem ********************************
340 rem *** erstellen der zeichnung ****
350 rem ********************************
360 :
380 graphic 0
390 print chr$(19)+chr$(19)+chr$(147)+chr$(14)
410 char ,15,0,"M e n u e"+chr$(13)
420 for x=1 to 40:print "-";:next
430 print:print:print:print "zwei-dimensionale Operationen"
440 print "----------------------------":print
450 print "Punkt.............1    Linie.............2"
460 print "Kreis.............3    Recheck..........4"
470 print "Rastereinteilung.5    Felder loeschen..6"
480 print "Felder kopieren..7    Flaechen fuellen.8"
490 :
500 for x=1 to 40:print chr$(183);:next:print
510 :
520 zs=1
530 :
540 data "Editieren",1
550 data "Grafik"   ,13
560 data "Diskette" ,21
570 data "Beenden"  ,31
580 :
590 for x=1 to 4           : rem menuepunkte einlesen
600 :read mn$(x),pm(x)     : rem menuepunkt & position
610 :char ,pm(x),23,mn$(x) : rem auf bildschirm ausgeben
620 next x
630 :
640 char ,1,23,chr$(18)+mn$(1)+chr$(146)
650 :
660 rem ********************************
670 rem *** menuepunkt auswaehlen ******
680 rem ********************************
690 :
710 restore:z=1:za=1
720 :
730 getkey a$
740 if a$=chr$( 29) then z=z+1:if z>4 then z=1
750 if a$=chr$(157) then z=z-1:if z<1 then z=4
```

```
760 if a$=chr$( 13) then 800
770 char ,pm(za),23,chr$(146)+mn$(za)
780 char ,pm( z),23,chr$( 18)+mn$( z)
790 za=z : goto 730
800 print chr$(146);
810 on z goto 810,810,810,810
820 scnclr
830 end
```

Wenn Sie dieses Programm ablaufen lassen, so werden Sie sehen, daß dieses Programm nur eine Auswahl an CAD-Operationen zur Verfügung stellt. Weiterhin haben Sie die Möglichkeit, mit den Cursortasten einen bestimmten Punkt dieses CAD-Systems anzuwählen und diesen dann durch Bestätigung durch die RETURN-Taste auszuführen.

Sollten Sie zu diesem Zeitpunkt schon einmal die Return-Taste betätigen, so wird sich der Computer noch "aufhängen", da ja noch keinerlei Programmteile zur Verwaltung dieser entsprechenden Punkte vorhanden sind. Dies geschieht in Zeile 10, wo später einmal die Zeilennummern zu den weiteren Menüpunkten stehen müssen.

Das erste Untermenü, das wir nun benötigen, wäre ein Eingabeprogramm, mit dem Sie, wie oben besprochen, die Möglichkeit haben, Ihre Zeichenschritte einzugeben. Es müßten weiterhin grundlegende Funktionen implementiert sein, die ein sinnvolles und relativ komfortables Editieren unseres späteren Zeichenablaufes ermöglichen.

Dieses Programm könnte nun, passend zu dem ersten Teil natürlich, wie folgt aussehen:

3.1.2.4 Das Dialogprogramm

```
840 :
850 rem ********************************
860 rem *** editieren ******************
870 rem ********************************
880 :
890 gosub 1550 : rem statuszeilen loeschen
900 char ,0,23,"1) input  2) delete 3) insert  4) list"+chr$(13)
910 input "Ihre Wahl (5)=ende ";mn
920 if mn<1 or mn>5 then 900 : rem bereichsueberschreitung
930 gosub 1550 : rem statuszeilen loeschen
940 on mn goto 990,1240,1060,1630,540
950 :
960 rem *** zeichenschritt eingeben ***
970 :
980 gosub 1550 : rem statuszeilen loeschen
990 char ,0,23:print "Zeichenschrittnr.:  ";:input z
1000 gosub  1380 : rem zeichenart & parameter eingeben
1010 char ,0,23,"Weitere Zeichenschritte eingeben (j/n) ?    "
1020 char ,0,24,"                                ": rem 30 * space
1030 getkey a$:if a$="j" then z=z+1:goto 980
1040 z=z+1:goto 850 : rem zum editier menue
1050 :
1060 rem *** zeichenschritt einfuegen ***
1070 :
1080 gosub 1550 : rem statuszeilen loeschen
1090 char ,0,23,"Vor welchem zeichenschritt ?"+chr$(13)
1100 input z2:gosub 1560 : rem statuszeilen loeschen
1110 if z2>59 then char ,0,23,"Kein Platz vorhanden     ":getkey
a$:goto 1090
1120 for x=z2+1 to 59
1130 :for x2=1 to 10
1140 ::cm(x+1,x2)=cm(x,x2)
1150 next x2,x:za=zs:zs=z2
1160 char ,0,23:print chr$(18)+"Zeichenschrittnr.:";zs;chr$(146)
1170 gosub 1380   : rem zeichenart & parameter eingeben
1180 zs=za
1190 char ,0,23,"Weitere Zeichenschritte einfuegen (j/n) ?"
```

```
1200 char ,0,24,"                    ": rem 20 * space
1210 getkey a$:if a$="j" then 1090
1220 goto 850
1230 :
1240 rem *** zeichenschritt loeschen ***
1250 :
1260 gosub 1550 : rem statuszeilen loeschen
1270 char ,0,23,"Welchen Zeichenschritt ?"+chr$(13)
1280 input z2
1290 for x=60 to z2+1 step -1
1300 :for x2=1 to 10
1310 ::cm(x,x2)=cm(x-1,x2)
1320 next x2,x
1330 char ,0,23,"Weitere Zeichenschritte loeschen (j/n) ?"
1340 char ,0,24,"                    ": rem 20 * space
1350 getkey a$:if a$="j" then 1270
1360 goto 850
1370 :
1380 rem ********************************
1390 rem *** schrittart eingeben ********
1400 rem ********************************
1410 :
1430 char ,0,23,"Bitte Zeichenart eingeben"+chr$(13)
1440 input "Zeichenart ";za:cm(z,1)=za
1450 gosub 1550 : rem statuszeilen loeschen
1460 char,0,23,"Diese Zeichenart benoetigt";:print p(za);"Paramete
r"
1470 for vz=1 to 500:next vz
1480 gosub 1550 : rem statuszeilen loeschen
1490 for x=2 to p(za)+1
1500 :char ,0,23,"Parameternummer :"+str$(x-1)+"      "+chr$(13)
1510 :input cm(z,x)
1520 next x
1530 return
1540 :
1550 rem ********************************
1560 rem *** zeilen 23 und 24 loeschen **
1570 rem ********************************
1580 :
```

```
1590 char ,0,23,"                                        " : rem
40 space
1600 char ,0,24,"                                        " : rem
40 space
1610 return
1620 :
1630 rem ********************************
1640 rem *** programm listen ************
1650 rem ********************************
1660 :
1670 scnclr:print chr$(14):print chr$(27)+"l"
1680 print "auf Drucker oder Bildschirm ? {d/b}"
1690 getkey a$
1700 if a$="d" then open1,4,1:else open1,3
1710 for x=1 to 60
1720 :x$=str$(x)
1730 :print#1,chr$(12) : rem seitenwechsel ausfuehren
1740 :print#1,right$(x$,len(x$)-1)+"      ";:rem 7 * space
1750 :for x2=1 to 9
1760 ::print#1,right$(str$(cm(x,x2)),len(str$(cm(x,x2)))-1);",";
1770 :next x2
1780 :print#1,right$(str$(cm(x,x2)),len(str$(cm(x,x2)))-1)
1790 next x
1800 print:print"Bitte Taste druecken"
1810 getkey a$
1820 close 1
1830 print chr$(27)+"m";
1840 goto 380
1850 :
```

Sie sollten nun natürlich, bevor Sie sich an dieses Programm heranwagen, die erste Zeilennummer in der Zeile 810 in die Zeile 850 umändern.

Mit diesem zweiten Programmteil haben Sie nun die Möglichkeit, Ihren CAD-Ablauf zu bearbeiten. Dieses Editierprogramm strotzt zwar nicht gerade vor Eingabefreundlichkeit und Schnelligkeit, doch gerade es zu perfektionieren, sollten Sie sich zum Ziel setzen!

Die grundlegenden Dinge jedoch, wie Editieren, Schritt einfügen, Schritt löschen und Ablauf listen haben wir in diesen Programmteil integriert.

Doch nun weiter im Text. Es fehlen jetzt noch andere grundlegende Dinge, die das zukümftige Arbeiten mit unserem CAD-System ermöglichen.

Natürlich müssen die Zeichnungen und Grafiken, die wir erstellt haben, abgspeichert werden

Die einfachste Lösung hierfür wäre, die Grafik mit unseren Routinen aus dem 5. Kapitel als Ganzes abzuspeichern. Doch können wir, wenn wir dieses Verfahren benutzen, unsere Grafik nach einem erneuten Laden nicht mehr oder nur noch beschränkt bearbeiten. Aber eine andere Lösung bietet sich an. Da wir mit Arrays arbeiten, sollte es kein großes Problem darstellen, diese auf Diskette zu speichern oder von einer solche zu laden. Diese Lösung bieten wir Ihnen als weiteres Teilprogramm nun im folgenden an:

3.1.2.5 Die I/O-Operationen

```
1860 rem ********************************
1870 rem *** diskettenoperationen *******
1880 rem ********************************
1890 :
1900 gosub 1590 : rem statuszeilen loeschen
1910 char ,0,23,"1) Programm laden  2) Programm speichern"+chr$(13
)
1920 input "Ihre Wahl (3=Ende)";a:gosub1590 : rem statuszeilen loe
schen
1930 on a goto 2170,1990,530
1940 :
1950 rem ********************************
1960 rem *** programm speichern *********
1970 rem ********************************
```

```
1980 :
1990 gosub 1590 : rem statuszeilen loeschen
2000 char ,0,23,"Bitte Programmnamen eingeben"+chr$(13)
2010 input dn$
2020 open1,8,1,dn$
2030 for x=1 to 60
2040 :for z=1 to 10
2050 ::print#1,cm(x,z)
2060 next z,x
2070 close 1
2080 gosub 1590
2090 char ,0,23,ds$
2100 char ,0,24,"Bitte taste druecken"
2110 getkey a$:goto 1900
2120 :
2130 rem ********************************
2140 rem *** programm laden *************
2150 rem ********************************
2160 :
2170 gosub 1590 : rem statuszeilen loeschen
2180 char ,0,23,"Bitte Programmnamen eingeben"+chr$(13)
2190 input dn$
2200 open1,8,0,dn$
2210 for x=1 to 60
2220 :for z=1 to 10
2230 ::input#1,cm(x,z)
2240 next z,x
2250 close 1
2260 gosub 1590
2270 char ,0,23,ds$
2280 char ,0,24,"Bitte taste druecken"
2290 getkey a$:goto 1900
```

Wieder müssen Sie, bevor Sie sich dieses Programmteils bemächtigen, die entsprechende Zeilennummer in der Zeile 810 eintragen. Strikt nach Reihenfolge vorgegangen - der aufmerksame Leser wird dies schon erkannt haben - muß diesmal die dritte Zeilennummer in der Zeile 810 geändert werden.

Eine sinnvolle Erweiterung wären zusätzliche Ein- und Ausgaberoutinen für Drucker oder Plotter. Einige davon haben wir ja im 5. Kapitel abgedruckt, vielleicht haben Sie Lust, eine solche zu integrieren?

Nachdem nun alle "nicht-grafikorientierten" Probleme aus dem Weg geräumt wurden, lassen Sie uns den grafischen Problemen zuwenden.

Sie haben bestimmt schon im Eingabeprogramm erkannt, daß in unserem Befehlsarray zunächst die Nummer des Befehls gespeichert wurde und im Anschluß daran die entsprechenden Parameter folgten. Wir können nun unsere Gesamtfigur zeichnen, indem wir das gesamte Array durchgehen und aufgrund der Nummern der Grafikoperationen die einzelnen Grafikunterroutinen anspringen, in denen wiederum die einzelnen Befehlscodes übergeben werden.

Hier nun der letzte Teil des CAD-Systems:

3.1.2.6 Die Grafikroutinen

```
2300 :
2310 rem ********************************
2320 rem *** grafikoperationen **********
2330 rem ********************************
2340 :
2350 gosub 1590 : rem statuszeilen loeschen
2360 char ,0,23,"Grafik  1) clear  2) draw"+chr$(13)
2370 input "Ihre Wahl (3=Ende)";a
2380 on a goto 2450,2660,2390
2390 gosub 1550:goto 540
2400 :
2410 rem ********************************
2420 rem *** grafik loeschen ************
2430 rem ********************************
2440 :
2450 gosub 1590 : rem statuszeilen loeschen
```

```
2460 char ,0,23,"Grafik loeschen ?"+chr$(13)
2470 input a$:if a$="ja" then gl=1:else gl=0
2480 goto 2350
2490 :
2620 rem ********************************
2630 rem *** programm ablaufen lassen ***
2640 rem ********************************
2650 :
2660 graphic 1,gl
2680 for x=1 to 60
2690 :s=cm(x,1):if s=0 then 2710
2700 :on s gosub 2850,2920,2990,3060,3130,3240,3310,3380
2710 next x
2720 :
2730 getkey a$
2740 graphic 0
2750 goto 2350
2760 :
2790 return
2800 :
2810 rem ********************************
2820 rem *** punkt zeichnen *************
2830 rem ********************************
2840 :
2850 draw 1,cm(x,2),cm(x,3)
2860 return
2870 :
2880 rem ********************************
2890 rem *** linie zeichnen *************
2900 rem ********************************
2910 :
2920 draw 1,cm(x,2),cm(x,3) to cm(x,4),cm(x,5)
2930 return
2940 :
2950 rem ********************************
2960 rem *** kreis zeichnen *************
2970 rem ********************************
2980 :
2990 circle 1,cm(x,2),cm(x,3),cm(x,4),cm(x,5),cm(x,6),cm(x,7),
cm(x,8)
```

```
3000 return
3010 :
3020 rem *********************************
3030 rem ***  rechteck zeichnen *********
3040 rem *********************************
3050 :
3060 box 1,cm(x,2),cm(x,3),cm(x,4),cm(x,5),cm(x,6),cm(x,7)
3070 return
3080 :
3090 rem *********************************
3100 rem *** raster einteilung **********
3110 rem *********************************
3120 :
3130 z=0
3140 for xk=1 to xmax step cm(x,2)
3150 for yk=1 to ymax step cm(x,3)
3160 draw 1,xk,yk
3170 next yk,xk
3180 return
3190 :
3200 rem *********************************
3210 rem *** felder loeschen ************
3220 rem *********************************
3230 :
3240 box 0,cm(x,2),cm(x,3),cm(x,4),cm(x,5),0,1
3250 return
3260 :
3270 rem *********************************
3280 rem *** felder kopieren ************
3290 rem *********************************
3300 :
3310 sshape a$,cm(x,2),cm(x,3),cm(x,4),cm(x,5)
3320 gshape a$,cm(x,6),cm(x,7)
3330 return
3340 :
3350 rem *********************************
3360 rem *** flaechen ausfuellen ********
3370 rem *********************************
3380 :
```

```
3390 paint ,cm(x,2),cm(x,3)
3400 return
```

Wie Sie in diesem Listing selbst sehen, haben wir uns nur der Grafikbefehle bedient, die das Commodore-BASIC von vornherein bietet. Selbstverständlich können Sie dieses CAD-System ganz nach Belieben erweitern. Ihrer Phantasie sind hier keine Grenzen gesetzt.

Zu guter Letzt wollen wir Ihnen das CAD-Programm noch in seiner gesamten Länge auflisten, da durch die Änderungen, die beim Zusammenfügen der einzelnen Teile notwendig waren, Flüchtigkeitsfehler sehr schnell gemacht sind.

3.1.2.7 Das gesamte CAD-System

```
100 rem ********************************
110 rem ***                            ***
120 rem ***       caddimat plus 4      ***
130 rem ***       --------------       ***
140 rem ***(min. 32k)                  ***
150 rem ********************************
160 :
170 dim cm(60,10),p(8) : rem befehlsarrays vorbereiten
180 :
190 rem *** anzahl der parameter festlegen ***
200 :
210 p(1)=2 : p(2)=4 : p(3)=8
215 print chr$(27)+"m";:rem scrollen verhindern
220 p(4)=6 : p(5)=2 : p(6)=4
230 p(7)=6 : p(8)=2
240 :
270 color 0,1          : rem hintergrundfarbe bestimmen
280 color 1,6          : rem zeichen farbe bestimmen
290 color 4,1          : rem rahmenfarbe bestimen
300 :
310 graphic 1,1        : rem grafik loeschen
```

```
320 :
330 rem ********************************
340 rem *** erstellen der zeichnung ****
350 rem ********************************
360 :
380 graphic 0
390 print chr$(19)+chr$(19)+chr$(147)+chr$(14)
410 char ,15,0,"M e n u e"+chr$(13)
420 for x=1 to 40:print "-";:next
430 print:print:print:print "zwei-dimensionale Operationen"
440 print "----------------------------":print
450 print "Punkt.............1    Linie.............2"
460 print "Kreis.............3    Recheck...........4"
470 print "Rastereinteilung.5    Felder loeschen..6"
480 print "Felder kopieren..7    Flaechen fuellen.8"
490 :
500 for x=1 to 40:print chr$(183);:next:print
510 :
520 zs=1
530 :
540 data "Editieren",1
550 data "Grafik"   ,13
560 data "Diskette" ,21
570 data "Beenden"  ,31
580 :
590 for x=1 to 4           : rem menuepunkte einlesen
600 :read mn$(x),pm(x)     : rem menuepunkt & position
610 :char ,pm(x),23,mn$(x) : rem auf bildschirm ausgeben
620 next x
630 :
640 char ,1,23,chr$(18)+mn$(1)+chr$(146)
650 :
660 rem ********************************
670 rem *** menuepunkt auswaehlen ******
680 rem ********************************
690 :
710 restore:z=1:za=1
720 :
730 getkey a$
740 if a$=chr$( 29) then z=z+1:if z>4 then z=1
```

```
750 if a$=chr$(157) then z=z-1:if z<1 then z=4
760 if a$=chr$( 13) then 800
770 char ,pm(za),23,chr$(146)+mn$(za)
780 char ,pm( z),23,chr$( 18)+mn$( z)
790 za=z : goto 730
800 print chr$(146);
810 on z goto 850,2350,1860,820
820 scnclr
830 end
840 :
850 rem ********************************
860 rem *** editieren ******************
870 rem ********************************
880 :
890 gosub 1550 : rem statuszeilen loeschen
900 char ,0,23,"1) input  2) delete 3) insert  4) list"+chr$(13)
910 input "Ihre Wahl (5)=ende ";mn
920 if mn<1 or mn>5 then 900 : rem bereichsueberschreitug
930 gosub 1550 : rem statuszeilen loeschen
940 on mn goto 990,1240,1060,1630,540
950 :
960 rem *** zeichenschritt eingeben ***
970 :
980 gosub 1550 : rem statuszeilen loeschen
990 char ,0,23:print "Zeichenschrittnr.:  ";:input z
1000 gosub  1380 : rem zeichenart & parameter eingeben
1010 char ,0,23,"Weitere Zeichenschritte eingeben (j/n) ?    "
1020 char ,0,24,"                                    ": rem 30 * space
1030 getkey a$:if a$="j" then z=z+1:goto 980
1040 z=z+1:goto 850 : rem zum editier menue
1050 :
1060 rem *** zeichenschritt einfuegen ***
1070 :
1080 gosub 1550 : rem statuszeilen loeschen
1090 char ,0,23,"Vor welchem zeichenschritt ?"+chr$(13)
1100 input z2:gosub 1560 : rem statuszeilen loeschen
1110 if z2>59 then char ,0,23,"Kein Platz vorhanden     ":getkey
a$:goto 1090
1120 for x=z2+1 to 59
1130 :for x2=1 to 10
```

```
1140 ::cm(x+1,x2)=cm(x,x2)
1150 next x2,x:za=zs:zs=z2
1160 char ,0,23:print chr$(18)+"Zeichenschrittnr.:";zs;chr$(146)
1170 gosub 1380   : rem zeichenart & parameter eingeben
1180 zs=za
1190 char ,0,23,"Weitere Zeichenschritte einfuegen (j/n) ?"
1200 char ,0,24,"                    ": rem 20 * space
1210 getkey a$:if a$="j" then 1090
1220 goto 850
1230 :
1240 rem *** zeichenschritt loeschen ***
1250 :
1260 gosub 1550 : rem statuszeilen loeschen
1270 char ,0,23,"Welchen Zeichenschritt ?"+chr$(13)
1280 input z2
1290 for x=60 to z2+1 step -1
1300 :for x2=1 to 10
1310 ::cm(x,x2)=cm(x-1,x2)
1320 next x2,x
1330 char ,0,23,"Weitere Zeichenschritte loeschen (j/n) ?"
1340 char ,0,24,"                    ": rem 20 * space
1350 getkey a$:if a$="j" then 1270
1360 goto 850
1370 :
1380 rem ********************************
1390 rem *** schrittart eingeben ********
1400 rem ********************************
1410 :
1430 char ,0,23,"Bitte Zeichenart eingeben"+chr$(13)
1440 input "Zeichenart ";za:cm(z,1)=za
1450 gosub 1550 : rem statuszeilen loeschen
1460 char ,0,23:print "Diese zeichenart benoetigt";p(za);
"Parameter"
1470 for vz=1 to 500:next vz
1480 gosub 1550 : rem statuszeilen loeschen
1490 for x=2 to p(za)+1
1500 :char ,0,23,"Parameternummer :"+str$(x-1)+"      "+chr$(13)
1510 :input cm(z,x)
1520 next x
1530 return
```

```
1540 :
1550 rem ********************************
1560 rem *** zeilen 23 und 24 loeschen **
1570 rem ********************************
1580 :
1590 char ,0,23,"                                        " :
rem 40 space
1600 char ,0,24,"                                        " :
rem 40 space
1610 return
1620 :
1630 rem ********************************
1640 rem *** programm listen ************
1650 rem ********************************
1660 :
1670 scnclr:print chr$(14):print chr$(27)+"l"
1680 print "auf Drucker oder Bildschirm ? {d/b}"
1690 getkey a$
1700 if a$="d" then open1,4,1:else open1,3
1710 for x=1 to 60
1720 :x$=str$(x)
1730 :print#1,chr$(12) : rem seitenwechsel ausfuehren
1740 :print#1,right$(x$,len(x$)-1)+"       ";:rem 7 * space
1750 :for x2=1 to 9
1760 ::print#1,right$(str$(cm(x,x2)),len(str$(cm(x,x2)))-1);",";
1770 :next x2
1780 :print#1,right$(str$(cm(x,x2)),len(str$(cm(x,x2)))-1)
1790 next x
1800 print:print"Bitte Taste druecken"
1810 getkey a$
1820 close 1
1830 print chr$(27)+"m";
1840 goto 380
1850 :
1860 rem ********************************
1870 rem *** diskettenoperationen *******
1880 rem ********************************
1890 :
1900 gosub 1590 : rem statuszeilen loeschen
1910 char ,0,23,"1) Programm laden  2) Programm speichern"
```

```
+chr$(13)
1920 input "Ihre Wahl (3=Ende)";a:gosub1590 : rem statuszeilen
loeschen
1930 on a goto 2170,1990,530
1940 :
1950 rem ********************************
1960 rem *** programm speichern *********
1970 rem ********************************
1980 :
1990 gosub 1590 : rem statuszeilen loeschen
2000 char ,0,23,"Bitte Programmnamen eingeben"+chr$(13)
2010 input dn$
2020 open1,8,1,dn$
2030 for x=1 to 60
2040 :for z=1 to 10
2050 ::print#1,cm(x,z)
2060 next z,x
2070 close 1
2080 gosub 1590
2090 char ,0,23,ds$
2100 char ,0,24,"Bitte taste druecken"
2110 getkey a$:goto 1900
2120 :
2130 rem ********************************
2140 rem *** programm laden *************
2150 rem ********************************
2160 :
2170 gosub 1590 : rem statuszeilen loeschen
2180 char ,0,23,"Bitte Programmnamen eingeben"+chr$(13)
2190 input dn$
2200 open1,8,0,dn$
2210 for x=1 to 60
2220 :for z=1 to 10
2230 ::input#1,cm(x,z)
2240 next z,x
2250 close 1
2260 gosub 1590
2270 char ,0,23,ds$
2280 char ,0,24,"Bitte taste druecken"
2290 getkey a$:goto 1900
```

```
2300 :
2310 rem ********************************
2320 rem *** grafikoperationen **********
2330 rem ********************************
2340 :
2350 gosub 1590 : rem statuszeilen loeschen
2360 char ,0,23,"Grafik  1) clear  2) draw"+chr$(13)
2370 input "Ihre Wahl (3=Ende)";a
2380 on a goto 2450,2660,2390
2390 gosub 1550:goto 540
2400 :
2410 rem ********************************
2420 rem *** grafik loeschen ************
2430 rem ********************************
2440 :
2450 gosub 1590 : rem statuszeilen loeschen
2460 char ,0,23,"Grafik loeschen ?"+chr$(13)
2470 input a$:if a$="ja" then gl=1:else gl=0
2480 goto 2350
2490 :
2620 rem ********************************
2630 rem *** programm ablaufen lassen ***
2640 rem ********************************
2650 :
2660 graphic 1,gl
2680 for x=1 to 60
2690 :s=cm(x,1):if s=0 then 2710
2700 :on s gosub 2850,2920,2990,3060,3130,3240,3310,3380
2710 next x
2720 :
2730 getkey a$
2740 graphic 0
2750 goto 2350
2760 :
2790 return
2800 :
2810 rem ********************************
2820 rem *** punkt zeichnen *************
2830 rem ********************************
2840 :
```

```
2850 draw 1,cm(x,2),cm(x,3)
2860 return
2870 :
2880 rem ********************************
2890 rem *** linie zeichnen *************
2900 rem ********************************
2910 :
2920 draw 1,cm(x,2),cm(x,3) to cm(x,4),cm(x,5)
2930 return
2940 :
2950 rem ********************************
2960 rem *** kreis zeichnen *************
2970 rem ********************************
2980 :
2990 circle 1,cm(x,2),cm(x,3),cm(x,4),cm(x,5),cm(x,6),cm(x,7),
cm(x,8)
3000 return
3010 :
3020 rem ********************************
3030 rem ***  rechteck zeichnen *********
3040 rem ********************************
3050 :
3060 box 1,cm(x,2),cm(x,3),cm(x,4),cm(x,5),cm(x,6),cm(x,7)
3070 return
3080 :
3090 rem ********************************
3100 rem *** raster einteilung **********
3110 rem ********************************
3120 :
3130 z=0
3140 for xk=1 to xmax step cm(x,2)
3150 for yk=1 to ymax step cm(x,3)
3160 draw 1,xk,yk
3170 next yk,xk
3180 return
3190 :
3200 rem ********************************
3210 rem *** felder loeschen *************
3220 rem ********************************
3230 :
```

```
3240 box 0,cm(x,2),cm(x,3),cm(x,4),cm(x,5),0,1
3250 return
3260 :
3270 rem ********************************
3280 rem *** felder kopieren ************
3290 rem ********************************
3300 :
3310 sshape a$,cm(x,2),cm(x,3),cm(x,4),cm(x,5)
3320 gshape a$,cm(x,6),cm(x,7)
3330 return
3340 :
3350 rem ********************************
3360 rem *** flaechen ausfuellen ********
3370 rem ********************************
3380 :
3390 paint ,cm(x,2),cm(x,3)
3400 return
```

3.2 Einführung in die Statistik

Wir kommen nun zur wohl bekanntesten Form der Grafikanwendung, der Statistik, die Sie sicher aus unzähligen Wahlsendungen oder aus der Schule kennen.

Doch was ist Statistik nun eigentlich?

Ich habe für Sie in einem Lexikon nachgeschaut, hier die Definition:

STATISTIK [die], (frz.-ital.), die Methode zur Untersuchung von Massenerscheinungen. Sie versucht den Umfang, die Gliederung oder Struktur einer Masse, die zeitliche Entwicklung einer oder das Verhältnis mehrerer Massenerscheinungen zueinander zu erkennen. Sie hat ihre mathematischen Grundlagen und Rechtfertigungen im "Gesetz der großen Zahl" (wenn die Zahl der untersuchten Einzelfälle genügend groß ist, werden die zufälligen Abweichungen aufgehoben und die typischen Zahlenverhältnisse kommen zum Vorschein) und in der Wahrscheinlichkeitsrechnung. Die Statistik findet besonders in der Wirt-

schafts- und Bevölkerungs-Statistik Anwendung. Auch in der Physik (z.B. in der Wärmelehre und in der Quantentheorie) bildet die Statistik ein wichtiges Hilfsmittel.

(Aus "Bertelsmann Volkslexikon")

Doch auch schon in der grauen Vorzeit liegt die Kinderwiege der Statistik.

In Griechenland wurden Statistiken von den Siegern der Olympischen Spielen, in Ägypten über die Bewegung der Sterne geführt und auch die Stammbäume der chinesischen Dynastien waren Anfänge der Statistik.

Doch erst der berühmte französische Mathematiker Rene Descartes beschäftigte sich im siebzehnten Jahrhundert systematisch mit der grafischen Statistik. Er war der erste, der Zahlen mit Strichen und Kreisen darstellte

Die Methoden zur grafischen Darstellung wurden in den nachfolgenden drei Jahrhunderten immer mehr vervollkommnet, und mit besseren Formeln und Umrechnungstabellen wurden auch die erstellten Statistiken genauer. Es dauerte zwar immer noch seine Zeit, bis alle Werte scaliert bzw. so angepaßt waren, daß sie dann auf dem Papier dargestellt werden konnten, aber es waren immerhin schon Fortschritte im Vergleich zu den "aus der Hand gamalten" Statistiken der Anfänge. Außerdem forderten Industrielle Revolution und die Besiedelung fremder Kontinente genauere Mittel zur Auswertung des erhaltenen Zahlenmaterials. Eine Schwachstelle im System war jedoch immer noch der Mensch. Er mußte die Werte grafisch darstellen, das führte zu ungenauen Zeichnungen. Und schon beim Umrechnen, Angleichen oder Abschreiben von Werten wurden einige Fehler gemacht. Ich möchte nur kurz die Geschichte des Spinats anreißen, bei der eine Sekretärin bei einer Studie über verschiedene Gemüsearten den Eisengehalt des Spinats falsch abschrieb und seit daher Spinat als sehr gesund gilt.

Doch heute sind solche Fehler seltener, denn die Daten werden einmal in einen Computer getippt oder direkt von Messinstru-

menten an den Computer übertragen, der diese dann in Bruchteilen von Sekunden skaliert und dann auf einen Grafikschirm mit einer Auflösung von 1024*1024 Punkten und einer Farbpalette von 4096 Farben darstellt (was das bedeutet, müßten Sie ja inzwischen gelernt haben!). Und dann sehen wir diese Kuchen - oder Balkengrafiken in einer der nächsten Wahlsendungen.

Nach diesem kleinen Ausflug in die Geschichte der Statistik möchte ich Ihnen einen Statistikprogrammierkurs und danach ein paar fertige Anwendungsprogramme vorstellen.

Zur grafischen Darstellung der verschiedensten Statistiken gibt es je nach Aussagewunsch die unterschiedlichsten Möglichkeiten. Im folgenden sollen die geläufigsten vorgestellt und erläutert werden:

- Kurvendiagramme
- Balkendiagramme
- Kuchendiagramme

3.2.1 Kurvendiagramme

Besitzen wir viele "Meßwerte" in Abhängigkeit eines bestimmten Faktors (z.B. der Umsatz einer Firma in Abhängigkeit vom jeweiligen Monat (d.h. in jeweils verschiedenen Monaten)), so verwenden wir allgemein diese erste Form der Grafikausgabe. Hierbei werden die gleichen Techniken angewandt, wie dies bei der Darstellung von 2-dimensionalen Funktionen der Fall ist (s. Kapitel 3.5). Der Unterschied liegt lediglich in der Herkunft der einzelnen Daten. Bei der Funktionsdarstellung werden sie jeweils und beliebig dicht und genau berechnet, während wir die einzelnen Werte hier direkt aus bestimmten Tabellen holen.

Ein Problem, auf das wir bei allen Statistikprogrammen stoßen werden, ist die Eingrenzung der unterschiedlichen Werte auf das enge Bildschirmfenster. Entweder sind die Werte zu hoch oder sie sind zu niedrig, als daß sie direkt eins zu eins auf dem Bildschirm dargestellt werden könnten. Wir sind also gezwungen, die verschiedenen Werte mit einem gemeinsamen Streckungs-

bzw. Stauchungs-Faktor zu multiplizieren. Diesen Faktor wählt man zweckmäßigerweise so, daß die entstehende Grafik das volle Bildschirmfenster ausfüllt. Das heißt also, die beiden Extrema Minimum und Maximum der Statistikwerte sollten jeweils die oberste bzw. unterste Grenze des Bildschirms erreichen.

Um dieses Ziel zu realisieren, müssen wir zunächst einmal feststellen, welche Werte diese beiden Extrema haben. Im Anschluß daran berechnen wir aus dem zur Verfügung stehenden Platz in der Grafik (wir arbeiten bekanntlich mit einer Auflösung von 319 mal 200 Punkten) und diesen beiden Werten den Streckungs- bzw. Stauchungsfaktor für alle anderen Werte. Weiterhin interessiert die Anzahl der verschiedenen Werte, da auch nach rechts hin alle Kurveneintragungen etc. ins Grafikfenster passen sollen. Aus der Anzahl nun berechnen wir ebenfalls einen Faktor, der uns die Dichte der Werte in x-Richtung angibt.

Haben wir uns um die Berechnung dieser Faktoren gekümmert, so bleibt uns noch die Zeichnung der Achsen und der Skalierungen (neben der eigentlichen Kurve selbstverständlich). Doch schauen Sie sich das ganze einmal in vivo an.

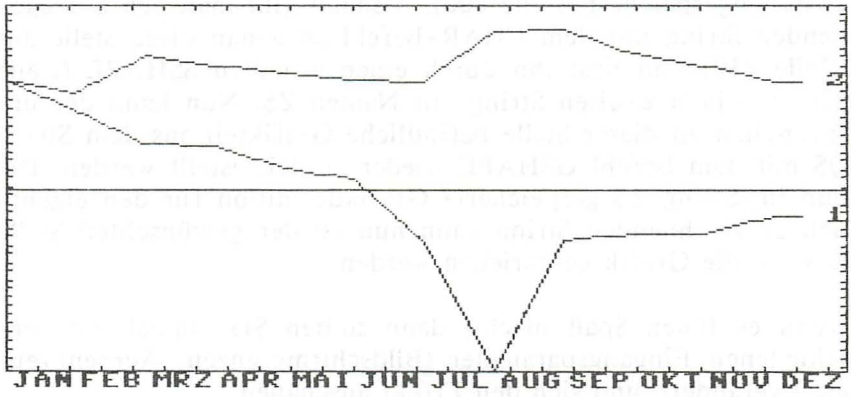
Beachten Sie bitte bei diesem Programm: Diejenigen von Ihnen, die sich mit einem C16 mit 16 Kbyte RAM beschäftigen, dürfen weder REM-Zeilen noch die hier zur besseren Lesbarkeit eingegebenen Leerzeichen zwischen den Befehlen in den Computer eintippen. Wird dieser gut gemeinte Ratschlag nicht beherzigt, so hat man spätestens nach dem ersten Probelauf mit der Fehlermeldung "out of memory error" zu rechnen.

```
100 rem ********************
110 rem **                  **
120 rem **  kurvengrafik    **
130 rem **                  **
140 rem ********************

150 rem werteinitialisierung:
160 xl= 0 : xh=319
```

```
170 yl= 0 : yh=190
180 tt=yh+2

190 rem bildschirm-Init
200 color 0,6 : color 1,1
210 scnclr

220 rem werteeingabe:
230 input "anzahl der kurven";ak
240 input "raster       j/n ";ra$
250 input "monatsangabe j/n ";mo$
260 mo$ = left$(mo$+"n",1)
270 if mo$="j" then aw=12 : goto 360

280 input "anzahl werte     ";aw
290 if aw>(xh-xl)/3 or aw=0 then 320 : rem erlaubt?

300 rem speicher-init:
310 dim we(ak,aw+1)

320 for i=1 to ak                    : rem anzahl kurven
330    for j=1 to aw                 : rem anzahl werte
340       print "reihe:";i;"wert:";j;
350       input we(i,j)              : rem wert eingeben
360       a=abs(we(i,j))             : rem maximum bestimmen
370       if a>hi then hi=a
380    next j
390    we (i,j)=we(i,j-1)            : rem letzter = vorletzter
400 next i

410 yf=1 : a=(yh-yl)/2               : rem y-umrechnungsfaktor
420 if hi>a then yf=a/hi             : rem berechnen

430 rem grafik zeichnen:
440 graphic 1,1

450 for i=1 to ak
460    z=0 : rem index auf null setzen
470    for x=xl to xh step20
```

```
480       z=z+1                          : rem index erhoehen
490       y=yl+(yh-yl)/2                 : rem y-koordinate
500       y=y-we (i,z)*yf                : rem berechnen
510       if z=1 then x0=xl : y0=y       : rem erster wert?
520       draw 1,x0,y0 to x,y            : rem linie zeichnen
530       x0=x : y0=y                    : rem koordinaten speichern
540    next x
550    t$=str$(i) : x=x0-len (t$)*8 : rem nummer in string
560    y=y0-4 : gosub 790             : rem text ausgeben
570 next i

580 rem monate eintragen:
590 if mo$="n" then 850
600 for i=xl to xh step (xh-xl)/aw
610    read t$
620    x=i+2 : y=tt : gosub 790
630 next i

640 rem rahmen zeichnen:
650 box 1,xl,yl,xh,yh
660 draw 1,xl,yl+(yh-yl)/2 to xh,yl+(yh-yl)/2
670 for x=xl to xh step aw
680    draw 1,x,yl to x,yl+2 : rem x-einteilungen
690    draw 1,x,yh to x,yh-2 : rem zeichnen
700 next x

710 for y=yl to yh step 10*yf
720    draw 1,xl,y to xl+2,y : rem y-einteilungen
730    draw 1,xh,y to xh-2,y : rem zeichnen
740 next y

750 data jan,feb,mrz,apr,mai,jun
760 data jul,aug,sep,okt,nov,dez
770 data 0

780 rem text ausgeben:
790 xa=(xl and 248)+8 : ya=(yl and 248)+8
800 sshape q$,xa,ya,xa+len (t$)*8-1,ya+7
810 char 1,xa/8,ya/8,t$
820 sshape z$,xa,ya,xa+len (t$)*8-1,ya+7
```

```
830 gshape q$,xa,ya
840 gshape z$,x,y
850 return
```



Hier kurz eine kleine Beschreibung des obigen Programmes:

Nach der Definition der Bildschirmgrenzwerte, die wir ja benötigen, um die nachfolgend zu zeichnende Grafik auf die richtige Größe zu bringen, folgt in der Zeile 180 die Festlegung der y-Koordinate, an der die eventuellen Monatsnamen ausgegeben werden sollen.

Es folgt die Werteeingabe mit Werteanzahl, Kurvenzahl etc. Diese Werte werden nun in ein vorher dimensioniertes Array geladen und dabei der höchste und der niedrigste Wert identifiziert.

Nun wird in den Zeilen 400 und 410 der Umrechnungsfaktor berechnet, um die Kurve (Werte) in y-Richtung zu strecken oder zu stauchen. Alsbald werden die verschiedenen Kurven, bei Bedarf die Monatsbezeichnungen und das Koordinatensystem, gezeichnet.

Noch ein Wort zur Textausgabe, die hier in einem Unterprogramm ab Zeile 790 realisiert wird: Zunächst wird ein beliebiger Bereich der Grafik mit einer durch 8 teilbaren x- und y-Koordinate mittels SSHAPE-Befehl in einem String (Variable Q$) zwischengespeichert (Zeile 800). Alsdann gibt man den auszugebenden String mit dem CHAR-Befehl an genau diese Stelle aus (Zeile 810) und liest ihn durch einen erneuten SSHAPE (Zeile 820) in einen zweiten String mit Namen Z$. Nun kann der ursprünglich an dieser Stelle befindliche Grafikteil aus dem String Q$ mit dem Befehl GSHAPE wieder zurückgestellt werden. Die nun im String Z$ gespeicherte Grafikdefinition für den eigentlich zu zeichnenden String kann nun an der gewünschten Stelle (x,y) in die Grafik geschrieben werden.

Wenn es Ihnen Spaß macht, dann sollten Sie einmal die verschiedenen Eingangsparameter (Bildschirmgrenzen, Achsenkreuz etc.) verändern und sich den Effekt anschauen.

3.2.2 Balkendiagramme

Bei den Balkendiagrammen, mit denen wir uns im folgenden beschäftigen wollen, wird die Erstellung übersichtlicher Kurven bereits etwas komplizierter, obwohl wir auch hier nach dem gleichen Prinzip vorgehen. Man wendet sie bei relativ kleinen Datenmengen an, die optisch besser sichtbar gemacht werden sollen. Ein Wertepaar dient nun nicht zur Bestimmung der Lage eines Punktes auf dem Bildschirm, sondern der Höhe und Position eines Balkens, der sich von der x-Achse in die Höhe zieht.

Wenn Sie bereits die Einführung zur Kurvengrafik gelesen haben (und das sollten Sie an dieser Stelle nachholen, wenn Sie das bisher versäumt haben), dann dürfte Ihnen die Berechnung eines Streckungs- bzw. Stauchungs-Faktors nicht mehr neu vorkommen, die notwendig ist, um die entstehende Grafik für das zur Verfügung stehende Bildschirmfenster passend zu machen.

Das folgende kleine Programm soll Ihnen die Programmierung solcher Balken demonstrieren. Nehmen wir an, es sollen die

Umsatzzahlen einer Firma über ein Jahr mit Hilfe eines Balkendiagramms dargestellt werden, um dem Leiter eine dringend nötige Übersicht zu vermitteln. Die jeweiligen Zahlen (in Tausend DM) können Sie dabei nach dem Programmstart eigenhändig eingeben. Sie können sie natürlich auch der Einfachheit halber in DATA-Zeilen unterbringen, oder ganz luxuriös von Diskette lesen. Dafür sind dann jeweils nur kleine Änderungen in dem Eingabe-Programmteil notwendig.

Auch hier gilt wieder das bereits oben Gesagte: C16-Besitzer mit 16 KByte RAM dürfen keine REM-Zeilen und keine Leerzeichen zwischen den einzelnen Befehlen eingeben, um nicht unnötigen Speicherplatz zu verschwenden, der bei diesem Rechner dringend vonnöten ist. Andernfalls erscheint nach dem Programmstart ein "out of memory error".

```
100 rem **********************
110 rem **                    **
120 rem **  balkendiagramm    **
130 rem **                    **
140 rem **********************

150 xl=0 : xh=319       : rem koord. x-grenzwerte
160 yl=0 : yh=199       : rem koord. y-grenzwerte

170 rem achsenkoordinaten:
180 v1=xl+35 : v2=xh                        : rem senkrecht
190 h1=yl+15 : h2=yh-31 : h3=h2-(h2-h1)/3 : rem waagerecht
200 sb=((v2-v1)/24)-2                       : rem saeulenbreite

210 rem bildschirm-init
220 color 0,1 : color 4,1 : color 1,15 : rem farben
230 scnclr              : rem bildschirm loeschen

240 rem speicher-init:
250 dim we(11),mo$(11)
260 data jan,feb,mrz,apr,mai,jun
270 data jul,aug,sep,okt,nov,dez
280 for i=0 to 11 : read mo$(i) : next : rem monate lesen
```

```
290 input "grafiktitel";s$                  : rem eingabe
300 for i=0 to 11
310   print "wert fuer monat";mo$(i);". ";
320   input we(i)                           : rem wert einlesen
330   if we(i)>max then max=we(i)           : rem finde maximum?
340   if we(i)<min then min=we(i)           : rem und minimum?
350 next i

360 rem achsen zeichen:
370 graphic 1,1                             : rem grafik einschalten

380 ft=(h3-h1)/600                          : rem normal-faktor
390 if max<=600 and min>=-300 then 580      : rem innerhalb?
400 ft=(h3-h1-1)/abs(max)                   : rem neuer faktor
410 if h3+abs (min)*ft <=h2 then 580        : rem minimum auch ok?
420 ft=(h2-h3)/abs(min)                     : rem neuer faktor

430 draw 1,v1,h1 to v2,h1 : rem 1. waagerechte
440 draw 1,v1,h2 to v2,h2 : rem 2. waagerechte
450 draw 1,v1,h3 to v2,h3 : rem 3. waagerechte
460 draw 1,v1,h1 to v1,h2 : rem 1. senkrechte
470 draw 1,v2,h1 to v2,h2 : rem 2. senkrechte

480 ziel=h1   : stp=-(h3-h1)/6 : inc= 100
490 gosub 760
500 ziel=h2+1 : stp= (h2-h3)/3 : inc=-100
510 gosub 760

520 for i=1 to 3 : z=0
530   for j=v1+sb+3 to v2 step (v2-v1)/12
540     x=j-4 : y=h2+4+((i-1)*8)
550     t$=mid$ (mo$(z),i,1)
560     gosub 690                 : rem wert ausgeben
570     draw 1,j,h2+2 to j,h2-2   : rem einteilungen
580     z=z+1                     : rem index erhoehen
590   next j
600 next i

610 char 1,(xl+(xh-xl)/2-(len(s$)*4))/8+1,s$
```

```
620 rem werte eintragen:
630 x=0 : rem index
640 for i=v1+sb+3 to v2 step (v2-v1)/12
650    box 1,i-sb,h3-(we(x)*ft),i+sb,h3,0,1
660    x=x+1 : rem index erhoehen
670 next i

680 rem text(x,y,t$) ausgeben:
690 xa=(xl and 248)+8 : 248)+8
700 sshape q$,xa,ya,xa+len(t$)*8-1,ya+7
710 char 1,xa/8,ya/8,t$
720 sshape z$,xa,ya,xa+len(t$)*8-1,ya+7
730 gshape q$,xa,ya
740 gshape z$,x,y
750 return

760 z=0
770 for i=h3 to ziel step stp
780    x=xl : y=i-3 : t$=str$(z)
790    gosub 690                  : rem wert ausgeben
800    draw 1,v1-2,i to v1+2,i    : rem einteilungen setzen
810    z=z+inc                    : rem erhoehen/erniedrigen
820 next i : return
```

Um Ihnen das Verständnis dieses Programms etwas zu erleichtern, folgt in diesem Absatz eine kleine Beschreibung:

Nach der Definition der Bildschirmränder (wie im Programm zur Erstellung von Kurvengrafiken) werden die x- und y-Koordinaten der Achsen angegeben. Nun folgt die Berechnung der Breite aller Säulen, die sich aus der Anzahl der Säulen und der Bildschirmbreite ergeben.

Die ganze Grafik soll mit einer Überschrift versehen werden, die Sie nun ebenso eingeben können wie die verschiedenen Werte für die einzelnen Monate.

Auch hier (Sie bemerken die Ähnlichkeit mit der Kurvengrafik) werden nun Minimum und Maximum der Werte ermittelt und zur Berechnung eines Streckungsfaktors (y-Zerrfaktor) herangezogen. In diesem Fall bleiben aber alle Werte innerhalb eines Bereiches von -300 bis +600. Sollten sich diese Werte bereits von Anfang an innerhalb dieses Bereiches befinden, so bleibt der vorgegebene y-Zerrfaktor unverändert.

Achsen und Achsenbezeichnungen sowie die Achsenbeschriftungen stellen jetzt kein Problem mehr dar (für die Erklärung der Schreib-Unterroutine schauen Sie bitte bei den Erläuterungen zur Kurvengrafik nach). Zum Schluß werden noch die Balken eingezeichnet und die fertige Grafik liegt Ihnen zu Füßen.

3.2.3 Kuchendiagramme

Diese Art der grafischen Statistik ist geeignet zur übersichtlichen Darstellung von Ver- bzw. Aufteilungen von Mengen in Teilmengen und der Größenverhältnisse unter ihnen. Hierzu wird ein Kuchen (Kreis) gezeichnet, der die Gesamtheit aller zu betrachtenden Elemente darstellt (100%) und der sich in verschieden große Teilstücke unterteilt. Die Größe der Teilstücke gibt dann den Anteil dieser Teilmenge an der Gesamtmenge an.

Oft sehen wir solche Grafiken bei Wahlen zur Darstellung z.B. der Sitzverteilung im Bundestag etc. oder im Geographieatlas, um beispielsweise die Anteile bestimmter Import- oder Exportwaren eines Landes am Gesamtumschlag zu demonstrieren.

Doch wie ist solches Vorhaben programmtechnisch zu verwirklichen? Schließlich haben wir es mit der ziemlich komplizierten Relation eines Kreises zu tun, der in bestimmte Winkelausschnitte unterteilt werden muß.

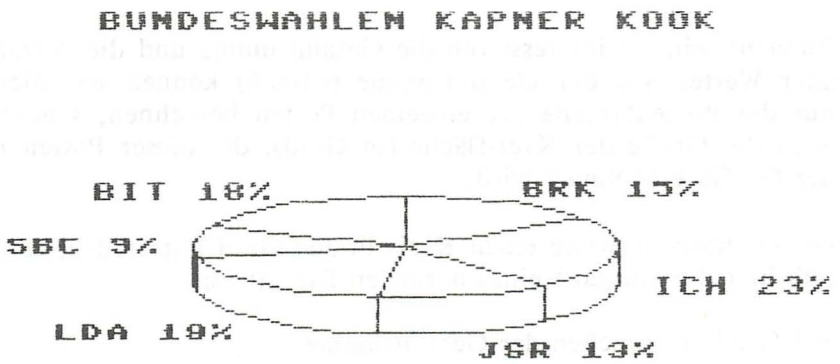
In dem folgenden Programm sollen verschiedenen Werten in einer Kuchengrafik ihrem prozentualen Anteil an der Gesamtheit entsprechend verschieden große Kuchenstücke zugeordnet werden. Das Ganze soll in einer ansprechenden und übersichtlichen Form grafisch dargestellt werden.

Bevor wir uns mit Einzelheiten beschäftigen, hier zunächst einmal das Programm:

```
100 rem **********************
110 rem **                    **
120 rem **   kuchengraphik    **
130 rem **                    **
140 rem **********************

150 rem werteinitialisierung:
160 xl= 0 : xh=319          : rem x-grenzwerte
170 yl= 0 : yh=199          : rem y-grenzwerte
180 mx=xl+(xh-xl)/2         : rem x-mittelpunkt
190 my=yl+(yh-yl)/2         : rem y-mittelpunkt
200 rx=(yh-yl)/3            : rem x-radius
210 ry=rx/3                 : rem y-radius
220 as=20                   : rem abstand text
230 fa=3.141562/180         : rem umrechnungsfaktor

240 rem bildschirminit:
250 color 0,1 : color 4,1 : rem farben
260 color 1,6 : scnclr    : rem bildschirm clr

270 rem speicherinit:
280 dim we(11),tx$(11)

290 rem werteeingabe:
300 input "ueberschrift    ";ue$
310 input "anzahl werte    ";az%
320 if az%>12  or az%<1 then 310          : rem anzahl ok?

330 for i=0 to az%-1
340    print i+1;". name"; : input tx$(i) : rem text eingeben
350    print i+1;". wert"; : input we(i)  : rem wert eingeben
360    tx$(i)=left$(tx$(i)+"   ",3)       : rem formatieren
370    su=su+we(i)                        : rem summe aller werte
380 next i
390 gs=360/su             : rem faktor
```

```
400 rem grafik zeichnen:
410 graphic 1,1            : rem grafik einschalten

420 rem kuchen zeichnen:
430 char 1,((xh-xl)/2-len(ue$)*4)/8+1,yl/8,ue$
440 circle 1,mx,my,rx,ry
450 circle 1,mx,my+15,rx,ry,90,270
460 draw 1,mx-rx,my to mx-rx,my+15
470 draw 1,mx+rx,my to mx+rx,my+15

480 rem kucheneinteilungen zeichnen:
490 for i=0 to az%-1
500    w  = (270+(lw+we(i))*gs)*fa      : rem winkelposition
510    wd = (270+(lw+we(i)/2)*gs)*fa    : rem textposition
520    x  = rx*cos(w)+mx                  : rem x-koord. der kreis-
pos.
530    y  = ry*sin(w)+my                : rem y-koord.
540    draw 1,mx,my to x,y              : rem radius einzeichnen
550    a  = (lw+we(i))*gs
560    if a<90 or a>270 then 710
570    draw 1,x,y to x,y+14              : rem vorderansicht zeich-
nen

580    rem text zeichnen:
590    x  = (rx+as/2)*cos(wd)+mx        : rem koordinaten berechnen
600    y  = (ry+as/2)*sin(wd)+my
610    t$ = str$(int(we(i)/(su/100)))
620    t$ = right$(t$,len(t$)-1)        : rem zahl isolieren
630    tx$(i)=tx$(i)+" "+t$+"%"         : rem % anhaengen

640    if x<mx then x=x-len (tx$(i))*8 : rem links oder
650    if y>my then y=y+10              : rem rechts v. mp
660    if x<xl or x>xh or y<yl or y>yh then 830 : rem ok?
670    t$=tx$(i) : gosub 710            : rem text ausgeben

680    lw=lw+we(i)                      : rem aufaddieren
690 next i

700 rem text bei Koordinaten (x,y) ausgeben:
```

```
710 xa=(xl and 248)+8 : ya=(yl and 248)+8
720 sshape q$,xa,ya,xa+len(t$)*8-1,ya+7
730 char 1,xa/8,ya/8,t$
740 sshape z$,xa,ya,xa+len(t$)*8-1,ya+7
750 gshape q$,xa,ya
760 gshape z$,x,y
770 return
```



Mit diesem Programm können Sie maximal 12 Werte mit Kennamen versehen als Ausgangswerte für ein entsprechendes Kuchendiagramm statistisch verarbeiten. Sie geben nach den üblichen Formalitäten (Überschrift, Anzahl der Werte), die Sie bereits aus anderen Statistikprogrammen kennen, einfach die Benennung und im Anschluß daran den Wert selbst ein. Die Benennung wird durch Zeile 360 auf drei Zeichen gekürzt und später als Beschriftung der einzelnen Kuchenausschnitte samt den zugehörigen Prozentzahlen in der Grafik ausgegeben.

Dieses Programm hat zunächst einmal die Aufgabe, einen Kreis durch Einzeichnung von Radien an verschiedenen Winkelpositionen zu unterteilen. Die Position der Radien hängt natürlich von den Prozentwerten der einzelnen Posten ab. Es stellt diese

Grafik zusätzlich noch 3-dimensional dar, um dem Auge beim Ablesen bessere Anhaltspunkte zu geben. Je nach den Ambitionen der verschiedenen Programmierer und selbstverständlich je nach den Ansprüchen, die an das Programm gestellt werden, können die einzelnen Kreisausschnitte noch in diverser Art und Weise schraffiert, ausgefüllt oder angefärbt werden.

Um die Winkelpositionen der einzelnen Radien zu ermitteln, gehen wir wie folgt vor:

Zunächst einmal interessieren die Gesamtsumme und die Anzahl aller Werte. Aus der Gesamtsumme (=100%) können wir nicht nur die Prozentanteile der einzelnen Posten berechnen, sondern auch die Größe der Kreisfläche (in Grad), die dieser Posten in der Grafik einnehmen wird.

Da wir normalerweise einen Kreis in 360 Grad unterteilen, stellt sich hier die Aufgabe eines normalen Dreisatzes:

360 Grad entsprechen der Gesamtsumme
Dann entsprechen wieviel Grad dem einzelnen Wert?

Nach Adam Riese gibt uns dann die folgende Formel den Faktor an, mit dem wir jeden Wert multiplizieren müssen, um schließlich die Gradzahl zu erhalten:

Faktor = 360 Grad / Gesamtsumme

Dieser Faktor wird in obigem Programm in der Zeile 390 berechnet.

Gradzahl = Faktor * Wert

ist dann die Formel, die uns direkt die Anzahl der Winkelgrade angibt, die die Größe des Kreisausschnittes bestimmen.

Nun stellt sich jedoch das Problem des Zeichnens. Die Darstellung eines Kreises auf unserem Grafikbildschirm nimmt uns freundlicherweise das Betriebssystem ab (CIRCLE-Befehl). Bei

der Einzeichnung der Radien jedoch sind wir auf unsere mathematischen Fähigkeiten angewiesen.

Zu diesem Zeitpunkt spätestens sollten Sie einmal einen Blick in das Kapitel über Polarkoordinaten geworfen haben, da wir hier darauf zurückgreifen müssen. Zur Angabe eines Punktes im Polarkoordinatensystem sind danach die Entfernung des Punktes vom Nullpunkt und der Winkel zwischen der Verbindungslinie Nullpunkt-Punkt und der waagerechten x-Achse notwendig. Ersteres übernimmt hier die Länge des Radius, den anderen Wert erhalten wir durch die gewünschte Position (in Grad) des einzuzeichnenden Radius auf dem Kreis.

In dem gerade erwähnten Kapitel wurde eine Formel zur Umrechnung von Polarkoordinaten in kartesische Koordinaten vorgestellt, die wir hier (leicht modifiziert durch die Addition der Kreismittelpunktskoordinaten) anwenden:

```
x = cos(w)*l + mx
y = sin(w)*l + my
```

Dabei bedeuten:

x,y: kartesische Koordinaten des gesuchten Punktes auf dem Kreis
w: Winkel des Radius
mx,my: Koordinaten des Kreismittelpunktes

Wie Sie vielleicht sehen, wurden diese Formeln in den Zeilen 520/530 im Programm verwendet.

Zur dreidimensionalen Aufarbeitung der Grafik wird nun noch ein etwas tiefer liegender Halbkreis gezogen, der die Vorderansicht des Kuchens abgrenzt. In dieser Frontansicht müssen die Radien natürlich ebenfalls sichtbar sein, um den gewünschten Effekt zu erzeugen. Hierzu wird in der Zeile 570 (sofern der Kucheneinschnitt überhaupt in der Frontseite liegt) einfach eine Senkrechte von der Kuchen-Oberfläche zum Kuchen-Boden gezogen.

Nach diesen recht kompliziert anmutenden Prozeduren werden nun noch an den entsprechenden Stellen die Beschriftungen in die Grafik eingebracht und fertig ist der Kuchen.

Auch bei diesem Programm sollten Sie ruhig einmal den Cursor in die Hand nehmen und selbst herumdoktern. Nur so erfassen Sie die volle Tragweite der einzelnen Anweisungen, vor allem aber der im Programmkopf definierten Konstanten.

3.3 Funktionsplotter

Wir wollen uns nun mit den sogenannten Funktionsplottern beschäftigen. Sicherlich erinnern Sie sich noch an Ihre Schulzeit, an den Mathematikunterricht der 7., 8. und 9. Klasse und an Funktionen, Parabeln, Hyperbeln und andere diversen Funktionsgraphen (oh je). Sie hatten dort sicherlich mit einer Funktion f(x) Wertetabellen anzulegen und diese dann in ein Koordinatensystem einzutragen. Bei einfachen Funktionen wie z.B. f(x)=x oder f(x)=x*x war dieses Einzeichnen in das Koordinatensystem sicherlich noch leicht zu bewältigen, wurden die Funktionen dann jedoch komplizierter (z.B.: f(x)=x^3+2*x^2-4*x), hatte man doch seine Müh' und Not mit diesen Funktionen.

Doch wofür haben Sie einen Computer?

Dieser schnelle Rechner kann Werte einer jeden Funktion errechnen, ja sogar zeichnen. Mit einer unserer Hardcopy-Routinen aus dem Kapitel 5 sollten Sie in der Lage sein, die gezeichnete Funktion auch auszudrucken, um diese dann schwarz auf weiß weiter zu verwerten.

Obwohl uns eigentlich nur die grafische Funktionsauflösung interessiert, möchte ich Ihnen doch noch ein kleines Programm zum Anlegen von Wertetabellen vorstellen:

```
10 scnclr
20 :
30 def fn f(x)=x*x          :rem hier kann auch eine andere funktion
stehen
```

```
40 :
50 input"bereich (von; bis)";bv,bb  :rem eingabe berechnungsbereich
60 :
70 input"schrittweite      ";sw     :rem eingabe der schrittweite
80 :
90 for x =bv to bb step sw          :rem berechnungsschleife
100 :
110 print "x-
wert =";x;"  funktion  f(x) =";fn f(0):rem ausgebe der funktion
120 :
130 getkey a$                       :rem auf taste warten
140 :
150 next x                          :rem schleife schliessen
160 :
170 end                             :rem ende des programmes
```

Doch so ein Programm ist natürlich im Vergleich mit einem Funktionsplotter nur eine Spielerei, deshalb folgt im nächsten Kapitel eben ein solches Programm.

3.3.1 2-D-Funktionsplotter

```
100 scnclr
110 key 8,"-10,10"+chr$(13)
120 color 1,8,7
130 color4,7,0:color0,7,0
140 :
150 char1,10,0,"********************"
160 char1,10,1,"**   f-plotter   **"
170 char1,10,2,"********************"
180 :
190 f$="x*x"
200 :
210 char1,16,5,f$
220 char1,0,5,"funktion f(x)=":input f$
230 :
```

```
240 printchr$(147);"190 f$="+chr$(34)+f$+chr$(34)
250 print"680 f$="+chr$(34)+f$+chr$(34)
260 print"690 deffn f(u)="+f$
270 print"goto 310"
280 poke1319,19:poke1320,13:poke1321,13:poke1322,13:poke1323,13:po
ke239,5
290 end
300 :
310 trap 1020
320 scnclr
330 :
340 input "grenzen des x-werts";xk,xg
350 :: if xk>xg then 340
360 input "grenzen des y-werts";yk,yg
370 :: if yk>yg then 360
380 input "schrittweite      ";sw
390 :: if sw=0 then sw=.5
400 :
410 input "x-koordinaten eint";xe
420 :
430 input "y-koordinaten eint";ye
440 :
450 print"alte grafik loeschen [j/n]":getkey l$
460 :
470 graphic1,0:if l$="j" then scnclr
480 :
490 mx=199/2:my=199/2
500 :
510 xf=mx/xg
520 yf=my/yg
530 :
540 draw1,0,my to 199,my:draw 1,mx,0 to mx,199
550 :
560 if xe=0 and ye=0 then 660
570 :
580 for i=0 to 199 step xe*xf
590 :draw 1,i,95 to i,104
600 next
610 :
620 for i=0 to 199 step ye*yf
```

```
630 :draw 1,95,i to 104,i
640 next
650 :
660 draw1,199,0 to 199,199:box1,0,0,319,199
670 :
680 f$="x*x"
690 deffn f(u)=x*x
700 :
710 for x=xk to xg step sw
720 :y1=my-fn f(0)*yf
730 :x1=(mx+x*xf)
740 :if x=xk then x2=x1:y2=y1
750 :if x1<0 or y1<0 or x2<0 or y2<0 then 770
760 ::  draw1,x1,y1  to x2,y2
770 :x2=x1:y2=y1
780 next
790 :
800 if l$<>"j"then960
810 :
820 :char1,28, 0,"f-plotter",1
830 :char1,25, 2,"f(x)=",1
840 :char1,25, 3,f$
850 :char1,25, 5,"x-bereich:",1
860 :char1,25, 6,str$(xk)+";"+str$(xg)
870 :char1,25, 8,"y-bereich:",1
880 :char1,25, 9,str$(yk)+";"+str$(yg)
890 :char1,25,12,"schrittweite:",1
900 :char1,25,13,str$(sw)
910 :char1,25,15,"x-einteilung:",1
920 :char1,25,16,str$(xe)
930 :char1,25,18,"y-einteilung:",1
940 :char1,25,19,str$(ye)
950 :
960 color 4,2
970 :
980 getkey a$:graphic0
990 run
1000 :
```

```
1010 :
1020 if er=20 then x=x+sw:resume next
1030 if er=11 then graphic0:print chr$(130)+"fehler in funktion":e
nd
1040 graphic0:print err$(er):end
```

Dieses Programm ist, wie Sie sicherlich bemerkt haben, nicht dokumentiert. Das war nötig, da das Funktionsplotterprogramm auch auf der Grundversion des C 16/ C 116 laufen sollte und dann der Speicherplatz nicht ausgereicht hätte. Selbst mit dieser schon ziemlich optimierten Version bleiben nach dem Start und dem Anlegen aller Variablen nur noch magere 70 (siebzig) Bytes frei. Wenn Sie also dieses Programm abgetippt haben, sollten Sie es erst einmal abspeichern, den schon ein paar zuviel getippte Buchstaben könnten den BASICspeicher an die Grenzen seines Fassungsvermögens bringen.

Sollten Sie vorhaben dieses Programm noch ein wenig zu verschönern, empfehle ich Ihnen, alle Zeilen in denen nur Doppelpunkte stehen, einfach zu löschen. Diese Zeilen werden vom Programm nicht angesprungen. Wollen Sie allerdings das Programm "RENUMBERN", müssen Sie auch in den Zeilen 240-270 die Zeilennummern und GOTO-Adressen innerhalb der Anführungsstriche umändern, da der RENUMBER-Befehl diese natürlich nicht berücksichtigen kann.

Um Ihnen das Lesen des Programms dennoch zu vereinfachen, folgt hier nun die fehlende Dokumentation:

100	Löschen des Bildschirms
110	Umbelegen der HELP-Taste (wird bei der Grenzwerteingabe gebraucht)
120,130	Setzen der Farben. Gelb für Schrift; Blau für Rahmen & Hintergrund
150-170	Ausgabe des Titels
190	Alte Funktion in f$
210,220	Eingabe der neuen Funktion

240-290	Übernehmen der neuen Funktion per Tastaturbuffermethode
310	Bei Fehlermeldung Sprung nach 1020
320	Bildschirm löschen
340-370	Eingabe der X- und Y-Grenzwerte mit Überprüfung auf Gültigkeit
380,390	Eingabe der Schrittweite. Bei 0 wird die Schrittweite zu 0.5
410,430	Eingabe der Koordinateneinteilung von X- und Y-Achse
450	Fragen, ob alte Funktion gelöscht werden soll
470	Einschalten der Grafik und event. Löschen derselben
490	Festlagen des Mittelpunktes der Funktion
510,520	Festlegen des X- und Y-Umrechnungsfaktors
540	Zeichnen des Koordinatensystems
560	Falls keine Koordinateneinteilung angegeben, Sprung nach 660
580-600	Einteilen der X-Achse
620-640	Einteilen der Y-Achse
660	Zeichnen einer Trennlinie zwischen Funktion und Wertangaben und Umrahmen der Funktion
680	Funktion in f$
690	Definition der Funktion mittels DEF FN f(x)
710	Schleife vom kleinsten Grenzwert zum größten mit der eingegebenen Schrittweite
720	Umrechnen der Y-Koordinate in das Format des Plotters
730	Dasselbe mit der X-Koordinate
740	Überprüfen, ob es sich um den ersten Schleifenwert handelt
750	Überprüfen, ob die Koordinaten im erlaubten Bereich liegen
760	Einzeichnen der Koordinaten durch Verbinden des letzten Wertes mit der neuen Koordinate
770	Merken der letzten Koordinaten
780	Berechnungsschleife schließen
800	Falls Grafik vorher nicht gelöscht war, Überspringen der Beschriftungsteile

820-940	Beschriften der Grafik mit allen angegebenen Werten
960	Rahmenfarbe auf Weiß, um anzuzeigen, daß die Funktion fertig gezeichnet ist
980	Auf Tastendruck warten und auf den Textbildschirm schalten
990	Erneut starten
1020	Falls ein DIVISION BY ZERO ERROR auftritt, Rechenwert erhöhen und erneut probieren
1030	Falls ein Fehler in der Funktion auftritt, diese melden und stoppen
1040	Jeden anderen Fehler anzeigen und stoppen

Ich hoffe, daß Ihnen der Programmaufbau nun klarer ist, daher werde ich nun die Beschreibung dieses Programmes vorstellen

Sie starten das Funktionsplotterprogramm ganz normal mit RUN. Danach erscheint ein kleines Titelbild und die zuletzt berechnete Funktion. Sie können diese dann durch Drücken von RETURN übernehmen oder eine neue Funktion f(x) eingeben. Dies geschieht in der üblichen BASIC-Syntax, also statt:

$$f(x)= \frac{2x}{1/x}$$

schreiben Sie nun:

```
f(x)= (2*x)/(1/x)
```

Sollten Sie Schwierigkeiten mit der Eingabe der korrekten Formel haben, empfiehlt es sich, im Handbuch auf den Seiten 183 bis 189 und 210 die wichtigsten Sachen noch einmal nachzulesen.

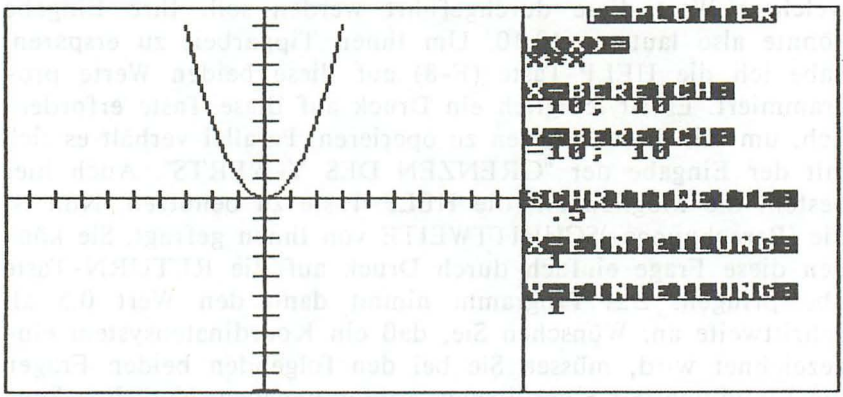
Haben Sie zum Abschluß der Formeleingabe RETURN gedrückt, so wird die Formel mit Hilfe der sog. "Tastaturbuffer-Methode" in drei BASIC-Zeilen übernommen. Doch zurück zu

unserem Programm. Dieses verlangt nämlich jetzt von Ihnen die Eingabe der "GRENZEN DES X-WERTS". Sie sollen also angeben, von welchem Wert an die Berechnung beginnen und bis zu welchem Wert diese durchgeführt werden soll. Ihre Eingabe könnte also lauten: -10,10. Um Ihnen Tipparbeit zu ersparen, habe ich die HELP-Taste (F-8) auf diese beiden Werte programmiert. Es ist lediglich ein Druck auf diese Taste erforderlich, um mit diesen Werten zu operieren. Parallel verhält es sich mit der Eingabe der "GRENZEN DES Y-WERTS". Auch hier besteht die Möglichkeit, die HELP-Taste zu benutzen. Nun ist die (Berechnungs-)SCHRITTWEITE von Ihnen gefragt. Sie können diese Frage einfach durch Druck auf die RETURN-Taste überspringen. Das Programm nimmt dann den Wert 0.5 als Schrittweite an. Wünschen Sie, daß ein Koordinatensystem eingezeichnet wird, müssen Sie bei den folgenden beiden Fragen die Einteilung der Koordinaten-Achsen angeben. Natürlich können Sie auch auf das Koordinatensystem verzichten, indem Sie bei beiden Fragen nur die RETURN-Taste drücken. Schließlich möchte der Funktionsplotter noch von Ihnen "wissen", ob die zuletzt berechnete Funktion erst vom Grafik-Bildschirm gelöscht werden soll. Geben Sie also J [Ja] oder N [Nein] ein.

Damit ist Ihre Arbeit erledigt. Sie können sich nun in Ihren Sessel zurücklegen und zuschauen, wie der Funktionsplotter die eingegebene Funktion berechnet. Sind dann alle Berechnungen abgeschlossen, werden im rechten Teil des Bildschirms alle wichtigen Daten eingeblendet. Sollten Sie also die Funktion ausdrucken oder abspeichern wollen, wissen Sie immer, was Sie dort für eine Grafik vorliegen haben. Schließlich schaltet der Bildschirmrahmen noch auf Weiß um. Auf Knopfdruck startet der Funktionsplotter dann erneut.

Es kann natürlich vorkommen, daß eine Funktion fehlerhaft von Ihnen eingegeben wurde - doch keine Angst, der Computer stürzt nicht ab. Er zeigt Ihnen durch eine kleine Fehlermeldung, daß ein Fehler in der Funktion vorliegt. Auch der so gefürchtete "DIVISION BY ZERO ERROR" wird abgefangen. Das Programm erhöht den Rechenwert und führt die gleiche Berechnung dann mit dem neuen Rechenwert durch. Selbst wenn Sie spaßeshalber eingeben f(x)=x/0, wird nichts passieren.

Hier nun zur Verdeutlichung ein Ergebniß des Funktionsplotters:



3.3.2 3-D-Funktionsplotter

```
100 scnclr
110 key 8,"-10,10"+chr$(13)
120 color 1,8,7
130 color4,7,0:color0,7,0
140 :
150 char1,10,0,"********************"
160 char1,10,1,"** 3-d f-plotter **"
170 char1,10,2,"********************"
180 :
190 f$="6*(sin(.31*x)*sin(.31*y))-9"
200 :
210 char1,9,5,f$
220 char1,0,5,"f(x,y)=":input f$
230 :
```

```
240 printchr$(147);"190 f$="+chr$(34)+f$+chr$(34)
250 print"610 deffn f(u)="+f$
260 print"goto 300"
270 poke 1319,19:poke1320,13:poke1321,13:poke1322,13:poke239,4
280 end
290 :
300 trap 870
310 scnclr
320 :
330 input "grenzen des x-wert";xk,xg
340 :if xk>xg then 330
350 input "grenzen des y-wert";yk,yg
360 :if yk>yg then 350
370 input "grenzen des z-wert";zk,zg
380 :if zk>zg then 370
390 input "x-schrittweite";sx
400 :if sx=0 then sx=.4
410 input "y-schrittweite";sy
420 :if sy=0 then sy=1
430 input"aufsichtswinkel [1-89]";sw
440 :if sw<=0 or sw>=89 then sw=55
450 :
460 print"verdecktes loeschen":getkey v$
470 :
480 print"koordinatensystem":getkey k$
490 :
500 print"grafik loeschen":getkey l$
510 :
520 graphic1,0:if l$="j" then scnclr
530 :
540 gr=3.14159265/180:rem pi/180
550 si=sin(sw*gr):co=cos(sw*gr)
560 fx=abs(319/(xg-xk))
570 fz=abs(199/(zg-zk))
580 fy=abs((fz*(zk/co)-(zg/si))/yg-yk)
590 :
600 :
610 deffn f(u)=6*(sin(.31*x)*sin(.31*y))-9
620 deffn xr(u)=x*fx+y*fy*co+160
630 deffn yr(u)=199-(fz*fn f(0)+y*fy*si+100)
```

```
640 :
650 :for y=yg to yk step -sy
660 ::for x=xk to xg step sx
670 ::y1=fn yr(0)
680 ::x1=fn xr(0)
690 ::if x=xk then x2=x1:y2=y1
700 ::if x1<0 or y1<0 or x2<0 or y2<0 then 730
710 ::if v$="j" then gosub 910
720 ::draw1,x1,y1 to x2,y2
730 ::x2=x1:y2=y1
740 ::next x
750 :next y
760 :
770 if k$<>"j" then 830
780 :draw 1,0,100 to 319,100
790 :draw 1,160,0 to 160,199
800 :draw 1,160,100 to 100; 90-sw
810 :draw 1,160,100 to 100;270-sw
820 :
830 color 4,2
840 :
850 getkey a$:graphic0:run
860 :
870 if er=20 then resume 740
880 if er=11 then graphic0:print chr$(147)+chr$(130)+"fehler in
funktion":end
890 graphic0:print err$(er):stop
900 :
910 if y2=y1 then 940
920 ify2>y1thenfori=y1toy2:draw0,x2,y2tox1,i:next:box0,x2,y2,x1,y2
+50,,1
930 ify1>y2thenfori=y2toy1:draw0,x1,y1to x2,i:next:box0,x2,y1,x1,y
1+50,,1
940 return
```

Auch hier die aus Speicherplatzgründen fehlende Dokumentation. Hierbei möchte ich nur auf die wichtigsten Eigenschaften

und Änderungen eingehen, da der 3-D- Funktionsplotter auf dem 2-D-Funktionsplotter aufbaut.

100-430	(siehe Dokumentation des 2-D-Funktionsplotters).
460-500	Diverse Eingaben für Löschen, verdeckte Linien, Koordinatensystem
540	Umrechnungsfaktor Grad im Radiant
550	Scalierter Sinus und Kosinus
560-580	Berechnung der X-,Y-,Z-Faktoren
610-630	Definition
650	Y-Schleife rückwärts
660	X-Schleife vorwärts
670-750	siehe Dokumentation des 2-D-Funktionsplotters).
710	Falls das Löschen verdeckter Linien erwünscht ist, zum Unterprogramm springen
770-810	Koordinatensystem
830-890	(siehe Dokumentation des 2-D-Funktionsplotters).
910	Falls die zuletzt gezeichnete Linie waagerecht verlief, nur den unteren Block löschen
920	Falls die letzte Linie nach unten gezeichtet wurde, löschen des linken Dreiecks bis zur Waagerechten. Danach Löschen des tieferen Blocks
930	Fals die letzte Linie nach oben gezeichnet wurde, löschen des rechten Dreiecks bis zur Waagerechten. Danach löschen des tieferen Blocks
940	Rückkehr zum Hauptprogramm

Beschreibung des 3-D-Funktionsplotters:

Der 3-D-Funktionsplotter ist wie schon erwähnt "nur" eine Weiterentwicklung des 2-D-Funktionsplotters. Daher sind viele Teile ähnlich oder identisch. So auch die auf "-10,10" programmierte HELP-Taste.

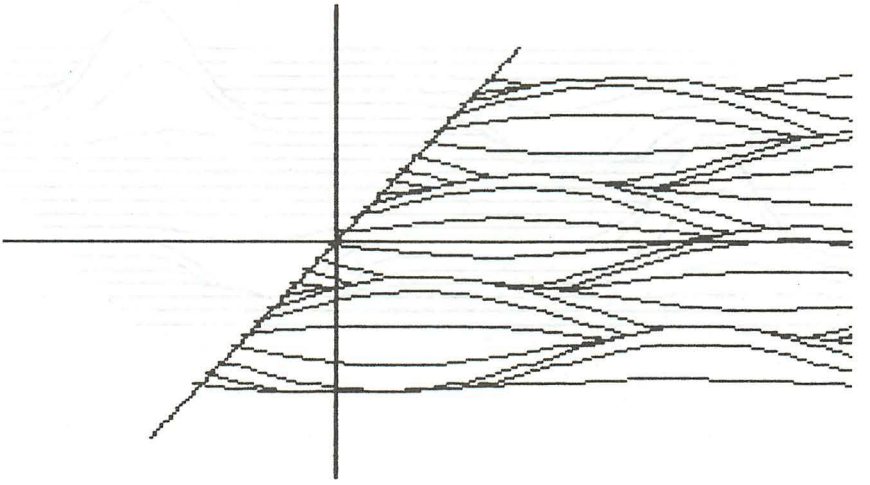
Das Programm fragt nach dem Start durch RUN nach der Funktion f(x,y). Diese ist in der Üblichen BASIC-Syntax einzugeben. Die Funktion wird dann mit der Tastaturpuffermethode übernommen. Es folgen die schon bekannten Fragen nach Grenzen der einzelnen Achsen und der Berechnungsschrittweite. Neu ist jedoch die Frage nach dem Aufsichtswinkel. Hier können Sie eingeben, mit welchem Kippwinkel die Grafik berechnet werden soll. Eingabewerte von 1-89 sind erlaubt. Bei einer Falscheingabe nimmt das Programm einen Winkel von 55 Grad an. Übrigens sind nach meiner Meinung nur Eingaben von 30-60 Grad interessant.

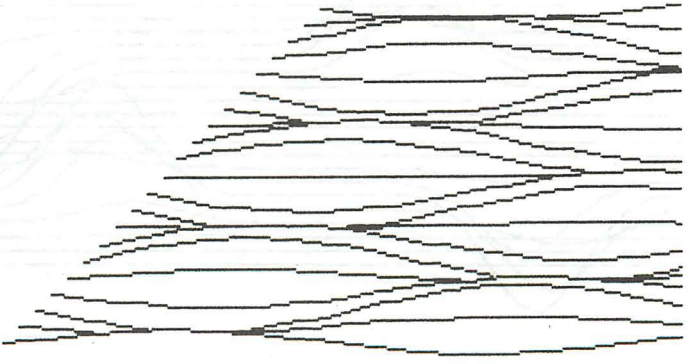
Auch hinzugekommen ist die Frage nach dem Löschen verdeckter Linien. Um diesen Programmpunkt zu verstehen, sollten Sie sich einige der abgedruckten Grafiken ansehen. Die Frage nach dem Löschen der alten Grafik erklärt sich von selbst, wie auch die Frage nach dem Koordinatensystem. Letzteres wird selbstverständlich dreidimensional eingezeichnet und zwar mit dem Befehl: "DRAW x1,y1 to Länge ; Winkel", also mit Hilfe des Polarkoordinatensystems.

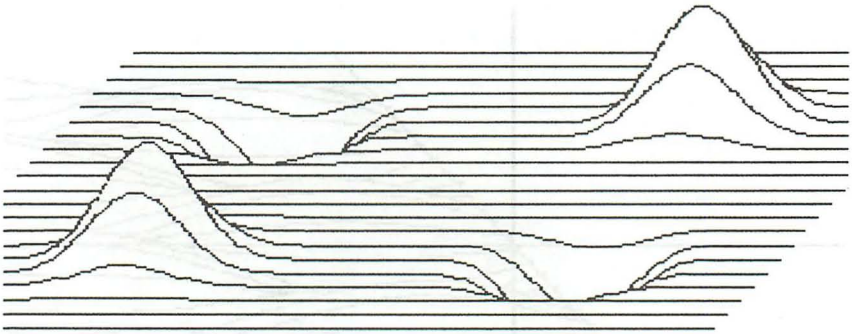
Jetzt können Sie sich in Ihren Sessel zurücklehnen und zusehen, wie die Funktion geplottet wird. Je nachdem, was für eine Schrittweite eingegeben und ob verdeckte Linien gelöscht werden sollen oder nicht, dauert das Zeichen einer Funktion 2-200 Minuten. Die Grafiken können dann mittels der vorgestellten Routinen im Kapitel 5 ausgedruckt oder gespeichert werden.

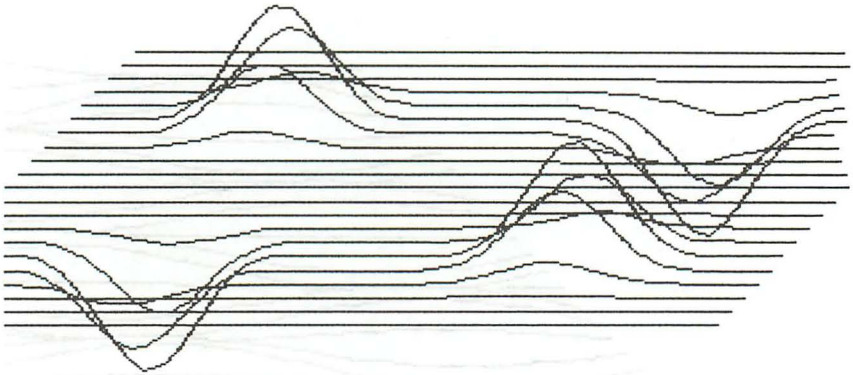
Um Sie von der Leistungsfähigkeit des Programmes zu überzeugen, haben wir Ihnen noch ein paar Grafiken abgedruckt.

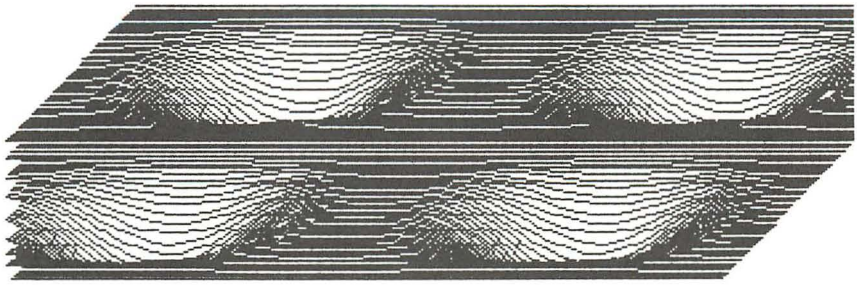
Zuerst eine typische Hardcopy einmal mit und anschließend ohne Koordinatensystem, dann zwei Hardcopies, an denen wir das Prinzip der Hinterschneidungen verdeutlichen möchten und zu guter Letzt noch zwei Hardcopies, die keinen tieferen Sinn haben, die es aber dennoch Wert sind, ausgedruckt zu werden.

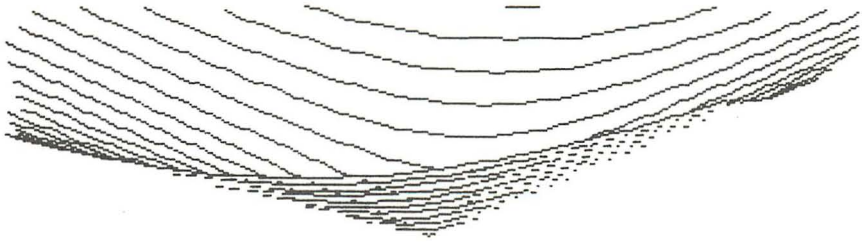












4. Ein- und Ausgabe von Grafiken

4.1 Einführung

In diesem letzten Kapitel unseres Buches wollen wir Ihnen nun noch einige in Assembler geschriebene Utilities anbieten, die die Ein- und Ausgabe der Grafiken betrifft. Das wären zunächst Hardcopyroutinen für die gängigsten Drucker, anschließend auch noch ein Screen-Utility-Programm, das das Abspeichern und Laden der Textgrafik als auch der Hiresgrafik zum Kinderspiel werden läßt. Diese Programme können Sie natürlich in Ihre eigenen Programme einbauen. Zu diesem Zweck befindet sich hinter dem für die "Könner" unter Ihnen, abgedruckten Assemblerlisting auch der dazugehörige BASIC-Lader.

4.2 Hardcopyroutinen

Im folgenden ist nun das Assemblerlisting der 1. Hardcopyroutine abgedruckt. Dieses Programm erzeugt eine Hardcopy, also einen Bildschirmausdruck, auf allen Star- und kompatiblen Druckern. Im Anschluß daran finden Sie die BASIC-Laderoutine für Epsondrucker. Sollten Sie einen grafikfähigen Drucker besitzen, mit dem die nachstehende Routine nicht funktioniert, so können Sie die in Ihrem Handbuch entsprechenden Codes zum Einschalten der Grafik mit doppelter Dichte in die Zeile 840, rückwärts aufgezählt, eintragen. Der Anzahl der Codes entsprechend müssen Sie dann natürlich auch den Wert, mit dem das Y-Register in Zeile 630 geladen wird, angleichen.

Hier nun das Assemblerlisting, direkt dahinter der BASIC-Lader zunächst für die Star, anschließend auch für alle Epsondrucker:

4.2.1 Hardcopyroutine für Star-Drucker

```
110:   05f5                    .opt p,oo
120:   05f5                    *=   1525
130:                  ;
```

```
140:   9491          chkcom  =   $9491
150:   9d84          getbyt  =   $9d84
160:   05f5 20 91 94         jsr  chkcom
160:   05f8 20 84 9d         jsr  getbyt
160:   05fb e0 00            cpx  #0
160:   05fd d0 03            bne  skip
170:   05ff 20 c7 06         jsr  ctg
180:   0602 20 91 94 skip    jsr  chkcom
180:   0605 20 84 9d         jsr  getbyt
180:   0608 8a               txa
180:   0609 48               pha
190:   060a 20 91 94         jsr  chkcom
190:   060d 20 84 9d         jsr  getbyt
200:                 ;
210:   0610 a9 00            lda  #<8192
210:   0612 a0 20            ldy  #>8192
220:   0614 85 fa            sta  $fa
220:   0616 84 fb            sty  $fb     ; basis adresse setzen
230:                 ;
240:   0618 68               pla
240:   0619 48               pha
250:   061a a0 01            ldy  #1      ; sekundaeradresse
260:   061c 20 ba ff         jsr  $ffba   ; setpara
270:   061f a9 00            lda  #0
280:   0621 20 bd ff         jsr  $ffbd   ; setnam
290:   0624 20 c0 ff         jsr  $ffc0   ; open
300:   0627 68               pla
310:   0628 48               pha
320:   0629 aa               tax
330:   062a 20 c9 ff         jsr  $ffc9   ; chkout
340:                 ;
350:   062d a9 1b            lda  #27     ; escape "1" senden,
360:   062f 20 d2 ff         jsr  $ffd2   ; dadurch zeilenabstand
370:   0632 a9 31            lda  #"1"    ; setzen
380:   0634 20 d2 ff         jsr  $ffd2
390:                 ;
400:   0637 20 54 06         jsr  hardcopy ; hardcopy drucken
410:                 ;
420:   063a a9 1b            lda  #27     ; escape "2" senden,
430:   063c 20 d2 ff         jsr  $ffd2   ; dadurch zeilenabstand
```

```
440:  063f a9 32           lda  #"2"    ; normalisieren
450:  0641 20 d2 ff        jsr  $ffd2
460:  0644 20 cc ff        jsr  $ffcc   ; clrch
470:  0647 68              pla
480:  0648 4c c3 ff        jmp  $ffc3   ; close
490:                ;
500:  064b 00       bytes  .byt %00000000
510:  064c 00              .byt %00000000
520:  064d 00              .byt %00000000
530:  064e 00              .byt %00000000
540:  064f 00              .byt %00000000
550:  0650 00              .byt %00000000
560:  0651 00              .byt %00000000
570:  0652 00              .byt %00000000
580:                ;
590:  0653 00       spalte .byt 0       ; spaltenzaehler
600:                ;
610:  0654 a2 18    hardcopy ldx  #24   ; zeilenzaehler setzen
620:  0656 a9 0d    lab1     lda  #13
620:  0658 20 d2 ff         jsr  $ffd2  ; zeilenvorschub
630:  065b a0 04            ldy  #4
640:  065d b9 84 06 loop    lda  drcktab,y  ; steuerzeichen
650:  0660 20 d2 ff         jsr  $ffd2  ; ausgeben
660:  0663 88               dey
660:  0664 10 f7            bpl  loop
670:                ;
680:  0666 a9 27            lda  #39    ; spaltenzaehler
690:  0668 8d 53 06         sta  spalte ; setzen
700:  066b a0 00    lab2    ldy  #0     ; y-register loeschen
710:  066d 20 89 06         jsr  sendacht  ; 8ter block ausgeben
720:  0670 a5 fa            lda  $fa    ; $fa/$fb = $fa/$fb + 8
730:  0672 18               clc
740:  0673 69 08            adc  #8
750:  0675 90 02            bcc  ok1
760:  0677 e6 fb            inc  $fb
770:  0679 85 fa    ok1     sta  $fa
780:  067b ce 53 06         dec  spalte ; spalte=spalte-1
790:  067e 10 eb            bpl  lab2
800:  0680 ca               dex
```

```
810:   0681 10 d3              bpl  lab1
820:   0683 60                 rts
830:                ;
840:   0684 02 80 4c drcktab   .byt 2,128,76,27,9
850:                ;
860:                ;*** drehen und setzen eines 8ter blocks ***
870:                ;
880:   0689 8a       sendacht txa
880:   068a 48                 pha
880:   068b 98                 tya
880:   068c 48                 pha
890:   068d a9 80               lda  #$80
900:   068f 8d c6 06            sta  maske
910:   0692 a2 07               ldx  #7
920:   0694 a0 00    hrdloop1  ldy  #0
930:   0696 b1 fa    hrdloop2  lda  ($fa),y
940:   0698 2d c6 06            and  maske
950:   069b 38                  sec
960:   069c d0 01               bne  ok2
970:   069e 18                  clc
980:   069f 3e 4b 06 ok2        rol  bytes,x
990:   06a2 c8                  iny
1000:  06a3 c0 08               cpy  #8
1010:  06a5 d0 ef               bne  hrdloop2
1020:  06a7 4e c6 06            lsr  maske
1030:  06aa ca                  dex
1030:  06ab 10 e7               bpl  hrdloop1
1040:               ;
1050:  06ad a2 07               ldx  #7     ; bytes ausgeben
1060:  06af bd 4b 06 outlab     lda  bytes,x
1070:  06b2 20 d2 ff            jsr  $ffd2
1070:  06b5 20 d2 ff            jsr  $ffd2
1070:  06b8 a9 00               lda  #0
1070:  06ba 9d 4b 06            sta  bytes,x
1080:  06bd ca                  dex
1090:  06be 10 ef               bpl  outlab
1100:  06c0 68                  pla
1100:  06c1 a8                  tay
1100:  06c2 68                  pla
```

```
1100:  06c3 aa              tax
1100:  06c4 60              rts
1110:                  ;
1120:  06c5 00         temp     .byt 0
1130:  06c6 00         maske    .byt 0
1140:                  ;
1150:                  ;copy text --> grafik
1160:                  ;
1170:  0c00            video    =   3072
1180:  2000            grafik   =   8192
1190:  d000            char     =   $d000
1200:                  ;
1210:  06c7 a9 00      ctg      lda #<video
1210:  06c9 a0 0c               ldy #>video
1220:  06cb 85 61               sta $61
1220:  06cd 84 62               sty $62
1230:  06cf a9 00               lda #<grafik
1230:  06d1 a0 20               ldy #>grafik
1240:  06d3 85 63               sta $63
1240:  06d5 84 64               sty $64
1250:  06d7 78                  sei
1250:  06d8 a9 33               lda #51
1250:  06da 85 01               sta 1
1260:  06dc a0 04               ldy #4
1260:  06de 8c 5a 07            sty ykord
1270:  06e1 a0 00               ldy #0
1280:  06e3 a2 00               ldx #0
1290:  06e5 b1 61      cclab0   lda ($61),y ; zeichen laden
1291:  06e7 c9 80               cmp #128
1292:  06e9 90 0b               bcc ccskip1
1292:  06eb 29 7f               and #127
1292:  06ed 48                  pha
1293:  06ee a9 ff               lda #255
1293:  06f0 8d 5b 07            sta flag
1293:  06f3 4c fc 06            jmp ccskip2
1294:  06f6 48         ccskip1  pha
1294:  06f7 a9 00               lda #0
1295:  06f9 8d 5b 07            sta flag
1296:                  ;
1300:  06fc 68         ccskip2  pla
```

```
1300:  06fd 85 65          sta  $65
1300:  06ff 48             pha
1310:  0700 a9 00          lda  #0
1310:  0702 85 66          sta  $66
1320:  0704 68             pla
1330:  0705 0a             asl
1330:  0706 26 66          rol  $66
1340:  0708 0a             asl
1340:  0709 26 66          rol  $66
1350:  070b 0a             asl
1350:  070c 26 66          rol  $66
1360:  070e 85 65          sta  $65
1370:  0710 a9 00          lda  #<char
1370:  0712 18             clc
1370:  0713 65 65          adc  $65
1370:  0715 85 65          sta  $65
1380:  0717 90 02          bcc  ccok1
1380:  0719 e6 66          inc  $66
1390:  071b a9 d0    ccok1 lda  #>char
1390:  071d 18             clc
1390:  071e 65 66          adc  $66
1400:  0720 85 66          sta  $66
1410:                ;
1420:  0722 a0 00          ldy  #0
1430:  0724 b1 65    cclab1 lda  ($65),y
1430:  0726 4d 5b 07       eor  flag
1430:  0729 91 63          sta  ($63),y
1440:  072b c8             iny
1440:  072c c0 08          cpy  #8
1440:  072e d0 f4          bne  cclab1
1450:  0730 a0 00          ldy  #0
1460:  0732 e6 61          inc  $61
1460:  0734 d0 02          bne  ccok2
1460:  0736 e6 62          inc  $62
1470:  0738 a5 63    ccok2 lda  $63
1470:  073a 18             clc
1470:  073b 69 08          adc  #8
1470:  073d 85 63          sta  $63
1480:  073f 90 02          bcc  ccok3
1480:  0741 e6 64          inc  $64
```

```
1490:  0743 ca       ccok3   dex
1490:  0744 d0 9f            bne  cclab0
1510:  0746 ac 5a 07         ldy  ykord
1510:  0749 88               dey
1510:  074a f0 08            beq  skip1
1520:  074c 8c 5a 07         sty  ykord
1520:  074f a0 00            ldy  #0
1520:  0751 4c e5 06         jmp  cclab0
1530:  0754 a9 77    skip1   lda  #119
1530:  0756 85 01            sta  1
1530:  0758 58               cli
1530:  0759 60               rts
1540:                ;
1550:  075a 00       ykord   .byt 0
1560:  075b 00       flag    .byt 0
]05f5-075c
no errors
```

Hier nun der dazugehörige BASIC-Lader:

```
100 rem ********************************
110 rem ***                            ***
120 rem ***                            ***
130 rem ***     hardcopy star          ***
140 rem ***    sg/sd/sr 10/15          ***
150 rem ***   ---------------          ***
160 rem ***                            ***
170 rem ********************************
180 :
190 for x=1525 to 1882
200 read a$:a=dec(a$):p=p+a:poke x,a
210 next x
220 :
230 if p<>2525424 then print "fehler in datas !!!":end
240 print "ok"
250 end
260 :
```

```
270 rem assemblerdatas
280 :
290 data 20,91,94,20,84,9d,e0,00,d0,03,20
300 data c7,06,20,91,94,20,84,9d,8a,48,20
310 data 91,94,20,84,9d,a9,00,a0,20,85,fa
320 data 84,fb,68,48,a0,01,20,ba,ff,a9,00
330 data 20,bd,ff,20,c0,ff,68,48,aa,20,c9
340 data ff,a9,1b,20,d2,ff,a9,31,20,d2,ff
350 data 20,54,06,a9,1b,20,d2,ff,a9,32,20
360 data d2,ff,20,cc,ff,68,4c,c3,ff,00,00
370 data 00,00,00,00,00,00,00,a2,18,a9,0d
380 data 20,d2,ff,a0,04,b9,84,06,20,d2,ff
390 data 88,10,f7,a9,27,8d,53,06,a0,00,20
400 data 89,06,a5,fa,18,69,08,90,02,e6,fb
410 data 85,fa,ce,53,06,10,eb,ca,10,d3,60
420 data 02,80,4c,1b,09,8a,48,98,48,a9,80
430 data 8d,c6,06,a2,07,a0,00,b1,fa,2d,c6
440 data 06,38,d0,01,18,3e,4b,06,c8,c0,08
450 data d0,ef,4e,c6,06,ca,10,e7,a2,07,bd
460 data 4b,06,20,d2,ff,20,d2,ff,a9,00,9d
470 data 4b,06,ca,10,ef,68,a8,68,aa,60,00
480 data 00,a9,00,a0,0c,85,61,84,62,a9,00
490 data a0,20,85,63,84,64,78,a9,33,85,01
500 data a0,04,8c,5a,07,a0,00,a2,00,b1,61
510 data c9,80,90,0b,29,7f,48,a9,ff,8d,5b
520 data 07,4c,fc,06,48,a9,00,8d,5b,07,68
530 data 85,65,48,a9,00,85,66,68,0a,26,66
540 data 0a,26,66,0a,26,66,85,65,a9,00,18
550 data 65,65,85,65,90,02,e6,66,a9,d0,18
560 data 65,66,85,66,a0,00,b1,65,4d,5b,07
570 data 91,63,c8,c0,08,d0,f4,a0,00,e6,61
580 data d0,02,e6,62,a5,63,18,69,08,85,63
590 data 90,02,e6,64,ca,d0,9f,ac,5a,07,88
600 data f0,08,8c,5a,07,a0,00,4c,e5,06,a9
610 data 77,85,01,58,60
```

4.2.2 Hardcopyroutine für Epson-Drucker

Die folgende Routine entspricht der obigen, nur daß durch sie Epson-Drucker angesteuert werden.

```
10 forx=0 to 7:reada$:ifa$="-1"then end
20 a=dec(a$)
30 s=s+a
40 poke  1525 +x+z,a:next
50 read b:if b<>s then 70
60 x=0:z=z+8:s=0:goto 10
70 print"fehler in zeile"peek(66)*256+peek(65):end
80 :
2000 data20,91,94,20,84,9d,e0,00, 870
2010 datad0,03,20,c8,06,20,91,94, 774
2020 data20,84,9d,8a,48,20,91,94, 856
2030 data20,84,9d,a9,00,a0,20,85, 815
2040 datafa,84,fb,68,48,a0,01,20, 1002
2050 databa,ff,a9,00,20,bd,ff,20, 1118
2060 datac0,ff,68,48,aa,20,c9,ff, 1281
2070 dataa9,1b,20,d2,ff,a9,31,20, 943
2080 datad2,ff,20,54,06,a9,1b,20, 815
2090 datad2,ff,a9,32,20,d2,ff,20, 1213
2100 datacc,ff,68,4c,c3,ff,00,00, 1089
2110 data00,00,00,00,00,00,ff,a2, 417
2120 data18,a9,0d,20,d2,ff,a0,05, 868
2130 datab9,84,06,20,d2,ff,88,10, 972
2140 dataf7,a9,27,8d,53,06,a0,00, 845
2150 data20,8a,06,a5,fa,18,69,08, 728
2160 data90,02,e6,fb,85,fa,ce,53, 1299
2170 data06,10,eb,ca,10,d3,60,02, 784
2180 data80,02,2a,1b,09,8a,48,98, 570
2190 data48,a9,80,8d,c7,06,a2,07, 884
2200 dataa0,00,b1,fa,2d,c7,06,38, 893
2210 datad0,01,18,3e,4b,06,c8,c0, 768
2220 data08,d0,ef,4e,c7,06,ca,10, 956
2230 datae7,a2,07,bd,4b,06,20,d2, 912
2240 dataff,20,d2,ff,a9,00,9d,4b, 1153
2250 data06,ca,10,ef,68,a8,68,aa, 1009
2260 data60,00,00,a9,00,a0,0c,85, 570
```

```
2270 data61,84,62,a9,00,a0,20,85, 821
2280 data63,84,64,78,a9,33,85,01, 805
2290 dataa0,04,8c,5b,07,a0,00,a2, 724
2300 data00,b1,61,c9,80,90,0b,29, 799
2310 data7f,48,a9,ff,8d,5c,07,4c, 939
2320 datafd,06,48,a9,00,8d,5c,07, 740
2330 data68,85,65,48,a9,00,85,66, 814
2340 data68,0a,26,66,0a,26,66,0a, 414
2350 data26,66,85,65,a9,00,18,65, 668
2360 data65,85,65,90,02,e6,66,a9, 982
2370 datad0,18,65,66,85,66,a0,00, 830
2380 datab1,65,4d,5c,07,91,63,c8, 898
2390 datac0,08,d0,f4,a0,00,e6,61, 1139
2400 datad0,02,e6,62,a5,63,18,69, 931
2410 data08,85,63,90,02,e6,64,ca, 918
2420 datad0,9f,ac,5b,07,88,f0,08, 1021
2430 data8c,5b,07,a0,00,4c,e6,06, 710
2440 dataa9,77,85,01,58,60,01,00, 607
2450 data00,00,00,00,00,00,00,00, 0,-1
```

Nun ein paar Worte zur Funktionsweise des Programms oder vielmehr der beiden Programme, gerichtet an diejenigen unter Ihnen, die sich für die Assemblerprogrammierung interessieren sollten. Sollten Sie nicht zu diesen Leuten gehören, möchten aber dennoch einmal die Assemblerprogrammierung erlernen, so kann ich an dieser Stelle auf das Maschinensprachebuch zum C16, C116 und Plus/4 verweisen, mit dem Sie wohl den Einstig in die Assemblerprogrammierung spielend schaffen sollten.

Das Programm ist in der Lage, wahlweise Textgrafik oder auch Hiresgrafik auszudrucken. Das besondere bei diesem Programm ist aber, daß sowohl Text als auch Grafik über dieselbe Routine ausgedruckt werden. Das ist deshalb möglich, weil der Text vorher in die hochauflösende Grafik kopiert wird. Dabei wird folgendermaßen vorgegangen: Zunächst wird ermittelt, welches Zeichen sich unter der laufenden Adresse befindet. Anschließend wird überprüft, ob der Bildschirmcode dieses Zeichens oberhalb 128 liegt, also ob dieses Zeichen revers ist, da der Charactergenerator ja nicht, wie das z.B. beim Commodore

64 oder Commodore 128 der Fall ist, die reversen Zeichen beinhaltet, sondern dieses beim C16, C116 und Plus/4, der Videocontroller darstellt.

Aufgrund dessen wird ein Flag (0 oder 255) gesetzt, mit dem später, beim Schreiben in die hochauflösende Grafik, eine Exclusive-Oder-Verknüpfung durchgeführt wird, woraus weiterhin eine korrekte Wiedergabe auch aller reversen Zeichen erfolgt.

Das eigentliche Kopieren geschieht dadurch, daß der Wert des aktuellen Zeichens mit 8 multipliziert wird. Danach wird die Adresse des Chargenerators auf den Wert, der sich durch die vorhergehende Operation ergab, noch dazuaddiert, und wir erhalten einen Zeiger auf das Zeichen, das wir in die hochauflösende Grafik kopieren möchten. Alles weitere stellt nun auch kein Problem mehr da, da uns die Anordnung der Bytes innerhalb der hochauflösenden Grafik, welche uns ja sonst beim schnellen Programmieren mehr hinderte als half, da ja die Anordnung der Bytes im Chargenerator genau dieselbe ist.

Was uns nun noch zu tun bleibt, ist eine Hardcopy des hochauflösenden Bildschirms anzufertigen, und unser Ergebnis kann sich sehen lassen: Eine Hardcopy des Textbildschirms in, man wird es nicht leugnen können, wirklich exzellenter Form, die alle commodorespezifischen Sonderzeichen und Steuerzeichen schwarz auf weiß auf's Papier bringt.

Nun gilt es nur noch, diese Hardcopyroutine anwenden zu können. Dies ist wesentlich einfacher, als Ihnen vielleicht die Funktionsweise der Hardcopyroutine vorgekommen sein sollte. Der Aufruf der Routine geschieht nämlich wie folgt:

```
sys 1536,bildschirmmodus,gerät,logfil
```

Wobei folgende Parameter verwendet werden:

bildschirmmodus:	0, für Textgrafik
	1, für Hiresgrafik

gerät: die Geräteadresse des Druckers (4)
logfil: die logische Filenummer (0-14

4.2.3 Posterhardcopy für Star-Drucker

Anschließend haben wir für Sie noch einen Leckerbissen vorbereitet, sofern Sie über einen Star-Drucker verfügen. Mit den folgenden 2 Routinen sind Sie nämlich in der Lage, Poster von einer Größe von 3*2 Seiten zu erstellen.

```
10 rem ********************************
20 rem ***                          ***
30 rem ***      poster hardcopy     ***
35 rem *** fuer alle star-drucker   ***
40 rem ***   ++++++++++++++++++     ***
50 rem ***                          ***
60 rem ********************************
70 :
80 bl$=chr$(239)
90 wh$=chr$(32)
100 :
110 open 1,4,1
120 print#1,chr$(27)+"a"+chr$(6)+chr$(27)+"e";: rem line spacing s
etzen
130 :
140 for z=0 to 3
150 : for y=0 to 199
160 :   for x=z*80 to z*80+79
170 :   locate x,y
180 :   if rdot(2)=1 then print#1,bl$;:else print#1,wh$;
190 :    draw 1,x,y:next x
200 :  print#1
210 :  next y
220 :  print#1,chr$(12);
230 next z
240 print#1,chr$(12)+chr$(27)+"@"+chr$(7)
```

4.3 Das Abspeichern und Laden von Grafik

Ebenso wie man "seine" Bilder schwarz auf weiß verewigt wissen möchte, will man auch diese Bilder jederzeit im Orginal wieder anschauen. Leider stellt der Commodore C16/116 und Plus/4 hierfür keinen Befehl zur Verfügung. Lediglich mit dem eingebauten TED-Mon, dem Maschinensprachemonitor des C16 (das Maschinensprachebuch zum C16/116 und Plus/4 gibt auch über diesen ausführliche Auskunft) können Sie Grafiken auf Diskette oder Kassette abspeichern, was jedoch recht umständlich geschieht, zumal Sie diese Routinen von BASIC aus sowieso nicht nutzen können. Aber wofür eigenlich besitzen Sie Ihr Grafikbuch? In eifriger und strapaziöser Kleinarbeit haben wir für Sie herausgearbeitet, wozu Commodore selbst keine Muße hatte! Die Screenutilities für die neue Commodoregeneration:

Zunächst wieder das Assemblerlisting:

```
20:   0600                    .opt p,oo
20:   0600                    .sym 4
30:   0600                    *=   1536
31:                 ;
32:   9491          chkcom    =    $9491
33:   9d84          getbyt    =    $9d84
34:   0c00          video     =    3072
41:                 ;
42:   0600 4c 0c 06           jmp  savehgr
43:   0603 4c 41 06           jmp  loadhgr
44:   0606 4c 7c 06           jmp  savelgr
45:   0609 4c a0 06           jmp  loadlgr
46:                 ;
50:   060c a9 00    savehgr   lda  #<8192
50:   060e a0 20              ldy  #>8192
60:   0610 85 f8              sta  $f8
60:   0612 84 f9              sty  $f9
70:                 ;
80:   0614 20 91 94           jsr  chkcom
80:   0617 20 84 9d           jsr  getbyt
80:   061a 8a                 txa
```

```
80:    061b 48                 pha
90:                    ;
100:   061c 20 c9 ff           jsr  $ffc9
100:   061f a9 00              lda  #<8192
100:   0621 20 d2 ff           jsr  $ffd2
100:   0624 a9 20              lda  #>8192
100:   0626 20 d2 ff           jsr  $ffd2
110:                   ;
120:   0629 a2 1f              ldx  #31
130:   062b a0 00              ldy  #0
140:                   ;
150:   062d b1 f8       lab1   lda  ($f8),y
160:   062f 20 d2 ff           jsr  $ffd2
170:   0632 c8                 iny
180:   0633 d0 f8              bne  lab1
190:                   ;
200:   0635 e6 f9              inc  $f9
200:   0637 ca                 dex
210:   0638 10 f3              bpl  lab1
220:                   ;
230:   063a 20 cc ff skip1     jsr  $ffcc
240:   063d 68                 pla
240:   063e 4c c3 ff           jmp  $ffc3
250:                   ;
260:   0641 a9 00       loadhgr lda  #<8192
260:   0643 a0 20              ldy  #>8192
270:   0645 85 f8              sta  $f8
270:   0647 84 f9              sty  $f9
280:                   ;
290:   0649 20 91 94           jsr  chkcom
290:   064c 20 84 9d           jsr  getbyt
290:   064f 8a                 txa
290:   0650 48                 pha
300:                   ;
310:   0651 20 c6 ff           jsr  $ffc6
310:   0654 20 e4 ff           jsr  $ffe4
310:   0657 20 e4 ff           jsr  $ffe4
320:                   ;
330:   065a a2 20              ldx  #32
340:   065c a0 00              ldy  #0
```

```
350:                    ;
360:  065e 8a       lab2    txa
360:  065f 48               pha
360:  0660 98               tya
360:  0661 48               pha
360:  0662 20 e4 ff         jsr  $ffe4
360:  0665 8d c0 06         sta  temp
360:  0668 68               pla
360:  0669 a8               tay
360:  066a 68               pla
360:  066b aa               tax
370:  066c ad c0 06         lda  temp
370:  066f 91 f8            sta  ($f8),y
380:  0671 c8               iny
380:  0672 d0 ea            bne  lab2
390:                    ;
400:  0674 e6 f9            inc  $f9
400:  0676 ca               dex
400:  0677 d0 e5            bne  lab2
410:                    ;
420:  0679 4c 3a 06         jmp  skip1
430:                    ;
440:  067c a9 00    savelgr lda  #<video
440:  067e a0 0c            ldy  #>video
450:  0680 85 f8            sta  $f8
450:  0682 84 f9            sty  $f9
460:                    ;
470:  0684 20 91 94         jsr  chkcom
470:  0687 20 84 9d         jsr  getbyt
470:  068a 8a               txa
470:  068b 48               pha
480:                    ;
490:  068c 20 c9 ff         jsr  $ffc9
490:  068f a9 00            lda  #<3072
490:  0691 20 d2 ff         jsr  $ffd2
490:  0694 a9 0c            lda  #>3072
490:  0696 20 d2 ff         jsr  $ffd2
500:                    ;
510:  0699 a2 04            ldx  #4
510:  069b a0 00            ldy  #0
```

```
510:   069d 4c 2d 06          jmp  lab1
520:                  ;
530:   06a0 a9 00     loadlgr lda  #<video
530:   06a2 a0 0c             ldy  #>video
540:   06a4 85 f8             sta  $f8
540:   06a6 84 f9             sty  $f9
550:                  ;
560:   06a8 20 91 94          jsr  chkcom
560:   06ab 20 84 9d          jsr  getbyt
560:   06ae 8a                txa
560:   06af 48                pha
570:                  ;
580:   06b0 20 c6 ff          jsr  $ffc6
580:   06b3 20 e4 ff          jsr  $ffe4
580:   06b6 20 e4 ff          jsr  $ffe4
590:                  ;
600:   06b9 a2 04             ldx  #4
600:   06bb a0 00             ldy  #0
600:   06bd 4c 5e 06          jmp  lab2
610:                  ;
620:   06c0 00        temp    .byt 0
]0600-06c1
no errors


symboltable:
temp      06c0   loadlgr   06a0   savelgr   067c   lab2      065e
loadhgr   0641   skip1     063a   lab1      062d   savehgr   060c
video     0c00   getbyt    9d84   chkcom    9491
11 symbols defined
```

danach der Basiclader:

```
100 rem ********************************
110 rem ***                            ***
120 rem ***                            ***
130 rem ***   laden und speichern      ***
```

```
140 rem ***  der text- /hgr grafik  ***
150 rem *** ------------------------ ***
160 rem ***                          ***
170 rem ********************************
180 :
190 for x=1536 to 1728
200 read a$:a=dec(a$):poke x,a:p=p+a
210 next x
220 :
230  if  p=25078  then  print  "ok":else  print  "fehler  in  da-
tas!!!":stop
240 :
250 rem assemblerdatas
260 :
270 data 4c,0c,06,4c,41,06,4c,7c,06,4c,a0
280 data 06,a9,00,a0,20,85,f8,84,f9,20,91
290 data 94,20,84,9d,8a,48,20,c9,ff,a9,00
300 data 20,d2,ff,a9,20,20,d2,ff,a2,1f,a0
310 data 00,b1,f8,20,d2,ff,c8,d0,f8,e6,f9
320 data ca,10,f3,20,cc,ff,68,4c,c3,ff,a9
330 data 00,a0,20,85,f8,84,f9,20,91,94,20
340 data 84,9d,8a,48,20,c6,ff,20,e4,ff,20
350 data e4,ff,a2,20,a0,00,8a,48,98,48,20
360 data e4,ff,8d,c0,06,68,a8,68,aa,ad,c0
370 data 06,91,f8,c8,d0,ea,e6,f9,ca,d0,e5
380 data 4c,3a,06,a9,00,a0,0c,85,f8,84,f9
390 data 20,91,94,20,84,9d,8a,48,20,c9,ff
400 data a9,00,20,d2,ff,a9,0c,20,d2,ff,a2
410 data 04,a0,00,4c,2d,06,a9,00,a0,0c,85
420 data f8,84,f9,20,91,94,20,84,9d,8a,48
430 data 20,c6,ff,20,e4,ff,20,e4,ff,a2,04
440 data a0,00,4c,5e,06,00
```

und nun die Erklärung:

sys 1526,lf

Dieser Befehl gibt eine hochauflösende Grafik auf Diskette oder Kassette aus. Zuvor muß jedoch mit dem OPEN-Befehl eine Datei eröffnet werden, auf die die Grafik anschließend abgelegt wird. Das geschieht wie folgt:

```
open lf,gerät,1,"grafikname"
```

```
sys 1529,lf
```

Äquivalent zum vorherigen Befehl funktioniert nun dieser, nur daß er die Grafik nicht speichert, sondern lädt. Vorher muß wieder der OPEN-Befehl folgen, wobei nur die 1 durch eine '0' ersetzt werden muß.

```
sys 1532,lf
```

Dieselben Rituale gelten auch für die Textgrafik. Durch diesen Befehl wird nun die Textgrafik auf Diskette abgespeichert. Auch hier wieder der vorweggestellte OPEN-Befehl, wie bei Befehl Nr. 1.

```
sys 1536,lf
```

Nun folgt das Laden der Textgrafik, welches sich genauso wie das Laden der hochauflösenden Grafik vollzieht, nur mit dem Unterschied, daß nicht die hochauflösende, sondern die Textgrafik verändert wird.

Anhang A

Nützliche Utilities

In diesem Anhang wollen wir Ihnen einige nützliche (insgesamt 3) Utilities vorstellen, also Programme, die Sie bei irgendwelchen Programmierarbeiten unterstützen. Das sind Programme, die auch wir bei unseren Arbeiten an diesem Buch benutzt haben.

Shape-Editor

Fangen wir zunächst mit einem Shape-Editor an. Das ist ein Programm, welches in der Lage ist, Shapes zu editieren, und anschließend, mit einem weiteren kleinen Hilfprogramm, in DATA-Zeilen abzulegen, so daß Sie nach einem solchen Erzeugungsdurchgang dieses Shape wann und wo immer Sie wollen in Ihren Programmen parat haben. Kommen wir zunächst zum Listing:

```
100 COLOR 0,1:COLOR 4,1:COLOR 1,6
110 BA=1024:TB=1319:AT=239
120 :
130 REM --- SHAPE EDIT
140 :
150 SCNCLR
160 GRAPHIC 1
170 FOR X=0 TO 38:FOR Y=0 TO 23
180 LOCATE X+140,Y+90:IF RDOT(2)=1 THEN POKEBA+X+40*Y,42:
    ELSE POKEBA+X+40*Y,46
190 NEXT Y,X:GRAPHIC0
200 :
210 X=0:Y=0:GOTO 400
220 :
230 RL=PEEK(BA+X+40*Y)-128
240 GETKEY A$:POKE BA+X+40*Y,RL
250 IF A$=CHR$(17) THEN Y=Y+1
260 IF A$="L" THEN 530
270 IF A$="S" THEN 450
280 IF A$="E" THEN 630
290 IF A$=CHR$(145) THEN Y=Y-1
300 IF A$=CHR$(29) THEN X=X+1
310 IF A$=CHR$(157) THEN X=X-1
320 IF A$=" " THEN CHAR ,X,Y,"*":DRAW 1,X+140,Y+90:X=X+1
330 IF A$=CHR$(160) THEN CHAR ,X,Y,".":DRAW 0,X+140,Y+90:X=X+1
340 IF A$=CHR$(13) THEN X=0:Y=Y+1
350 IF X<0 THEN Y=Y-1:X=38:ELSE IF X>38THEN Y=Y+1:X=0
360 IF Y<0 THEN Y=0:ELSE IF Y>23 THEN Y=23
370 IF A$="B" THEN GRAPHIC 1:GETKEY A$:GRAPHIC 0
380 IF A$=CHR$(147) THEN GRAPHIC 1,1:GRAPHIC 0:GOTO 150
390 IF A$=CHR$(27) THEN RUN
400 RL=PEEK(BA+X+40*Y)+128
410 POKE BA+X+40*Y,RL:GOTO 230
420 :
430 REM --- SAVE
440 :
450 GOSUB 690
460 OPEN 1,GA,1,F$
470 SSHAPE A$,140,90,179,114
480 PRINT#1,LEN(A$):FORI=1 TO LEN(A$):
    PRINT#1,ASC(MID$(A$,I,1)):NEXT
```

```
490 CLOSE 1:RUN
500 :
510 REM --- LOAD
520 :
530 GOSUB 690
540 OPEN 1,GA,0,F$:A$=""
550 INPUT#1,A:FORI=1 TO A:INPUT#1,A2:A$=A$+CHR$(A2):NEXT
560 CLOSE 1
570 GRAPHIC 1,1
580 GSHAPE A$,140,90
590 GRAPHIC 0:RUN
600 :
610 REM --- ENDE
620 :
630 SCNCLR:PRINT"◇IRKLICH ?":GETKEY A$:IF A$="J"
    THEN 640:ELSE RUN
640 PRINTCHR$(147);"DELETE -1170"
650 POKE TB,19:POKE TB+1,13:POKE AT,2
660 :
670 REM --- I/O OPERATION
680 :
690 SCNCLR:PRINT "■D■ISK ODER ■T■APE ? >";
700 GETKEY A$:IF A$<>"D"AND A$<>"T" THEN 700
710 PRINTA$:IF A$="D" THEN GA=8:ELSE GA=1
720 INPUT"FILENAME";F$
730 RETURN
```

Nun zur Erklärung: Nachdem Sie das Programm geladen haben, starten Sie es mit RUN. Danach tut sich erst einmal eine Weile gar nichts. Anschließend erscheint ein Editorfeld, auf dem sich ein Cursor befindet, den Sie durch die Cursorsteuerungstasten, ganz so wie Sie es gewohnt sind, bewegen können.

Punkte setzen können Sie, indem Sie die Leertaste drücken. Sollte unter dem Cursor schon ein Punkt gesetzt sein, so wird beim Betätigen der Leertaste dieser gelöscht.

Mit den Tasten "L" und "S" können Sie Ihr Shape von Diskette oder Kassette einladen oder abspeichern. Der Computer fragt Sie dann nach dem Filenamen und führt die gewünschte Operation aus.

Die Taste "B" führt Sie in die hochauflösende Grafik, wo Sie Ihr gerade editiertes Shape im Originalzustand betrachten können. Zurück zum Editierfeld gelangen Sie mit jeder beliebigen Taste.

Ein Druck auf die "RETURN"-Taste setzt den Cursor an den Anfang der nächsten Zeile.

Um das Shape zu löschen, bedienen Sie sich der "CLR"-Taste. Danach dauert es einen Moment, bis das Editorfeld wieder auf dem Bildschirm erscheint.

Schließlich gibt es noch die ESC-Taste, die sozusagen der Reset-Taster des gesamten Systems ist. Wird sie gedrückt, so kommt dieses einem Neustart des Programms gleich.

Nun will ich noch einmal, der besseren Übersicht halber, alle Funktionen unseres Shape-Editors tabellarisch zusammenfassen:

Taste	Funktionsbeschreibung
ESC	Neustart des Programms
CRSR-Tasten	Cursorsteuerung
SPACE	Löschen/Setzen eines Punktes
"B"	Shape im Original berachten
"L"	Laden eines Shapes
"S"	Speichern eines Shapes
"CLR"	Löschen des Shapes
"RETURN"	Setzen des Cursors an den Anfang der nächsten Spalte

Shape-Generator

Anschließend gleich noch ein Programm. Dieses Programm erzeugt nämlich aus einem Shape, dessen Filenamen Sie beim Start dieses Programms angeben müssen, Data-Zeilen, in welchen das Shape abgespeichert wird. So können Sie jederzeit Ihr Shape verwenden, ohne vorher umständlich mit GSHAPE und SSHAPE hantieren zu müssen:

```
10 REM **********************************
20 REM ***                            ***
30 REM ** SHAPE IN DATA-ZEILEN NEHMEN **
40 REM **                              **
50 REM **                              **
60 REM **       FUER SHAPOMAT 16       **
70 REM ***                            ***
80 REM **********************************
85 :
86 SCNCLR
90 PRINT"SHAPES  VON SHAPOMAT 16 WERDEN IN DATA-"
92 PRINT"ZEILEN GENERIERT. SHAPE WIRD GELADEN"
93 :
94 CHAR1,2,5,"":INPUT"FILENAME";F$
95 CHAR1,2,7,"■D■ISK ODER ■T■APE ? >"
96 :
100 GETKEY A$:IF A$<>"D"AND A$<>"T" THEN 100
105 :
110 PRINTA$:IF A$="D" THEN GA=8:ELSE GA=1
130 :
140 REM --- LOAD
150 :
170 OPEN 1,GA,0,F$:A$=""
180 INPUT#1,A:FORI=1 TO A:INPUT#1,A2:A$=A$+CHR$(A2):NEXT
190 CLOSE 1
200 GRAPHIC 1,1
210 GSHAPE A$,140,90
215 GRAPHIC CLR
220 :
230 REM --- EINGABEN
240 :
250 SCNCLR
260 INPUT"STARTZEILENNUMMER :";SZ
265 IF SZ<500 THEN 260
280 :
290 INPUT"SCHRITTWEITE      :";SW
300 IF SZ+130*SW >65535 THEN PRINT"NICHT MOEGLICH!!":GOTO 260
301 :
302 REM --- UEBERNEHMEN IN DATA-ZEILEN
310 :
315 PRINTCHR$(147)
316 :
320 FOR I=0 TO 7
321 PRINTSZ;"DATA";
325 :FOR T=1 TO 16
330 : B=ASC(MID$(A$,I*16+T,1))
340 : PRINT RIGHT$(HEX$(B),2)+",";
350 :NEXT
360 PRINT CHR$(20):SZ=SZ+SW
370 NEXT
380 :
390 B=ASC(MID$(A$,129,1))
395 PRINTSZ;"DATA"+RIGHT$(HEX$(B),2):REM LETZTER WERT
400 POKE 842,19
410 FORI=1 TO10:POKE 842+I,13:NEXT
420 POKE 208,10:END
```

Datawandler

Das nun folgende Programm ist so einfach wie nützlich. Es generiert nämlich aus Assemblerprogrammen Data-Zeilen auf sehr komfortable Weise. Das Programm an sich ist selbsterklärend, ich will an dieser Stelle nur ein paar Worte zu den Funktionen des Programms sagen, die es erzeugt:

Dieses Programm, das die Einleseroutine übrigens schon enthält (!!), ist in der Lage, Fehler, die eventuelle Abtipper dieses Programms gemacht haben, zu erkennen. Es meldet den Fehler dann in der entsprechenden Zeile. Alles was Sie dann noch zu tun haben, ist, die entsprechende Zeile zu überprüfen und zu berichtigen.

Nun das Programm:

```
10 REM ------  DATA-GENERATOR  ------
12 :
14 INPUT "STARTADR. MC-PRG  ";SA
16 :
18 INPUT "ENDADR. MC-PRG    ";EA
20 :
22 IF SA>EA THEN 14
24 :
26 INPUT "ERSTE ZEILENNUMMER";ZN
28 :
30 :::   IF ZN<=100 THEN 26
32 :
34 INPUT "SCHRITTWEITE      ";SW
36 :
38 SCNCLR
40 :
42 PRINT ZN;"DATA ";
44 :
46 FOR I=SA TO SA+10
48 :   A =PEEK(I)
50 :     A$=HEX$(A)
```

```
52 :   A$=RIGHT$(A$,2)
54 :  PRINT A$;",";
56 NEXT I
58 :
60 PRINT CHR$(20)
62 :
64 PRINT "ZN=";ZN;":SW=";SW;":SA=";SA;":EA=";EA
66 :
68 PRINT : PRINT
70 PRINT "GOTO 86"
72 :  POKE 1319,19
74 :   FOR I=1320 TO 1320+3
76 :    POKE I,13
78 :   NEXT I
80 :  POKE 239,4
82 END
84 :
86 SA=SA+11:ZN=ZN+SW
88 :
90 IF SA<EA THEN 38
92 :
94 PRINTCHR$(147)+"DELETE-100"
96 PRINT:PRINT:PRINT"DATA-ZEILEN WURDEN GENERIERT"
98 :
100 POKE 1319,19:POKE1320,13:POKE239,2:END
```

Anhang B

Tabellierung aller Control-Funktionen

CTRL -K- & CHR$(11):
Die Umschaltung Klein/Groß und Groß/Grafik wird verboten.

CTRL -L- & CHR$(12):
Die Umschaltung zwischen den beiden Alternativzeichensätzen wird wieder erlaubt.

CTRL -M- & Chr$(13):
Der Cursor wird an den Anfang der nächsten Zeile gesetzt (Carriage-Return).

CTRL -N- & Chr$(14):
Es wird von Groß/Grafik- auf den Klein/Groß-Zeichensatz umgeschaltet

CTRL -[- & CHR$(27):
entspricht Escape.

Anhang C

Tabellierung aller ESC-Funktionen

ESC A Der automatische Einfügemodus wird eingeschaltet, d.h. werden Zeichen eingegeben, so werden diese in den bestehenden Text eingefügt und dieser nicht überschrieben.

ESC B Durch diese Tastenkombination wird die momentane Cursorposition als rechte, untere Ecke eines Bildschirmfensters definiert.

ESC C Der durch ESC A aktivierte automatische Einfügemodus wird wieder ausgeschaltet, also der Überschreibmodus aktiviert.

ESC D Die aktuelle Bildschirmzeile wird gelöscht und der Bildschirmbereich unter der momentanen Cursorposition um eine Zeile nach oben verschoben.

ESC I Fügt oberhalb der Cursorposition eine neue Zeile ein und rollt das Bildschirmfenster um eine Zeile nach unten.

ESC J Der Cursor wird an den Anfang der aktuellen Bildschirmzeile gesetzt.

ESC K Der Cursor wird hinter das letzte Zeichen der aktuellen Zeile gesetzt.

ESC L Das Bildschirmscrolling wird erlaubt.

ESC M Das Bildschirmscrolling wird verboten. Bei Überschreitung der untersten Zeile wird ein Cursor-Home ausgeführt.

ESC N Hiermit wird der Bildschirm wieder auf Normalgröße gebracht und gelöscht (siehe ESC R).

ESC O Diese Tastenkombination besitzt die gleiche Funktion wie die weiter oben beschriebene Tastenkombination ESC-ESC.

ESC P Die momentane Zeile wird vom Zeilenanfang bis zur Cursorposition gelöscht.

ESC Q Die momentane Zeile wird von der Cursorposition bis zum Zeilenende gelöscht.

ESC R Verkleinert den Bildschirm auf 38 Spalten. Dieses kann von Vorteil sein, wenn Sie oft mit Soft-Scrolling arbeiten.

ESC T Es wird die rechte, obere Ecke des Windows bestimmt.

Ein Tip zum Window-Handling: Durch die Tastenkombination "{HOME}{HOME}", wir das aktuelle Window gelöscht. Selbstverständlich kann die Ausgabe auch durch das PRINT-Statement erfolgen.

Der WINDOW-Befehl, den Sie vielleicht vom C128 her kennen, existiert zwar nicht auf diesem Computertyp, kann jedoch durch ein kleines BASIC-Programm, welches später dann in Ihren eigenen Programmen als Unterroutine aufgerufen werden kann, diesen ersetzen:

```
50000 rem ******************************
50010 rem *** window-simulation      ***
50020 rem *** Param: x1,y1,x2,y2     ***
50030 rem ******************************
50040 :
50050 char ,x1,y1,chr$(27)+"t"
50060 char ,x2,y2,chr$(27)+"b"
50070 return
```

Und noch ein weiterer Tip: Der CHAR-Befehl kümmert sich leider nicht an die vorgegebene Window-Definition. Dieses Window-Handling funktioniert nur, wenn Bereichsüberschreitung des Cursors durch die betriebsinterne BSOUT-Routine übernommen wird, wie das z. B. im BASIC-PRINT-Befehl geschieht. Ebenso wird der Cursor niemals relativ zu der angegebenen Window-Definition, sondern immer aus der normalen Bildschirmgröße heraus berechnet.

Anhang D

ASCII- und Bildschirmcodetabellen

Chr$ & Poke Kodes

Dez	Hex	POKE- Zeichen Gross	POKE- Zeichen Klein	CHR$- Zeichen Gross	CHR$- Zeichen Klein
000	$00	@	@		Keine Funktion
001	$01	A	a		Keine Funktion
002	$02	B	b		Keine Funktion
003	$03	C	c		Keine Funktion
004	$04	D	d		Keine Funktion
005	$05	E	e		Weiss
006	$06	F	f		Keine Funktion
007	$07	G	g		Keine Funktion
008	$08	H	h		Shift Commodore verboten
009	$09	I	i		Shift Commodore erlauben
010	$0a	J	j		Keine Funktion
011	$0b	K	k		Keine Funktion
012	$0c	L	l		Keine Funktion
013	$0d	M	m		Return
014	$0e	N	n		Kleinbuchstaben
015	$0f	O	o		Keine Funktion
016	$10	P	p		Keine Funktion
017	$11	Q	q		Cursor abwaerts
018	$12	R	r		Revers anschalten
019	$13	S	s		Cursor Home
020	$14	T	t		Delete
021	$15	U	u		Keine Funktion
022	$16	V	v		Keine Funktion
023	$17	W	w		Keine Funktion
024	$18	X	x		Keine Funktion
025	$19	Y	y		Keine Funktion
026	$1a	Z	z		Keine Funktion
027	$1b	[	[		Escape
028	$1c	£	£		Rot
029	$1d	]	]		Cursor rechts
030	$1e	↑	↑		Gruen
031	$1f	←	←		Dunkelblau
032	$20				Space
033	$21	!	!	!	!
034	$22			"	"
035	$23	#	#	#	#
036	$24	$	$	$	$
037	$25	%	%	%	%
038	$26	&	&	&	&
039	$27	'	'	'	'
040	$28	(	(	(	(
041	$29	)	)	)	)
042	$2a	*	*	*	*
043	$2b	+	+	+	+
044	$2c	,	,	,	,
045	$2d	-	-	-	-
046	$2e	.	.	.	.
047	$2f	/	/	/	/
048	$30	0	0	0	0
049	$31	1	1	1	1
050	$32	2	2	2	2
051	$33	3	3	3	3
052	$34	4	4	4	4
053	$35	5	5	5	5
054	$36	6	6	6	6
055	$37	7	7	7	7
056	$38	8	8	8	8
057	$39	9	9	9	9
058	$3a	:	:	:	:
059	$3b	;	;	;	;

060	$3c	
061	$3d	
062	$3e	
063	$3f	
064	$40	
065	$41	
066	$42	
067	$43	
068	$44	
069	$45	
070	$46	
071	$47	
072	$48	
073	$49	
074	$4a	
075	$4b	
076	$4c	
077	$4d	
078	$4e	
079	$4f	
080	$50	
081	$51	
082	$52	
083	$53	
084	$54	
085	$55	
086	$56	
087	$57	
088	$58	
089	$59	
090	$5a	
091	$5b	
092	$5c	
093	$5d	
094	$5e	
095	$5f	
096	$60	
097	$61	
098	$62	
099	$63	
100	$64	
101	$65	
102	$66	
103	$67	
104	$68	
105	$69	
106	$6a	
107	$6b	
108	$6c	
109	$6d	
110	$6e	
111	$6f	
112	$70	
113	$71	
114	$72	
115	$73	
116	$74	
117	$75	
118	$76	
119	$77	
120	$78	
121	$79	
122	$7a	
123	$7b	
124	$7c	
125	$7d	
126	$7e	
127	$7f	
128	$80	Keine Funktion
129	$81	Keine Funktion
130	$82	Keine Funktion
131	$83	Keine Funktion

132	$84					Keine Funktion
133	$85					Weiss
134	$86					Keine Funktion
135	$87					Keine Funktion
136	$88					Shift Commodore verboten
137	$89					Shift Commodore erlauben
138	$8a					Keine Funktion
139	$8b					Keine Funktion
140	$8c					Keine Funktion
141	$8d					Return
142	$8e					Kleinbuchstaben
143	$8f					Keine Funktion
144	$90					Keine Funktion
145	$91					Cursor abwaerts
146	$92					Revers anschalten
147	$93					Cursor Home
148	$94					Delete
149	$95					Keine Funktion
150	$96					Keine Funktion
151	$97					Keine Funktion
152	$98					Keine Funktion
153	$99					Keine Funktion
154	$9a					Keine Funktion
155	$9b					Escape
156	$9c					Rot
157	$9d					Cursor rechts
158	$9e					Gruen
159	$9f					Dunkelblau
160	$a0					Space
161	$a1			!	!	
162	$a2			"	"	
163	$a3			#	#	
164	$a4			$	$	
165	$a5			%	%	
166	$a6			&	&	
167	$a7			'	'	
168	$a8			(	(	
169	$a9			)	)	
170	$aa			*	*	
171	$ab			+	+	
172	$ac			,	,	
173	$ad			-	-	
174	$ae			.	.	
175	$af			/	/	
176	$b0			0	0	
177	$b1			1	1	
178	$b2			2	2	
179	$b3			3	3	
180	$b4			4	4	
181	$b5			5	5	
182	$b6			6	6	
183	$b7			7	7	
184	$b8			8	8	
185	$b9			9	9	
186	$ba			:	:	
187	$bb			;	;	
188	$bc			<	<	
189	$bd			=	=	
190	$be			>	>	
191	$bf			?	?	
192	$c0			@	@	
193	$c1			A	a	
194	$c2			B	b	
195	$c3			C	c	
196	$c4			D	d	
197	$c5			E	e	
198	$c6			F	f	
199	$c7			G	g	
200	$c8			H	h	
201	$c9			I	i	
202	$ca			J	j	
203	$cb			K	k	

2Ø4	$cc			L	l
2Ø5	$cd			M	m
2Ø6	$ce			N	n
2Ø7	$cf			O	o
2Ø8	$dØ			P	p
2Ø9	$d1			Q	q
21Ø	$d2			R	r
211	$d3			S	s
212	$d4			T	t
213	$d5			U	u
214	$d6			V	v
215	$d7			W	w
216	$d8			X	x
217	$d9			Y	y
218	$da			Z	z
219	$db			[	[
22Ø	$dc			£	£
221	$dd			]	]
222	$de			↑	↑
223	$df			←	←
224	$eØ				
225	$e1				A
226	$e2				B
227	$e3				C
228	$e4				D
229	$e5				E
23Ø	$e6				F
231	$e7				G
232	$e8				H
233	$e9				I
234	$ea				J
235	$eb				K
236	$ec				L
237	$ed				M
238	$ee				N
239	$ef				O
24Ø	$fØ				P
241	$f1				Q
242	$f2				R
243	$f3				S
244	$f4				T
245	$f5				U
246	$f6				V
247	$f7				W
248	$f8				X
249	$f9				Y
25Ø	$fa				Z
251	$fb			+	+
252	$fc				
253	$fd				
254	$fe				
255	$ff				

Anhang E

Stichwortverzeichnis

F

G

H

K

L

M

P

R

S

T

V

W

Z

Anhang F

Quellennachweise

Das große Grafikbuch zum Commodore 128
Durben, Löffelmann, Plenge, Vüllers
ISBN 3-89011-154-8
Data Becker Verlag

Das Grafikbuch zum Commodore 64
Axel Plenge
ISBN 3-89011-011-8
Data Becker Verlag

Commodore 128 Intern
Gerits, Schieb, Thrun
ISBN 3-89011-098-3
Data Becker Verlag

Einführung in CAD mit dem Commodore 64
Heift
ISBN 3-89011-067-3
Data Becker Verlag

64 Intern
Brückmann, Englisch, Felt, Gelfand, Gerits, Krsnik
ISBN 3-89011-000-2
Data Becker Verlag

Commodore C16 Tips & Tricks
Baloui
ISBN 3-89011-168-8
Data Becker Verlag

Bücher zu C16, C116 und Plus/4

Machen Sie mehr aus Ihrem Rechner! Anhand vieler Beispiele führt der Autor klar verständlich in den Befehlssatz der Rechner C16, C116 und Plus/4 ein. Die außerordentlich nützlichen Editor-Funktionen werden vorgestellt.
Das Buch wird abgerundet durch die Erstellung eines praktischen Anwender-Programmes zur Dateiverwaltung. Die Stärken der Programmierung in BASIC werden im Überblick verdeutlicht. Für Einsteiger und Fortgeschrittene gleichermaßen interessant.

Baloui
Das große BASIC-Buch C16, C116, Plus/4
ca. 250 Seiten, DM 29,–
ISBN 3-89011-204-8

Bücher zu C16, C116 und Plus/4

Viele Computerbenutzer, die bislang die Hemmschwelle zur Maschinensprache nicht überwunden haben, erhalten hier die perfekte Einführung in den Befehlssatz des 7501-Mikroprozessors. Neben leichtverständlicher Beschreibung aller 7501-Befehle werden viele Betriebssystemroutinen unter die Lupe genommen.

Aus dem Inhalt:

- Von BASIC zu Maschinensprache
- Der Maschinensprachemonitor
- Der 7501-Prozessor und sein Befehlssatz
- Ein- und Ausgabe von Zeichen in Maschinensprache
- Einbinden von Maschinenprogrammen in BASIC-Programme
- Der BASIC-Lader
- Routinen des Betriebssystems richtig genutzt
- Die Vektoren des Betriebssystems
- Interruptprogrammierung
- Beispielprogramme mit trickreichen Utilities
- Zeropage-, Befehlscode- und Umrechnungstabellen

Vüllers
C16, C116, Plus/4 Maschinensprache
ca. 300 Seiten, DM 29,–
ISBN 3-89011-206-4

Bücher zu C16, C116 und Plus/4

C16 Tips und Tricks bietet eine hochinteressante Sammlung von Anregungen, Ideen und fertigen Lösungen zur Programmierung und Anwendung Ihres C16/116. Eine wahre Fundgrube für jeden, der auf dem Commodore 16 eigene Programme schreiben will!

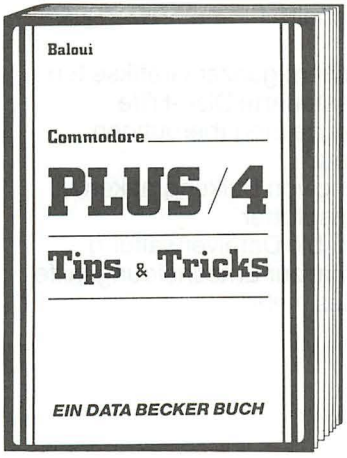
Aus dem Inhalt:

- Hinweise zur Eingabe der Programme
- Alle Programme für Kassetten- und Diskettenbetrieb
- Anwenderprogramme aus den Bereichen Unterhaltung, Grafik, Text- und Dateiverarbeitung, Mathematik
- Viele Utilities, wie Hardcopy auf Drucker, REM-Killer, Mergen von einzelnen Programmteilen, Shape-Editor
- Etikettendruck
- Lottozahlen
- Datumsberechnung
- Die wichtigsten Tips & Tricks
- Von BASIC zu Maschinensprache
- Wichtige Zeropageadressen
- Betriebssystemroutinen
- Routinen des BASIC-Interpreters

Baloui
C16 Tips & Tricks
201 Seiten, DM 29,–
ISBN 3-89011-168-8

Bücher zu C16, C116 und Plus/4

Plus/4 Tips & Tricks ist eine Sammlung von Anregungen, Ideen und fertigen Lösungen zur Programmierung und Anwendung Ihres Plus/4.
Alles, was Sie brauchen, um eigene Programme zu schreiben.



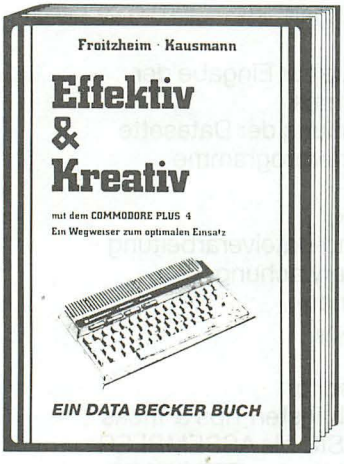
Aus dem Inhalt:

- Hinweise zur Eingabe der Programme
- Verwendung der Datasette
- Anwenderprogramme
  Spiele
  Graphik
  Text- und Dateiverarbeitung
  Kfz-Überwachung
  Mathematik
  Hardcopy
  Merge
  Shapeeditor
- Die wichtigsten Tips & Tricks
- Von BASIC bis ASSEMBLER
- Die wichtigsten ZEROPAGE-ADRESSEN
- Routinen des Betriebssystems und des BASIC-Interpreters
- Tips & Tricks für Fortgeschrittene

Baloui
Plus/4 Tips & Tricks
222 Seiten, DM 29,–
ISBN 3-89011-203-X

Bücher zu C16, C116 und Plus/4

Effektiv und kreativ mit dem Plus/4 und seiner eingebauten Software zu arbeiten ermöglicht dieses Buch. Im ersten Teil werden zahlreiche Ideen, Anwendungen und praktische Nutzungsmöglichkeiten der integrierten Software beschrieben, der zweite Teil enthält viele interessante Programmiertips und Tricks und eine Reihe kompletter Anwendungsprogramme.



Aus dem Inhalt:

- Abspeichern ganzer Grafikseiten
- Menügesteuerte Disk-Hilfe
- Verbesserte Eingaberoutinen
- Listschutz
- Erstellung von Balkengrafiken
- Funktionsplotter
- Komfortable Dateiverwaltung
- Programm zur Entscheidungshilfe und vieles mehr

Froitzheim, Kausmann
Effektiv und kreativ mit dem Plus/4
244 Seiten, DM 49,–
ISBN 3-89011-073-8

DAS STEHT DRIN:

Ein Team von Grafikspezialisten deckt wirklich alle Geheimnisse des C16, C116 und Plus/4 auf. Von den grundlegenden Grafikbefehlen bis hin zu nützlichen Utilities enthält dieses Buch alles, was Sie schon immer über Computergrafik auf Ihrem Rechner wissen wollten.

Aus dem Inhalt:

- theoretische Grundlagen
- Linienberechnung in Assembler
- Einführung in die Hardware
- hochauflösende Grafik
- komfortabler Charactereditor
- Computer Aided Design
- komplettes Malprogramm
- ein CAD-System
- Statistik
- Funktionsplotter
- Hardcopyroutinen
- Posterhardcopy auf allen Star-Druckern
- Shape-Editor

UND GESCHRIEBEN HABEN DIESES BUCH:

Axel Plenge und Klaus Löffelmann, beide bekannte Fachbuchautoren und hochkarätige Grafikspezialisten, haben in diesem Buch ihr komplexes Know-how zur Verfügung gestellt.

ISBN 3-89011-205-6