REISE DURCH DIE WU

TEIL 4

**Nachdem in den ersten drei Folgen unseres Grafikkurses all
Die hochauflösende Grafik des Commodore 64, die von un
men einzusetzen. Die nötigen Hilfsmittel, wie zum Beispiel ei**

Sie können bisher Zeichen im Mehrfarbenmodus und (aber nicht gleichzeitig) mit veränderten Hintergrundfarben darstellen. Das Prinzip der Bit-Map ist Ihnen vertraut und Sie wissen, wie man dem Computer sagt, daß er nun seine Bildschirminformationen aus dieser Bit-Map holt. Sie können in diesem Modus die Farben bestimmen und schließlich auch Punkte exakt in die Bit-Map setzen. Wenn Ihnen der Inhalt der Bit-Map nicht mehr gefällt, können Sie sie löschen.

In dieser vierten Folge werden wir lernen, wie man die Speicher des Commodore 64 für Grafikanwendungen umkrempelt. Wir werden »Dornröschen« in mehreren Farben erleben und schließen diesen Teil der hochauflösenden Grafik mit einer kleinen Unterprogrammbibliothek ab. Der einleitenden Worte sind genug gesagt, Dornröschen wartet.

Wir krempeln den Commodore 64 um: Speicherveränderungen für hochauflösende Grafik

Wenn Sie als stolzer Besitzer eines C 64 früher auch mal ebenso stolzer Besitzer eines VC 20 waren, dann ist Ihnen sicherlich in wehmütiger Erinnerung, was Sie sehen, wenn Sie durch die POKE-Kommandos
POKE 51,255:POKE 52, 31:POKE 55,255:POKE 56,31
in der letzten Folge die Bit-Map vor dem Überschreiben durch Basic geschützt haben und dann mal mit PRINT FRE (1) nach dem freien Basic-Speicher fragten: Da zeigte sich: 6144 (ohne Programm)!

Geht das Ringen um jedes Byte nun wieder los? Wie soll denn in diese 6 KByte ein besseres Spiel mit Hochauflösungsgrafik — ganz zu schweigen von anspruchsvolleren Programmen, zum Beispiel einer Kurvendiskussion — hineinpassen? Nun, keine Sorge: Wozu haben wir im C 64 denn 64 KByte RAM? Wir müßten nur wissen, wie wir sie nutzen können.

Dazu sehen wir uns nochmal den VIC-II-Chip an. Im Gegensatz zur CPU (unserem Prozessor 6510), die über 16 Adressenleitungen verfügen kann, stehen beim VIC-II-Chip lediglich 14 zur Disposition. Während man also mit 16 Leitungen alle Adressen
von 0 bis 1111 1111 1111 1111 = 65535
ansteuern kann, ist bei 14 Leitungen ein Maximum
von 11 1111 1111 1111 = 16383 Adressen möglich, also 16 KByte.

Der gesamte Speicherraum des C 64 ist in vier solche 16 KByte-Blöcke aufteilbar und wie wir wissen, blickt der VIC-II-Chip im Normalfall auf den ersten 16 KByte-Block (siehe Bild 1). Nun kann man dem VIC-II-Chip mitteilen, daß er seine Aufmerksamkeit auf die anderen Speicherviertel richten möge. Das erfordert die Mitarbeit des »Portiers« CIA 2 (siehe Folge 1). Er hat im Gebäude 56576 zwei Zimmer (Bits 0 und 1), aus denen dem VIC-II-Chip die Anweisungen gegeben werden, um welches Viertel unseres Speichers er sich kümmern soll. Auf welche Weise diese Bits den VIC-II-Chip-Zugriff regeln, sehen Sie aus Tabelle 1.

56576 BITS 10	BITWERT DEZIMAL	AB-SCHNITT	SPEICHERBEREICH von — bis	BEMERKUNGEN
11	3	0	0 — 16383	EINSCHALT-ZUSTAND VON 4096 — 8191 ZEICHEN-SPIEGELBILDER
10	2	1	16384—32767	KEINE ZEICHEN VERFÜGBAR VON 36864 — 40959 ZEICHEN-SPIEGELBILDER
01	1	2	32768—49151	VON 40960—49151 BASIC-ROM
00	0	3	49152 — 65535	KEINE ZEICHEN VERFÜGBAR VON 53258-65535 I/O, ZEICHEN-ROM BETRIEBSSYSTEM

Tabelle 1. Die Bits 1 und 0 von Speicherstelle 56576 regeln den Zugriff des VIC-II-Chips auf den Speicher

Commodore empfiehlt nun noch sicherzustellen, daß die zu dieser Abschnittsauswahl gehörigen Bits des Datenrichtungsregisters Port A im CIA 2 (Speicherstelle 56578) auf 1, also auf Ausgabe, gestellt werden. In allen Programmen, die ich bisher verwendet habe, wäre das eigentlich nicht nötig gewesen, jedenfalls habe ich nichts bemerkt, als ich das nicht getan habe.

Trotzdem gebe ich hier diese Empfehlung weiter, falls in Ihren Programmen diese Maßnahme notwendig wird. Falls Sie vergessen haben sollten (Folge 2), wie man Bits setzt oder löscht, hier die nötigen Programmzeilen dazu:
20 POKE 56576, (PEEK(56576) AND 252) OR I
30 POKE 56578, PEEK(56578) OR 3
I ist dabei der in Tabelle 1 gezeigte Dezimalwert der Bits 0 und 1.

Der VIC-II-Chip managt auch den Bildschirm. Im Einschaltzustand packt er den Bildschirmspeicher — wie wir schon wissen —

BIT-PAAR	FARBQUELLE
00	SPEICHER 53281, HINTERGRUNDREG. Nr.0
01	BITS 4-7 } DES VIDEO-RAM
10	BITS 0-3 } DES VIDEO-RAM
11	BILDSCHIRMFARBSPEICHER

▲ Tabelle 3. Herkunft der Farben bei Bit-Map-Mehrfarbenmodus in Abhängigkeit von den Bit-Paaren

in den Bereich 1024
und 2023. Wenn wir nun
einen anderen 16 KByte-Abschnitt wählen, legt er den Bildschirm an die entsprechende Stelle dieses Abschnittes, also:
In Abschnitt 0: Bildschirm von 1024 bis 2023
in Abschnitt 1: Bildschirm von 16384

NDERWELT DER GRAFIK

e Grundlagen geschaffen wurden, nähern wir uns endlich mit Riesenschritten unserem Ziel: s endlich aus ihrem Dornröschenschlaf geweckt wurde, gezielt in unseren Basic-Programne kleine Bibliothek von Grafik-Unterprogrammen, werden in dieser Folge vorgestellt.

+ 1024 = 17408
bis 16384 + 2023 = 18407
in Abschnitt 2: Bildschirm von 32768 + 1024 = 33792
bis 32768 + 2023 = 34791
in Abschnitt 3: Bildschirm von 49152 + 1024 = 50176
bis 49152 + 2023 = 51175.

Damit brauchen wir uns aber nicht zufrieden geben.

Multivisio — unter 64 Bildschirm wählen

Im 16 KByte-Speicherabschnitt hat der Bildschirmspeicher ja 16mal Platz und wir können ihn ohne weiteres an eine andere Stelle schieben. Damit kehren wir nochmal zur schon oft besungenen Speicherstelle 53272 zurück. Die Bits 4 bis 7 geben dem VIC-II-Chip den Ort des Bildschirmspeichers an. Durch Verändern dieser 4 Bits können wir tatsächlich die 16 Bildschirme pro 16 KByte-Abschnitt einrichten. Der Zusammenhang zwischen den Bits 4 bis 7 von Adresse 53272 und dem Ort des Bildschirmspeichers im Abschnitt 0 (und entsprechend parallelverschoben in den anderen Abschnitten) ist in Tabelle 2 gezeigt.

Um die entsprechende Bitanordnung zu erreichen, müssen wir also eingeben:

```
40 POKE 53272, (PEEK(53272) AND 15) OR W
```

Dabei ist W der Dezimalwert der Bits 4 bis 7 aus Tabelle 2.

Das Betriebssystem muß auch noch erfahren, daß der Bildschirmspeicher verlegt worden ist. Man kann es ihm mitteilen, indem man die Pagenummer der Bildschirmstartadresse in Speicherstelle 648 einPOKEd. Also ist zum Beispiel der normale Inhalt von Speicher 648: 1024/256 = 4. Auf Page 4 beginnt der Bildschirm ja im Einschaltzustand. Die Pagenummer ergibt sich aus I und W (siehe Tabelle 1 und 2) durch folgende Rechnung:

```
50 P = (W/16*1024 + 16384*(3-I))/256
```

und wird dann eingePOKEd:

```
55 POKE 648,P
```

So! Jetzt können wir theoretisch 64 Bildschirme erzeugen. Um uns das in der Praxis mal anzusehen, ergänzen wir die bisher verwendeten Programmzeilen (die Sie hoffentlich noch nicht mit RUN gestartet haben, das wäre nämlich mit etwas Glück eine gute Methode, den Rechner abstürzen zu lassen!) noch um folgendes:

```
10 INPUT"I,W";I,W:PRINT CHR$(147)
60 PRINT CHR$(147)I,W:END
65 PRINT CHR$(147)
70 POKE 56576,151:POKE 56578,63:POKE 53272,21:POKE 648,4
80 I=3:W=16:PRINT CHR$(147)I,W
90 END
```

Bevor Sie das starten, sollten Sie bitten daran denken, daß einige I,W-Kombinationen den Computer zum totalen Black-Out führen: Zum Beispiel I=3,W=0 legt den Bildschirmstart direkt in die Zeropage, ist also nicht empfehlenswert. I=3,W=32 zerstört unser Programm, das genau bei 2048 anfängt. Am besten speichern Sie das Programm vor dem Starten ab.

Jeweils nach RUN und Eingabe der Werte I und W, zum Beispiel I=3, W=48, wird der Bildschirm gelöscht und in der obersten Zeile wird der I- und der W-Wert angegeben. Danach meldet sich READY und ein Cursor. Jetzt befinden wir uns im neuen Bildschirm (unser Beispiel also 3072 bis 4071) und können damit herumexperimentieren.

SPEICHERSTELLE 53272 BITS: 7654	DEZIMALWERT (BITS 0-3 als 0 angenommen) W	BILDSCHIRMSTARTADRESSE (IM ABSCHNITT 0) DEZIMAL	HEX
0000	0	0	0
0001	16	1024	400
0010	32	2048	800
0011	48	3072	C00
0100	64	4096	1000
0101	80	5120	1400
0110	96	6144	1800
0111	112	7168	1C00
1000	128	8192	2000
1001	144	9216	2400
1010	160	10240	2800
1011	176	11264	2C00
1100	192	12288	3000
1101	208	13312	3400
1110	224	14336	3800
1111	240	15360	3C00

Tabelle 2. Zusammenhang zwischen den Bits 4 bis 7 von Speicher 53272 und dem Ort des Bildschirms in Abschnitt 0

Bildschirm-Experimente

Wenn wir jetzt (falls keine Programmänderung in der Zwischenzeit durchgeführt wurde) CONT eingeben, wird der Ursprungszustand (Bildschirm bei 1024) wieder hergestellt und dies durch die Angabe der I- und W-Werte 3 und 16 angezeigt.

Wenn Sie mit diesem Programm in den Bereich 4096 bis 8191 vorstoßen, werden Sie feststellen, daß hier kein normaler Bildschirm möglich ist. Hier stören die mehrfach schon beschworenen Geisterbilder des Zeichen-ROM, die in diesem Bereich liegen. Es kann sogar passieren, daß der Rechner nach der Eingabe von CONT nur noch SYNTAX ERRORs meldet und nicht mehr in den Normalzustand zurückzuführen ist. Ab 8192 bis 15360 (jeweils Start des Bildschirmes) kann man wieder ohne Störung Bildschirme einrichten. Wenn Sie jetzt mal I=2 und verschiedene W-Werte versuchen, sehen Sie nur Nonsens oder gar nichts auf dem Bildschirm, dasselbe geschieht bei I=0.

Das ist wieder eine Besonderheit des VIC-II-Chip. Er ist so strukturiert, daß der (im Normalfall) in diesen beiden Abschnitten keinen Zugang zu den normalen Zeichenspeichern hat. Dafür gibt es in Abschnitt 1 (I=2) keine Störung durch die Zeichen-Geisterbilder, ebenso in Abschnitt 3 (I=0). In Abschnitt 2 (I=1) begegnen wir zwischen 36864 und 40959 wieder den hier ein zweites Mal vorhandenen Zeichen-Gespenstern. Unterhalb von 36864 läßt sich der neue Bildschirm gut verwenden.

Der verborgene Speicher — RAM-Bereiche unter dem ROM

Ein Problem stellt sich hier und auch im obersten Abschnitt aber noch auf andere Weise: Wenn wir den Bildschirm zum Beispiel mit I=1 und W=128 nach 40960 legen (tun Sie es bitte nicht!), dann erhalten wir bei jeder Eingabe nur noch »SYNTAX ERRORs« und können den Computer nur durch Aus- und Einschalten wieder zu normaler Tätigkeit bewegen. Was ist da los? Die Erklärung ist, daß von 40960 bis 49151 das Basic-ROM und von 57344 bis 65535 das Betriebssystem dem RAM überlagert sind. Wenn man in diese Regionen hineinPOKEd, landet die Information natürlich im darunterliegenden RAM. Das was auf dem Bildschirm erscheint wenn wir aus dem Programm-Modus aussteigen, ist allerdings — leider — der Inhalt des darüberliegenden ROM. Ersetzen Sie aber mal das »END« in Zeile 60 durch die folgenden Zeilen:

```
60 PRINT CHR$(147)I,W:PRINT"
BILDSCHIRM LIEGT UNTER DEM
ROM«
65 GET A$:IF A$=" "THEN 65
```

Siehe da! Es funktioniert also im Programm-Modus (zum Beispiel mit I=1 und W=128). Wir können daher unter das Basic-ROM auf diese Weise acht Bildschirme legen. Unter das Betriebssystem lassen sich so auch Bildschirme legen, nur können wir hier den Text nicht lesen, weil der VIC-II-Chip — wie gesagt — hier keinen Zugriff zum Zeichen-ROM hat. So legt zum Beispiel I=0 und W=240 den Bildschirm nach 64512, was leicht nachprüfbar ist durch die Zeile:

```
62 POKE 65000, 1:POKE 55784,1.
```

Damit wenden wir uns nun dem kritischen Bereich zwischen 53248 und 57343 zu. Hier liegen ja das Zeichen-ROM und die Ein- und Ausgabebausteine. Normalerweise — wie man auch durch unsere POKE in diesen Bereich erkennen kann — sind hier die Ein- und Ausgabe-Bausteine eingeschaltet. Wenn wir hierher Bildschirme legen, kann alles mögliche passieren, weil wir Register des VIC-II-Chip, des SID und CIAs beeinflussen. Hier sollte man mit viel Vorsicht und gegebenenfalls nur in Maschinensprache operieren.

Was wir durch die Programmzeile 62 noch erkennen können: Das Bildschirmfarben-RAM verschiebt sich nicht, egal, welches Speicherviertel wir wählen und wohin wir den Bildschirm auch legen: Das Farb-RAM liegt immer von 55296 bis 56295.

Wohin mit der Bit-Map?

Nun aber zum großen Speicherfresser: Zur Bit-Map. Mit ihren 8000 Byte paßt sie im Prinzip achtmal in unseren Computer. Im ersten Viertel (0 bis 16383) haben wir sie schon gehabt und das als unbefriedigend empfunden. Nun wollen wir uns andere Möglichkeiten ansehen und dabei noch bedenken, daß wir auf den normalen Zeichensatz verzichten können (wir stellen ja hochauflösende Grafik dar!). Zu diesem Zweck werden wir das bisher verwendete Bildschirmtestprogramm um einen Hochauflösungsteil erweitern (Listing 1). Um die leidige Eintipperei minimal zu halten, wurde auf Schönheit und erläuternde REM-

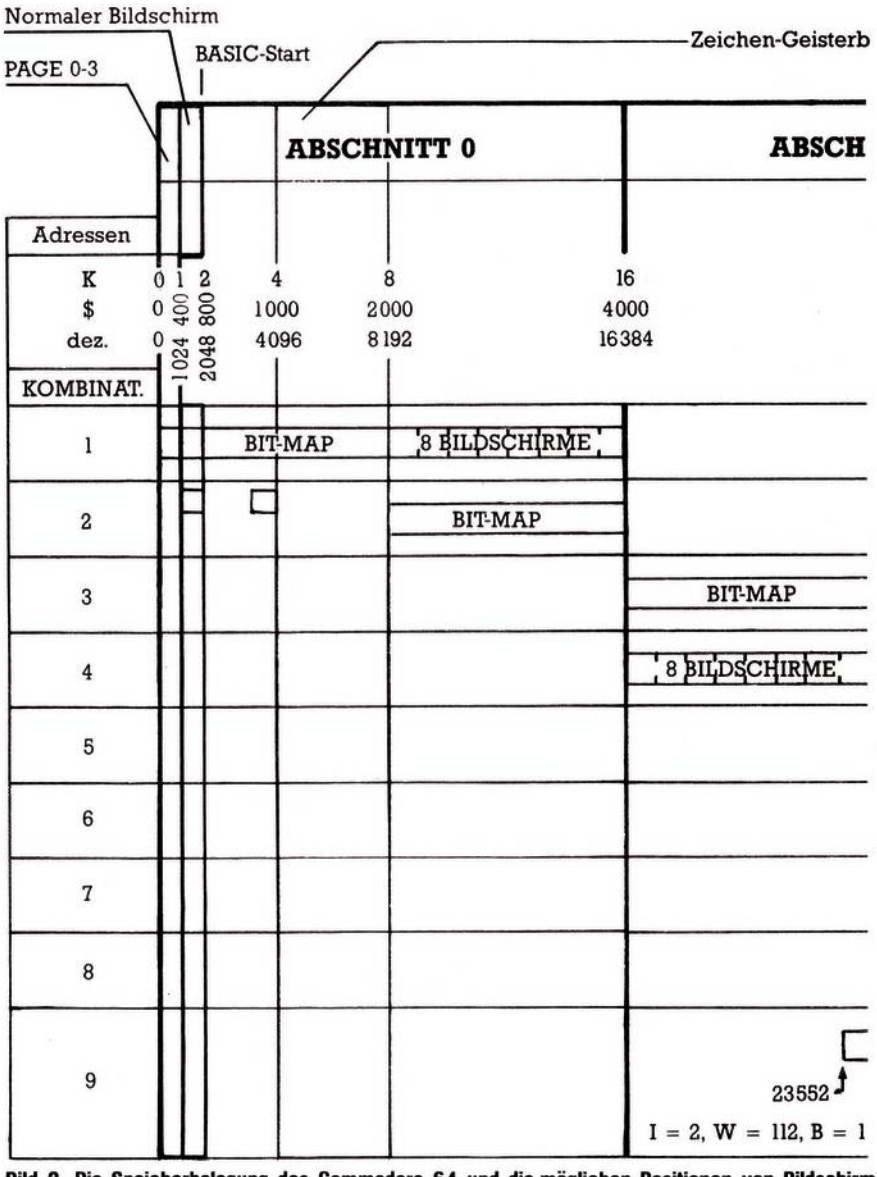


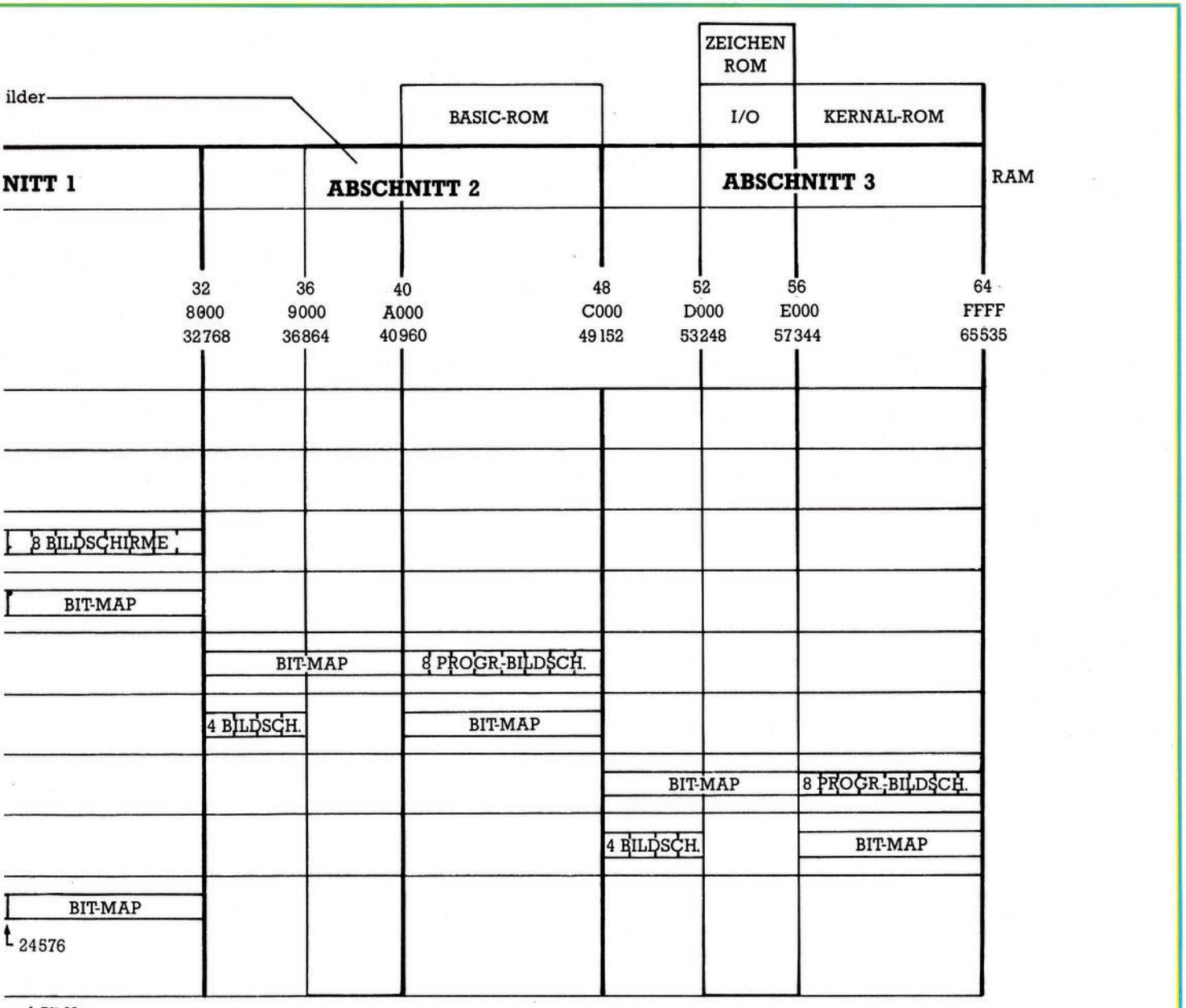
Bild 2. Die Speicherbelegung des Commodore 64 und die möglichen Positionen von Bildschirm

Zeilen verzichtet. Der Hochauflösungsteil stimmt weitgehend mit dem Programm aus der letzten Folge überein. Geben Sie also jetzt das Listing 1 ein, speichern Sie es möglichst gleich ab und probieren Sie es aus.

In Bild 2 sind alle möglichen Positionen der Bit-Map und des Bildschirmes angegeben.

Wie man sehen kann, scheiden die Kombinationen Nummer 1 und Nummer 7 von vornherein aus, weil wir mit dem Löschen der Bit-Map auch das Lebenslicht unseres Rechners ausblasen. Die Kombination Nummer 2 kennen wir schon: So haben wir in der letzten Folge hochauflösende Grafik betrieben und waren enttäuscht über den geringen verbliebenen Basic-Speicher. Bei Nummer 5 funken uns die Zeichen-Spiegelbilder in die Bit-Map, diese Kombination scheidet also auch aus. Nett sieht es aus, wenn wir die Kombinationen Nummer 6 und Nummer 8 testen. Hier machen sich die ROM-Inhalte grafisch zwar ganz interessant aus, aber mit dem Sinn unserer hochauflösenden Grafik hat das nichts mehr zu tun. Für uns brauchbar sind die Positionen im Abschnitt 1: Kombinationen Nummer 3 und Nummer 4. Ein Maximum an Basic-Speicher findet man bei der letzten gezeigten Kombination Nummer 9, wo ab 23552 der Bildschirm und ab 24576 die Bit-Map liegen. Wenn Sie den Basic-Speicher für diese Anordnung schützen, mit POKE 55,0:POKE 56,92:POKE 52,92 und den Computer dann mit PRINT FRE(I) nach dem freien Speicherplatz fragen, dann erhalten Sie als Antwort immerhin satte 21501 freie Byte.

Der Idealfall wäre es, wenn man die Bit-Map unter das ROM legen könnte (Kombinationen 6 oder 8). Das geht natürlich auch! Von Basic aus wird ein Programm dann allerdings noch langsamer, weil man für jeden Punkt ähnliche Operationen vornehmen müßte, wie wir sie in der zweiten Folge beim Kopieren des Zeichen-ROM ins RAM verwendet haben. Auch so ist die ganze Hochauflösungsgrafik schon ziemlich langsam. Wir werden aber in kommenden Folgen einige Routinen in Maschinensprache kennenlernen, die uns mehr Möglichkeiten eröffnen.



und Bit-Map

Hochauflösende Grafik in Farbe: Kann das der C 64 überhaupt? Die Antwort lautet Ja und Nein. Ja, weil der bislang von uns verwendete Bit-Map-Modus anstelle der zwei bisher benutzten auch mit vier Farben ablaufen kann. Nein, weil die Punkteauflösung dann eigentlich nicht mehr die Bezeichnung »hochauflösend« verdient. Die horizontale Auflösung geschieht hier nämlich nur noch in 160 Positionen anstelle der

Die Grafik bekennt Farbe: Der Bit-Map-Mehrfarben-Modus

bisher ansprechbaren 320. Meine persönliche Meinung dazu ist, ernsthafte hochauflösende Grafik sollte sich im bisher besprochenen Bit-Map-Modus abspielen, denn ein Bildschirm mit 64000 Bildpunkten ist schon eine Minimalanforderung. 32000 Bildschirmpositionen sind allenfalls für Spielereien ganz nett, wenn auch etwas teuer, denn ohne Farbmonitor haben Sie von der Farbe nicht viel und über den Unterschied zwischen Farbfernseher und Farbmonitor haben wir im Verlauf dieser Serie schon etwas erfahren.

Nach dieser etwas desillusionierenden Vorrede widmen wir uns also der mehrfarbigen Bit-Map-Grafik. Es handelt sich um eine Kombination aus der bisher bekannten Bit-Map-Grafik und dem Mehrfarben-Modus, den wir schon bei den mehrfarbigen Zeichen kennengelernt haben. Auf dem Bildschirm wird der Inhalt der Bit-Map wiedergegeben, aber die Farben der Zeichnungen sind abhängig von Bit-Paaren. Welche Farben Sie bei welcher Paarung sehen, darüber gibt Tabelle 3 Aufschluß.

Es existiert also eine Hintergrundfarbe für den gesamten Bildschirm, die überall dort auftaucht, wo in der Bit-Map ein Bit-Paar 00 vorhanden ist. Diese Hintergrundfarbe ist durch den Zahlenwert in Register 53281 bestimmt. Die anderen drei Farben treten jeweils in den 8 x 8-Bit-Bildschirmfeldern auf, in die das Video-RAM, beziehungsweise der Bildschirmfarbspeicher aufgeteilt sind.

Wir ergänzen unser Programm 1 für den Mehrfarben-Modus: Außer dem Bit-Map-Modus muß hier also noch der Mehrfarb-Modus eingeschaltet werden. Das haben wir in der letzten Folge kennengelernt:
145 POKE 53270,PEEK(53270) OR 16
Weiterhin ändern wir noch die Zeile

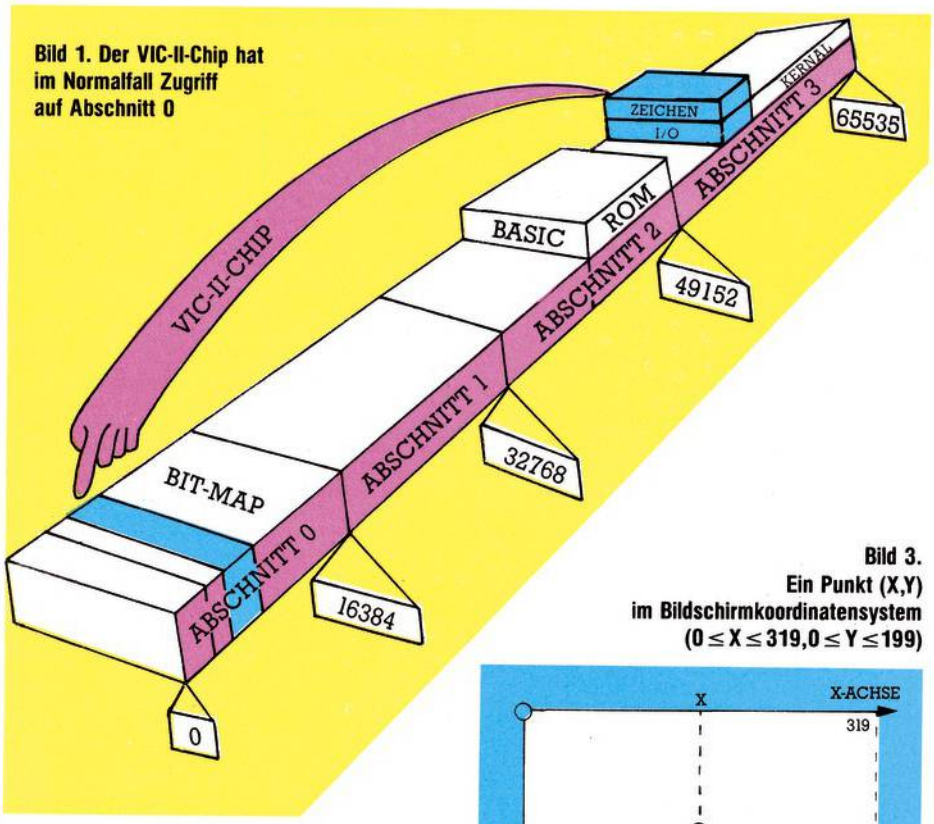


Bild 1. Der VIC-II-Chip hat im Normalfall Zugriff auf Abschnitt 0

5 und schreiben anstelle von F=6 jetzt GOSUB 300. Folgende Zeilen kommen neu hinzu:

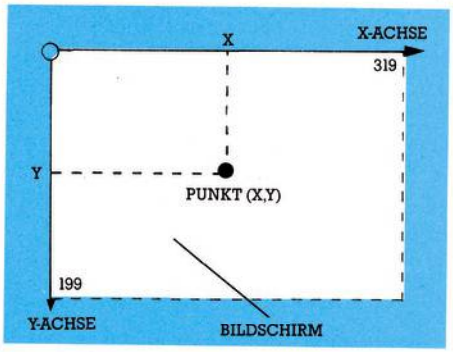
```
300 PRINT CHR$(147)CHR$(17)" FARBENWAHL:" 310 PRINT CHR$ (17)"HINTER-GRUNDFARBE"TAB (30);:INPUT F0 320 PRINT "BITS 4-7 VIDEOMATRIX"TAB(30);:INPUT F1
330 PRINT "BITS 0-3 VIDEOMATRIX"TAB(30);:INPUT F2
340 PRINT "BILDSCHIRMFARBSPEICHER"TAB(30);:INPUT F3
350 POKE 53281,F0:F=16*F1+F2: FOR I=55296 TO 56295:POKE I,F: NEXT I
360 PRINT CHR$(147):RETURN
```

Um des Effektes willen ändern wir noch die Zeile 220, in der die Videomatrix mit den Farbzahlen belegt wird:

```
220 FOR J=0TO 998 STEP 2:POKE J,F:NEXT J:FOR J=1TO 999 STEP 2:POKE J,F+1:NEXT J
```

Starten Sie dieses Programm nach dem Abspeichern mit RUN, probieren Sie alle möglichen Farbkennzahlen aus. Sie werden fast immer bemerken, daß die Hoch- und Tiefpunkte der Kurve nicht mitgezeichnet werden. Das liegt daran, daß unter Verfahren der Berechnung, welches Bit in welchem Byte gesetzt werden soll, noch auf einzelne Bits und nicht die paarweise Verwendung abgestimmt ist. Es gibt zwei Möglichkeiten: Entweder ändert man das Berechnungsverfahren in Zeile 240, um an die richtige Stelle die passenden Bit-Paare zu bekom-

Bild 3. Ein Punkt (X,Y) im Bildschirmkoordinatensystem (0 ≤ X ≤ 319, 0 ≤ Y ≤ 199)



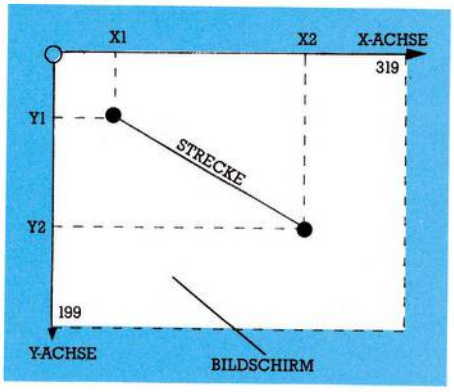


Bild 4: Eine Strecke (X1,Y1) bis (X2,Y2) im Bildschirmkoordinatensystem

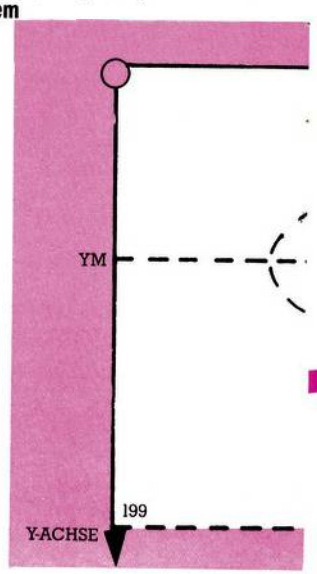


Bild 5: Ellipsenbogen der Ellipse mit Mittelpunkt (XM,YM), Halbmessern HX und HY, von Winkel WU bis WO im System der Bildschirmkoordinaten

men, oder wir ändern die Programmzeile 230 zu:
230 FOR I=0 TO 319 STEP 2:Y=FNA(X)

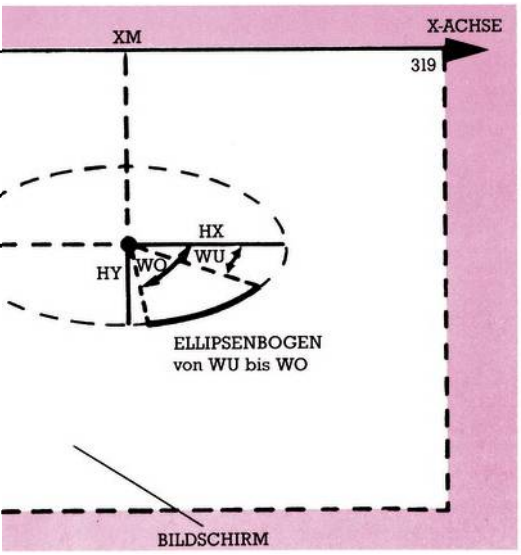
Das ist zwar mal wieder etwas primitiv, aber — wie gesagt — sind meine Ambitionen zur Mehrfarben-»Hochauflösung« sowieso nicht so stark. Wer Lust hat, kann sich ja gerne mal mit der korrekten Weise der neuen Berechnung herumschlagen. Das Wissen, diese Aufgabe zu bewältigen, haben Sie jetzt. Auf dem Schwarzweiß-Monitor sieht zum Beispiel folgende Zahlenkombination ganz gut aus:
Hintergrundfarbe: 7
Bits 4 bis 7: 5
Bits 0 bis 3: 1
Bildschirmspeicher: 0
Außerdem natürlich noch: I=2, W=112, B=1.

Mit dem hier folgenden Abschnitt soll zunächst einmal die hochauflösende Grafik beiseite gelegt werden. Dornröschen ist erwacht und genesen, allerdings noch nicht voll bei Kräften. Das wird in einer späteren Folge noch anders werden. Davor wollen wir aber noch weitere Grafik-Besonderheiten des C 64 behandeln. An dieser Stelle wollen wir jedoch einen Zwischenhalt einlegen und eine kleine Sammlung von Basic-Unterprogrammen zur Grafik-Programmierung vorstellen. In Listing 2 sind diese Grafik-Unterprogramme, in Listing 3 ein Beispiel-Aufrufprogramm abgedruckt. Beim Eintippen beider Programme können Sie die REM-Zeilen ohne Schaden weglassen.

Erläuterungen zu Listing 2 (Zeilenbereich 49990 bis 51500)

◆Sprungtabelle:

Das ist in Basic im allgemeinen nicht üblich, sondern wird häufiger bei Maschinensprache-Programmen verwendet. Trotzdem hat es auch hier seine Vorteile. Es kann ja sein, daß Sie einige Änderungen oder Ergänzungen in den Unterprogrammen vornehmen wollen. Sie müßten dann auch immer die Adressen in den jeweils aufzurufenden Hauptprogrammen umschreiben. Mit der Sprungtabelle ist das nicht mehr nötig, denn die GOSUB-Adressen im Hauptprogramm bleiben unverändert, nur die neuen GOTO-Adressen im Unterprogramm sind einzusetzen.



◆HIRES an

Hier machen wir uns die Erkenntnisse dieser Folge zunutze und legen den Bildschirm nach 23552 und die Bit-Map nach 24576. Beides müssen wir — wie gehabt — vor dem Überschreiben durch Basic schützen mit:
POKE 52,92:POKE 56,92.

Am besten packt man diese POKE-Befehle gleich in die ersten Zeilen des aufrufenden Hauptprogramms.

◆Bit-Map-Löschen

Hierzu gibt es nichts mehr zu sagen, außer, daß I als Laufvariable dient.

◆Farbgebung

Bevor dieses Unterprogramm aufgerufen wird, müssen im Hauptprogramm
F1 = Zeichenfarbe und
F2 = Hintergrundfarbe
definiert sein. An Variablen treten noch auf:
F = Farbcodezahle auf F1 und F2
I = Laufvariable

◆HIRES aus

Dieses Programm stellt die ursprüngliche Speicherorganisation wieder her (Bildschirm- und Zeichenspeicher) und schaltet in den Normalmodus zurück.

◆Punkt setzen

Auch hier müssen vor dem Aufruf des Unterprogramms im Hauptspeicher die
Punktkoordinaten X,Y
definiert sein (siehe Bild 3) sowie die
L — Löschmarke

Wenn L = 0 ist, wird der Punkt gesetzt (Zeile 50930), so, wie wir das schon kennen. Ist L = 1, dann wird ein vorhandener Punkt gelöscht (Zeile 50920).

Die Zeile 50905 achtet darauf, daß keine Punktkoordinate außerhalb des Bildschirms liegt, was unter Umständen ein Aussteigen mit Fehlermeldung im Hochauflösungsverfahren zur Folge hätte. Das ist hier zwar nicht so tragisch, weil man durch Eingeben von GOTO 50030 »RETURN« schnell wieder in den Normalmodus gelangen kann (eventuell muß vorher noch »SHIFT« +»CLR/HOME« gedrückt werden). Trotzdem ist es dumm, wenn inmitten all dieser zeitraubenden Grafiktätigkeiten auch noch der Rechner aussteigt. Eine Grenzüberschreitung der Koordinaten ist um so leichter möglich, als die Punkt-Routine von allen folgenden Unterprogrammen aufgerufen wird. Außer X,Y und L tauchen noch die Variablen BY und BI auf, die wir schon kennengelernt haben als das Byte, in dem ein Bit zu setzen oder zu löschen ist.

◆Punkt löschen

Hier geschieht nichts anderes, als die Löschmarke L auf 1 zu setzen und dann in die Punkt-Setz-Routine zu springen. Deswegen gilt für dieses Unterprogramm dasselbe wie für das vorangegangene.

◆Strecke zeichnen

Vor dem Aufruf müssen dem Rechner schon
der Startpunkt (X1,Y1) und
der Endpunkt (X2,Y2) der Strecke
bekannt gemacht sein (siehe Bild 4).

Den mathematisch Versierten wird es bei der Betrachtung der Zeilen 51120 beziehungsweise 51160 schon aufgefallen sein, daß zur Berechnung der Punkte, aus denen sich die Strecke zusammensetzt, die sogenannte 2-Punkte-Form der Geradengleichung verwendet wurde:

$$\frac{Y-Y1}{X-X1} = \frac{Y2-Y1}{X2-X1}$$

Den mit Mathematik nicht so vertrauten Lesern sei gesagt, daß es sich um eine Formel aus der sogenannten analytischen Geometrie handelt. Das ist ein Gebiet der Mathematik, das für die Grafik auf Computern eine nicht unerhebliche Rolle spielt.

Die Punkte (X1, Y1) und (X2, Y2) dürfen auch außerhalb des Bildschirmsystems liegen. Im ersten Teil des Unterprogramms dient die Übergabevariable X auch gleichzeitig als Laufvariable während Y berechnet wird. Wenn allerdings der Absolutbetrag von X2-X1 kleiner als 5 wird, verkehren sich die Verhältnisse: Y wird Laufvariable und X berechnet. Das beschleunigt das Zeichnen von Senkrechten und verhindert außerdem eine Division durch Null. Der Wert von 5 ist dabei ziemlich willkürlich gewählt. L ist wieder die Löschmarke.

◆Strecke löschen

Es gilt dasselbe wie für das Unterprogramm Strecke zeichnen, nur daß hier wieder die Löschmarke gesetzt wird.

Grafik-Grundlagen

◆Ellipse zeichnen

Vor dem Aufruf müssen folgende Werte schon definiert sein (siehe auch Bild 5):

(XM,YM) = Mittelpunktkoordinaten
HX = Halbmesser in X-Richtung
HY = Halbmesser in Y-Richtung
WU,WO = Der zu zeichnende Ellipsenbogen beginnt beim Winkel WU und endet beim Winkel WO (Gradmaß)

Eine volle Ellipse wird also gezeichnet, wenn WU = 0 und WO =

```
1 REM ************************
2 REM *    PROGRAMM  1       *
3 REM ************************
5 F=6:DEFFNA(X)=50*SIN(X/30)+100
10 PRINTCHR$(147)CHR$(17)"EINGABE DER WERTE"CHR$(17)
20 PRINTCHR$(17)"I(SIEHE TAB.1) ABSCHNITT-KENNZIFFER"CHR$(17)
30 PRINT"W(SIEHE TAB.2) BILDSCHIRM-KENNZIFFER"CHR$(17)
40 PRINT"B=0,BIT-MAP IM UNTEREN ABSCHNITT-BEREICH"
50 PRINT" =1,BIT-MAP IM OBEREN ABSCHNITT-BEREICH"CHR$(17)
60 PRINTCHR$(17):INPUT"I,W,B=";I,W,B:A1=3-I:A2=W/16*1024+A1*16384:A3=A1*16384+B*
8192
70 P=A2/256:PRINTCHR$(17)CHR$(17)"MIT DIESEN EINGABEN HABEN SIE"
80 PRINT"DEN ABSCHNITT "A1" GEWAEHLT."
90 PRINT"IHR BILDSCHIRM STARTET BEI "A2
100 PRINT"UND IHRE BIT-MAP BEI "A3
110 PRINTCHR$(17)"IST DAS SO IN ORDNUNG?(J/N)"
120 GETA$:IFA$=""THEN120
130 IFA$="N"THEN10
140 PRINTCHR$(147)
145 REM **** NEUER SPEICHER ****
150 POKE56576,(PEEK(56576)AND252)ORI
160 POKE56578,PEEK(56578)OR3
170 POKE53272,(PEEK(53272)AND15)ORW
180 POKE648,P
190 POKE53265,PEEK(53265)OR32
200 POKE53272,PEEK(53272)OR(8*B)
210 FORJ=0TO7999:POKEA3+J,0:NEXTJ
220 FORJ=0TO999:POKEA2+J,F:NEXTJ
230 FORX=0TO319:Y=FNA(X)
240 BY=(XAND504)+40*(YAND248)+(YAND7):BI=7-(XAND7)
250 POKEA3+BY,PEEK(A3+BY)OR(2↑BI):NEXTX
260 GETA$:IFA$=""THEN260
265 REM **** ALTER SPEICHER ****
270 POKE53272,21:POKE53265,27:POKE648,4:POKE56578,63:POKE56576,151
280 PRINTCHR$(147):END

READY.
```

Listing 1.
Testprogramm zur Bildschirmlokalisierung

```
49990 REM ------------------
49991 REM --UNTERPROGRAMME--
49992 REM ------------------
49993 REM
49996 REM ------------------
49997 REM --SPRUNGTABELLE--
49998 REM ------------------
49999 REM
50000 GOTO50500:REM HIRES AN
50010 GOTO50600:REM BIT-MAP-LOESCHEN
50020 GOTO50700:REM FARBGEBUNG
50030 GOTO50800:REM HIRES AUS
50040 GOTO50900:REM PUNKT SETZEN
50050 GOTO51000:REM PUNKT LOESCHEN
50060 GOTO51100:REM STRECKE ZEICHNEN
50070 GOTO51200:REM STRECKE LOESCHEN
50080 GOTO51300:REM ELLIPSE ZEICHNEN
50090 GOTO51400:REM ELLIPSE LOESCHEN
50100 GOTO51500:REM KOMBINIERTES HIRES AN
50490 REM
50491 REM ------------------
50492 REM ----HIRES AN-------
50493 REM ------------------
50494 REM
50500 POKE56576,(PEEK(56576)AND252)OR2
50510 POKE56578,PEEK(56578)OR3
50520 POKE53272,120:POKE648,92
50530 POKE53265,PEEK(53265)OR32
50540 RETURN
50590 REM
50591 REM --------------------
50592 REM -BIT-MAP-LOESCHEN--
50593 REM --------------------
50594 REM
50600 FORI=24576TO32575:POKEI,0:NEXTI
50610 RETURN
50690 REM
50691 REM --------------------
50692 REM ------FARBGEBUNG------
50693 REM --------------------
50694 REM
50700 F=16*F1+F2
50710 FORI=23552TO24551:POKEI,F:NEXTI
50720 RETURN
50790 REM
50791 REM ---------------------
50792 REM ------HIRES AUS------
50793 REM ---------------------
50794 REM
50800 POKE53265,27:POKE53272,21:POKE648,4
50810 POKE56578,63:POKE56576,151
50820 RETURN
50890 REM
50891 REM ---------------------
50892 REM -----PUNKT SETZEN-----
50893 REM ---------------------
50894 REM
50900 L=0
50905 IFX<00RX>319ORY<00RY>199THEN50940
50910 BY=(XAND504)+40*(YAND248)+(YAND7):BI=7-(XAND7)
50920 IFL=1THENPOKE24576+BY,PEEK(24576+BY)ANDNOT(2↑BI):GOTO50940
50930 POKE24576+BY,PEEK(24576+BY)OR(2↑BI)
50940 RETURN
50990 REM
50991 REM ------------------
50992 REM -----PUNKT LOESCHEN-----
50993 REM ------------------
50994 REM
51000 L=1:GOTO50905
51090 REM
51091 REM ------------------
51092 REM -----STRECKE ZEICHNEN-----
51093 REM ------------------
51094 REM
51100 L=0
51110 IFABS(X2-X1)<5THEN51150
51115 FORX=X1TOX2STEP(X2-X1)/319
51120 Y=(Y2-Y1)/(X2-X1)*(X-X1)+Y1
51130 GOSUB50905:NEXTX
51140 RETURN
51150 FORY=Y1TOY2STEP(Y2-Y1)/199
51160 X=(X2-X1)/(Y2-Y1)*(Y-Y1)+X1
51170 GOSUB50905:NEXTY
51180 RETURN
51190 REM
51191 REM ------------------
51192 REM -----STRECKE LOESCHEN-----
51193 REM ------------------
51194 REM
51200 L=1:GOTO51110
51290 REM
51291 REM ------------------
51292 REM -----ELLIPSE ZEICHNEN-----
51293 REM ------------------
51294 REM
51300 L=0
51310 FORW=WUTOWO:WB=W*π/180
51320 X=XM+HX*COS(WB):Y=YM+HY*SIN(WB)
51330 GOSUB50905
51340 NEXTW:RETURN
51390 REM
51391 REM ------------------
51392 REM -----ELLIPSE LOESCHEN-----
51393 REM ------------------
51394 REM
51400 L=1:GOTO51310
51490 REM
51491 REM ------------------
51492 REM --KOMBINIERTES HIRES AN--
51493 REM ------------------
51494 REM
51500 GOSUB50000:GOSUB50010:GOTO50020
```

Listing 2.
Unterprogramme zur hochauflösenden Grafik

```
1 REM ********************************
2 REM *  GRAFIK TEST PROGRAMM VON    *
3 REM *  H.PONNATH 1984 VERBROCHEN   *
4 REM ********************************
5 POKE52,92:POKE56,92:DEFFNA(X)=50*SIN(X/30)+100
7 REM ********************************
8 REM *     MENUE-GUTEN APPETIT      *
9 REM ********************************
10 PRINTCHR$(147)CHR$(17)"UNTERPROGRAMME GRAFIK TEST"CHR$(17)
20 PRINTTAB(2)"NUR HIRES AN"TAB(25)"1"
30 PRINTTAB(2)"NUR HIRES AUS"TAB(25)"2"
40 PRINTTAB(2)"BIT MAP LOESCHEN"TAB(25)"3"
50 PRINTTAB(2)"FARBGEBUNG"TAB(25)"4"
60 PRINTTAB(2)"KOMBINATION"TAB(25)"5"
62 PRINTTAB(2)"PUNKTE SETZEN"TAB(25)"6"
64 PRINTTAB(2)"PUNKTE LOESCHEN"TAB(25)"7"
66 PRINTTAB(2)"STRECKE ZEICHNEN"TAB(25)"8"
68 PRINTTAB(2)"STRECKE LOESCHEN"TAB(25)"9"
70 PRINTTAB(2)"ELLIPSE ZEICHNEN"TAB(25)"A"
72 PRINTTAB(2)"ELLIPSE LOESCHEN"TAB(25)"B"
74 PRINTTAB(2)"DEMONSTRATION"TAB(25)"C"
76 PRINTTAB(1)"MENUE"TAB(25)"M"
78 PRINTTAB(1)"AUSSTEIGEN"TAB(25)"←"
80 GETA$:IFA$=""THEN80
90 IFA$="←"THENEND
92 IFA$="A"THENA$="10"
94 IFA$="B"THENA$="11"
96 IFA$="C"THENA$="12"
98 IFA$="M"THEN10
100 ONVAL(A$)GOSUB50000,50030,50010,200,300,400,500,600,700,800,900,1000
110 GOTO80
190 REM ******************************
191 REM *     FARBGEBUNG             *
192 REM ******************************
200 INPUT"ZEICHENFARBE,HINTERGRUNDFARBE=";F1,F2:GOTO50020
290 REM ****************************
291 REM * FARBE FUER KOMBINATION  *
292 REM ****************************
300 INPUT"ZEICHENFARBE,HINTERGRUNDFARBE=";F1,F2:GOTO50100
390 REM ****************************
391 REM * BEISPIEL PUNKTE SETZEN  *
392 REM ****************************
400 GOSUB50000:FORX=0TO319:Y=FNA(X):GOSUB50040:NEXTX
410 RETURN
490 REM ****************************
491 REM * BEISPIEL PUNKTE LOESCHEN*
492 REM ****************************
500 GOSUB50000:FORX=20TO250:Y=FNA(X):GOSUB50050:NEXTX
510 RETURN
590 REM ****************************
591 REM * AUFRUFPROGRAMM FUER     *
592 REM * STRECKE ZEICHNEN        *
593 REM ****************************
600 PRINT"■STRECKE■ VON (X1,Y1) BIS (X2,Y2)":INPUT"X1,Y1,X2,Y2=";X1,Y1,X2,Y2
610 GOSUB50000:GOTO50060
690 REM ****************************
691 REM * AUFRUFPROGRAMM FUER     *
692 REM * STRECKE LOESCHEN        *
693 REM ****************************
700 PRINT"■STRECKE■ VON (X1,Y1) BIS (X2,Y2)":INPUT"X1,Y1,X2,Y2=";X1,Y1,X2,Y2
710 GOSUB50000:GOTO50070
790 REM ****************************
791 REM * AUFRUFPROGRAMM FUER     *
792 REM * ELLIPSE ZEICHNEN        *
793 REM ****************************
800 PRINT"■ELLIPSE■ MIT MITTELPUNKT (XM,YM),":PRINTTAB(1)"HALBMESSERN HX UND HY"
810 PRINTTAB(1)"ZEICHNEN VON WINKEL WU":PRINTTAB(1)"BIS WINKEL WO (GRADMASS)"
820 INPUT"XM,YM,HX,HY,WU,WO=";XM,YM,HX,HY,WU,WO:GOSUB50000:GOTO50080
890 REM ****************************
891 REM * AUFRUFPROGRAMM FUER     *
892 REM * ELLIPSE LOESCHEN        *
893 REM ****************************
900 PRINT"■ELLIPSE■ MIT MITTELPUNKT (XM,YM),":PRINTTAB(1)"HALBMESSERN HX UND HY"
910 PRINTTAB(1)"LOESCHEN VON WINKEL WU":PRINTTAB(1)"BIS WINKEL WO (GRADMASS)"
920 INPUT"XM,YM,HX,HY,WU,WO=";XM,YM,HX,HY,WU,WO:GOSUB50000:GOTO50090
990 REM ****************************
991 REM * AUFRUFPROGRAMM FUER     *
992 REM * DEMONSTRATION           *
993 REM ****************************
1000 GOSUB50030:PRINTCHR$(147):FORI=1TO10:PRINTCHR$(17);:NEXTI
1010 PRINTTAB(8)"BITTE ETWAS GEDULD":PRINTTAB(5)"DIE BIT-MAP WIRD GELOESCHT"
1020 GOSUB50010:PRINTTAB(5)"UND MIT FARBE VERSEHEN":FORI=1TO1000:NEXTI
1030 F1=7:F2=6:GOSUB50000:GOSUB50020:FORI=1TO500:NEXT:GOSUB50030
1040 PRINTCHR$(147):FORI=1TO10:PRINTCHR$(17);:NEXTI
1050 PRINTTAB(5)"ZEICHNEN VON STRECKEN":FORI=1TO500:NEXTI
1060 GOSUB50000:FORI=0TO12
1070 X1=30+I*10:Y1=180-I*14.17:X2=150+I*13.3:Y2=10+I*14.583
1080 GOSUB50060:NEXTI:FORK=0TO9:F1=K:F2=K+1:GOSUB50020:FORJ=1TO500:NEXTJ:NEXTK
1090 GOSUB50030:PRINT:PRINTTAB(5)"STRECKEN LOESCHEN"
1100 FORI=1TO500:NEXTI:GOSUB50000:X1=30:Y1=180:X2=150:Y2=10:GOSUB50070
1110 X1=310:Y1=185:GOSUB50070:FORI=1TO500:NEXTI:GOSUB50030
1120 PRINT:PRINTTAB(5)"ELLIPSEN ZEICHNEN":FORI=1TO500:NEXTI:GOSUB50000
1130 XM=170:YM=150
1140 FORI=0TO16:WU=I*90:WO=WU+90
1150 HX=20+8*INT((3+I)/4):HY=10+8*INT((2+I)/4)
1160 GOSUB50080:NEXTI:X1=0:Y1=0:X2=319:Y2=0:GOSUB50060
1170 X2=0:Y2=199:GOSUB50060:X1=319:Y1=199:GOSUB50060:X2=319:Y2=0:GOSUB50060
1180 X1=100:Y1=0:X2=100:Y2=80:GOSUB50060:X1=0:Y1=80:GOSUB50060
1190 FORX=0TO100:Y=40+25*SIN(X/15):GOSUB50040:NEXTX
1200 X1=219:Y1=0:X2=219:Y2=80:GOSUB50060:X1=319:Y1=80:GOSUB50060
1210 XM=249:YM=40:HX=20:HY=30:WU=0:WO=360:GOSUB50080
1220 X1=279:Y1=10:X2=279:Y2=70:GOSUB50060:Y1=40:X2=309:Y2=10:GOSUB50060
1230 Y2=70:GOSUB50060
2000 RETURN
```

Listing 3. Grafik-Test und Demonstration

Grafik-Grundlagen

360 ist. Der Kreis ist ein Sonderfall der Ellipse. Dann muß nur HX = HY sein.

Für mathematisch Interessierte: Es werden die Parametergleichungen der Ellipse verwendet:
X = XM + HX*COS(WB) und
Y = YM + HX* SIN(WB)

Auch hier gibt es keine Einschränkung wie beim Strecken-Zeichnen in der Größe von XM, YM, HX, WU, WO.

W ist eine Laufvariable (ein Winkel) der in WB (gleicher Winkel im Bogenmaß) umgerechnet wird. L ist wieder die Löschmarke.

◆Ellipse löschen

Bis auf das Setzen der Löschmarke gilt dasselbe wie für das Zeichnen der Ellipse.

◆Kombination

Erfordert schon definierte Farbkennzahlen F1 und F2 (siehe Farbgebung) und schaltet dann die Hochauflösung an, löscht die Bit-Map und sorgt für die Farbe.

Soweit die Unterprogramme in Listing 2.

Ein Beispiel für die Möglichkeiten der Grafik-Bibliothek

Als ein Beispiel für die Möglichkeiten der Unterprogramm-Sammlung habe ich (ohne nun besonders auf Schönheit zu achten — das sind Sie ja von mir schon gewohnt), noch ein Hauptprogramm angefügt, mit dem Sie etwas herumprobieren können (Listing 3). Das Listing ist ausführlich kommentiert, so daß hier nur wenige Erläuterungen folgen müssen.

Beim Eintippen müssen Sie für einige Zeilen die Abkürzungen (siehe Handbuch Seite 130 ff) der Basic-Befehle eingeben, da die Zeilen sonst länger als 80 Zeichen werden.

Nach »RUN« sehen Sie ein Menü, das alle Möglichkeiten der Grafik-Unterprogramme ansteuert. Die Optionen 8 (Strecke zeichnen) bis B (Ellipse löschen) sowie 4 (Farbgebung) und 5 (Kombinationen) erfordern Eingaben. Es ist daher sinnvoll, diese Optionen nur im Normalmodus anzuwählen. Der Normalmodus ist immer dann zu erreichen, wenn Zeichenoperationen im Hochauflösungsmodus abgeschlossen sind.

Fortsetzung auf Seite 178

Fortsetzung von Seite 169

Drücken Sie dann "2', sind Sie wieder im normalen Rechnerbetrieb.

Sollten Sie durch irgendeinen Umstand (zum Beispiel durch Drücken der »RUN/STOP«-Taste) im Hochauflösungsmodus aus dem Programm fallen, dann hilft der folgende Weg:

1. »SHIFT« + »CLEAR/HOME«
2. »RUN« »RETURN«
3. dann »2« eingeben

Die Option »C« zeigt eine kleine Demonstration von Möglichkeiten der Grafik-Unterprogramme. Allerdings sollten Sie ein bißchen Zeit mitbringen, wenn Sie C anwählen: Das ganze dauert zirka 25 Minuten.

Option 6 (Punkte zeichnen) ist so eingerichtet, daß 320 Punkte in Form einer Sinus-Funktion gezeichnet und mit Option 7 (Punkte löschen) teilweise wieder gelöscht werden.

Ausblicke — schnellere Grafik durch Maschinensprache

Diese Folge soll nicht beendet werden, ohne einen kleinen tröstlichen Ausblick. Wie Sie — besonders im letzten Programm — feststellen konnten, braucht man schon einiges Sitzfleisch für hochauflösende Grafik in Basic. Wenn Sie aber ein kommerzielles Grafik-Programm laufen sehen, geht das alles erheblich schneller. Was ist der Unterschied? Da wäre zunächst einmal die Programmiersprache: Unser C 64 kann eigentlich gar kein Basic. Er braucht den Basic-Interpreter, der zunächst jeden Befehl liest und dann in Maschinensprache übersetzt. Die versteht unser Rechner zwar, die Übersetzung und das Lesen dauern jedoch lange Zeit. Eine starke Beschleunigung der Grafik ist möglich durch Programmieren in Maschinensprache. Einige solche Maschinenspracheprogramme zur beschleunigten Grafik werden in den nächsten Folgen gezeigt. Allerdings stoßen wir da bald an die Grenzen unseres Commodore. Ein 8-Bit-Computer mit zirka 1 Megahertz Taktfrequenz wie unser C 64 ist beispielsweise in der Fließkomma-Arithmetik (wie sie für das Zeichnen von Ellipsen nötig ist) zeitlich gehandicapt, und deswegen sind der Geschwindigkeit bei komplexerer Grafik doch einige Grenzen gesetzt.

(Heino Ponnath)

```
100 REM"
110 REM"  UP SORTIEREN
120 REM"
130 REM"  Q1      = ANZAHL ELEMENTE
140 REM"  FF$()   = SORTIERFELD
150 REM"
160 REM"
170 REM"  JJ,LL   = LAUFVARIABLE
180 REM"  Q2$     = ZWISCHENSPEICHER
190 REM"
200 REM"  SORTIERT DAS FELD FF$()
210 REM"  MIT Q1 ELEMENTEN IN ALPHA-
220 REM"  BETISCHER REIHENFOLGE
230 REM"
240 :
250 :
260 FOR JJ=1 TO Q1-1
270 :   FOR LL=JJ+1 TO Q1
280 :     IF FF$(JJ)<=FF$(LL) THEN 320
290 :     Q2$     =FF$(JJ)
300 :     FF$(JJ)=FF$(JJ)
310 :     FF$(LL)=Q2$
320 :   NEXT LL
330 NEXT JJ
340 RETURN
```

Das Beispiel-Listing zum Programmierwettbewerb auf S. 177