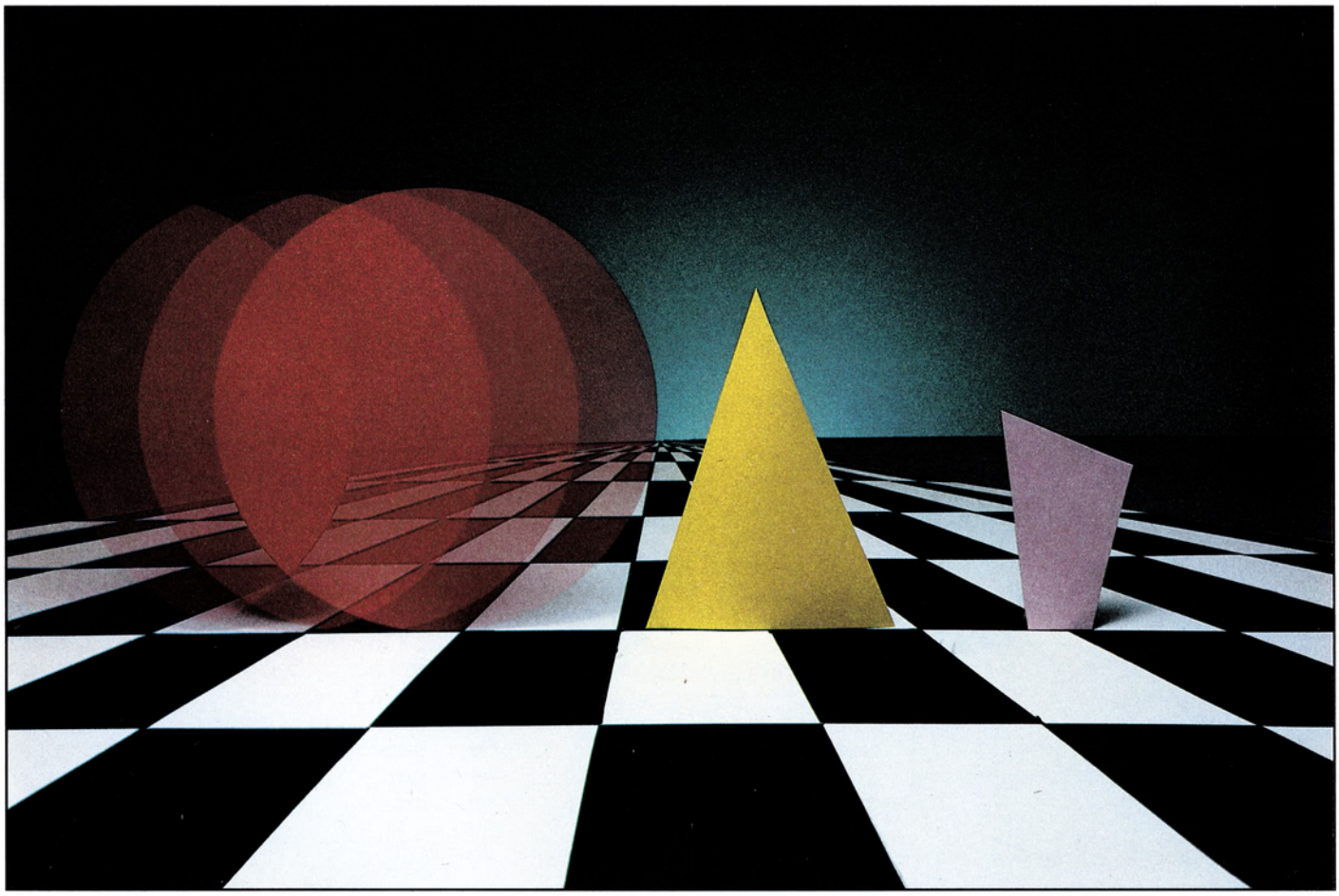

Sprites und Shapes auf dem C128

Grafik und C128: Das heißt optimales Zusammenspiel von Computer und Anwendung. Wie einfach Shapes und Sprites in der Handhabung sind, werden wir Ihnen auf den nächsten Seiten zeigen.

Um die Programmierung der Sprites auf dem C64 zu erklären, ist schon viel Druckerschwärze verbraucht worden. Fast schien es eher die Aufgabe von Assembler-Programmierern zu sein, durch die POKE-Wüsten und Zahlensysteme zu gelangen um schließlich das Ziel – ein bewegtes Sprite auf dem Bildschirm – zu erreichen. Im C128-Modus erspart uns das Basic 7.0 all diese Strapazen: Sprites zu erstellen und zu verwalten wird zum Vergnügen. Nachfolgend lernen wir die dazu nötigen Befehle bis in ihre Feinheiten kennen. Außerdem wird uns eine neue Gruppe von Grafikobjekten beschäftigen: die Shapes.

Sprites

Sollten Sie Ihnen noch nicht begegnet sein, die kleinen Kobolde (»sprite« ist englisch und heißt auf deutsch »Kobold« oder »Gespenst«), dann seien sie hiermit kurz vorgestellt. Es handelt sich um bewegliche grafische Objekte (daher auch der Ausdruck MOB = movable objects, der manchmal verwendet wird), die sowohl als mehr- als auch als einfarbige (dann hochaufgelöste) Figuren ohne Rücksicht auf den gerade eingeschalteten Grafikmodus munter über Text- und Grafikbildschirm huschen können. Die gesamte Verwaltung der Sprites geschieht durch den VIC-Chip, was ihr Auftreten auf den 40-Zeichen-Bildschirm beschränkt. Bevor Sie die folgenden Beispielprogramme ausprobieren, sollten Sie Listing 1 »Alle Sprites« abgetippt und gestartet haben.

Die Befehle zur Spriteprogrammierung

Zehn Basic-Befehle dienen zum Erstellen und Verwalten von Sprites:

SPRDEF

Damit ruft man den Sprite-Editor auf. Der Bildschirm wird gelöscht, links oben erscheint ein Raster mit 24 mal 21 möglichen Positionen, darunter fragt der Computer nach der gewünschten Sprite-Nummer.

Wir haben nun die Wahl zwischen acht vorgesehenen Sprites. Sollten Sie schon ein MOB im Speicher haben, dann rufen Sie es nun mit seiner Nummer auf, ansonsten müssen Sie sich jetzt festlegen: Alle weiteren Sprite-Befehle beziehen sich immer auf diese Nummer. Nach deren Eingabe wird das Rasterfeld mit dem Inhalt eines speziellen Speicherbereiches (aus $0E00 bis $0FFF = dezimal 3584 bis 4095) gefüllt, der die Informationen des gewählten Sprites enthält. Falls noch kein Sprite definiert war, kann dabei auch allerlei Byte-Müll abgebildet werden, der uns aber vorläufig nicht erschüttern soll. In der linken oberen Ecke des Rasterbereiches meldet sich ein Kreuz: das ist der Spritecursor. Oben rechts neben dem Raster ist unser Sprite so abgebildet, wie es im Normalbetrieb später auf dem Bildschirm erscheinen wird.

Bevor wir auf die einzelnen Optionen des Sprite-Editors eingehen, noch eine Bemerkung. Alles bisher Beschriebene benutzt den Grafik-Bildschirm. Ebenso wie bei der Verwendung der hochauflösenden und der Mehrfarbengrafik wird dazu der Basic-Anfang nach $4000 verschoben, also über die Bit-Map.

Sehen wir uns nun die Möglichkeiten unseres Editors an:

CLR/HOME: Damit kann das Definitionsfeld gelöscht werden. Das befreit uns vom Byte-Müll.

HOME: Der Sprite-Cursor marschiert in die linke obere Ecke.

CURSOR-TASTEN

Damit wandert der Sprite-Cursor über das Rasterfeld.

RETURN: Der Cursor wird auf den Anfang der nächsten Zeile gesetzt.

A: Die Wiederholungsfunktion der Tasten 1 bis 4 kann durch einmaliges Drücken aus-, durch nochmaliges Drücken wieder eingeschaltet werden.

M: Einmal Drücken schaltet den Mehrfarbenmodus ein. Dadurch bekommen die Tasten 1 bis 4 eine neue Bedeutung.

Nochmaliges Drücken schaltet wieder auf den normalen Hochauflösungsmodus.

X: Ein Druck verdoppelt das Sprite in waagerechter Richtung, ein weiterer Tastendruck erzeugt wieder ein Sprite in normaler Breite.

Y: Mit dieser Taste passiert in der Senkrechten dasselbe wie in der Horizontalen bei der X-Taste.

ZIFFERNTASTEN 1 BIS 4

a) Im Normalmodus setzt
1: die aktuelle Hintergrundfarbe (0),
2-4: die aktuelle Vordergrundfarbe (1).

b)Im Mehrfarbenmodus setzt
1: die Hintergrundfarbe (00)
2: eine Multicolorfarbe (01)
3:dieaktuelleVordergrundfarbe(10)
4:dieandereMulticolorfarbe(11)

Zur Erläuterung:

– Im Normalmodus tritt jedes gesetzte Bit (= 1) als Bildpunkt in der Vordergrundfarbe auf, alle nicht gesetzten (= 0) erscheinen in der Hintergrundfarbe.

– Im Mehrfarbenmodus tragen die Bits paarweise zur Farbgebung bei. Deshalb ist hier der Spritecursor auch doppelt so breit.

CONTROL 1-8

Wählt die aktuelle Vordergrundfarben 1-8 aus.

COMMODORE 1-8

Leistet dasselbe mit den Farbcodes 9 bis 16.

Zwei Optionen werden merkwürdigerweise im Handbuch nicht erwähnt, die recht nützlich sind:

C: Das ist eine Kopierfunktion. Nach Tastendruck erscheint die Frage »Copy from ?«. Nach Eingabe einer Spritenummer wird das dazugehörige Muster in das aktuelle Sprite kopiert und dieses dabei überschrieben. Das ist ganz sinnvoll, wenn man mehrere ähnliche Sprites erzeugen möchte.

Was nicht im Handbuch steht

Dasselbe passiert auch, wenn man die Funktionstaste F1 betätigt.

Ist man einmal versehentlich auf diese Tasten geraten, dann kann man durch RETURN wieder in den normalen Editorzustand gelangen.

CONTROL C: Erlaubt das Umschalten zwischen den Sprites. Nach Betätigen dieser Tastenkombination verschwindet die aktuelle Spritenummer. Gibt man nun die gewünschte Nummer ein, schaltet sich das dazugehörige Sprite an.

Zwei Möglichkeiten zum Verlassen des Sprite-Editors sind vorgesehen:

STOP-Taste: Nach dem Betätigen der Taste verliert man alle neuen Eingaben im Definitionsfeld.

Gibt man anschließend eine Spritenummer ein, wird das dazugehörige Muster eingeladen.

Ein anschließendes RETURN führt zum Verlassen des Editors. Mit READY meldet sich der Textbildschirm zurück.

SHIFT-RETURN

Speichern des aktuellen Sprites im Bereich $0E00 - 0FFF.

Ebenso wie bei der STOP-Taste kann man danach durch Eingabe einer neuen Spritenummer das nächste Sprite zur Bearbeitung in den Raster holen.

Ein RETURN führt zum Verlassen des Sprite-Editors.

Interessant an SPRDEF ist, daß man diesen Befehl auch im Programm-Modus verwenden kann. Auf diese Weise kann man im Programm ein oder mehrere Sprites erstellen und anschließend - also nach Aussteigen mit STOP/RETURN oder SHIFT-RETURN/RETURN - läuft das Programm weiter.

Das also sind alle Möglichkeiten unseres Sprite-Editors: Leider fehlen einige nötige Optionen. So ist es nicht möglich, die frisch zusammengebauten Sprites in DATA-Zeilen abzulegen, die Multicolorregister müssen vor dem SPRDEF-Aufruf belegt werden, die Muster kann man nicht invertieren, drehen oder spiegeln.

SPRCOLOR

Im Normalmodus kann die Punktfarbe eines Sprites und auch die Farbe der Bitkombination 10 des Multicolormodus durch den SPRITE-Befehl bestimmt werden oder in SPRDEF durch die Kombinationen von Control- oder Commodoretaste mit einer Ziffer. Die Belegung der Multicolorregister mit Farbe - das entspricht den SPRDEF-Zifferntasten 2 und 4 -, also der Bitkombinationen 01 und 11 erfolgt durch den SPRCOLOR-Befehl. Die Verwendung von

SPRCOLOR A,B

führt in Multicolorregister 1 (Bit-Kombination 01 oder SPRDEF-Taste 2) zur Farbe A, in Multicolorregister 2 (Bit-Kombination 11 oder SPRDEF-Taste 4) zur Farbe B.

Somit ergibt der Befehl

SPRCOLOR 3,6

die Farben ROT und GRÜN, entsprechend den Bitkombinationen 01 und 11.

SPRITE

Dies ist zweifellos der wichtigste Befehl für ein schon vorhandenes Spritemuster. Er schaltet ein Sprite ein und legt seine Eigenschaften fest. Syntax:

SPRITE A,B,C,D,E,F,G

Keine Angst vor den vielen Parametern: Sie müssen nicht immer alle angeben. Zur Bedeutung der einzelnen Buchstaben:

A: Spritenummer (1 - 8)
B = 0: Ausschalten des Sprites
B = 1: Einschalten des Sprites
C: Farbcode für die Vordergrund-

farbe (das ist das Bitpaar 10 bei Multicolorsprites) (1 bis 16)
D: Priorität gegenüber Text:
0 = Sprite vor Text
1 = Text vor Sprite
Nebenbei: Die Priorität von Sprites untereinander wird durch die Sprite-Nummer geregelt: 1 vor 2 vor 3 ... vor 8.
E: X-Vergrößerung:
0 = normale Größe
1 = doppelte Größe
F: Y-Vergrößerung:
0 = normale Größe
1 = doppelte Größe
Sind sowohl E als auch F auf 1 gesetzt, dann erhält man ein vierfach ausgedehntes Sprite.
G: Modus.
0 = normales Hires-Sprite
1 = Multicolor-Sprite
Ein Beispiel soll die Wirkung des Befehls demonstrieren:
SPRITE 1,1,3,0,1,1,0

Dadurch wird Sprite 1 eingeschaltet. Es ist rot, erscheint vor Bildschirmzeichen und ist sowohl in horizontaler wie auch in vertikaler Richtung verdoppelt. Es handelt sich um ein normales (also Hires-Sprite). Nun soll das gleiche Sprite auf die normale Größe gebracht werden:
SPRITE 1,,,,0,0

Es genügt, die Werte anzugeben, die zu ändern sind. Alle anderen werden so übernommen, wie sie vorher festgelegt wurden. Die Spritennummer und die Kommata vor dem zu ändernden Wert sind obligatorisch.
MOVSPR

Das ist ein recht vielgestaltiger Geselle. MOVSPR kommt von "move sprite", also hat dieser Befehl mit der Positionierung und Bewegung von MOBs zu tun. Drei Varianten sind möglich:
a: MOVSPR n,X,Y

Das Sprite mit der Nummer n wird an den Ort mit den Koordinaten X,Y gesetzt. Als Bezugspunkt beim Sprite dient die linke obere Ecke (auch wenn diese unsichtbar ist). Bild 1 zeigt Ihnen das normale Sprite-Koordinatensystem, das in allen Grafik-Modi gültig ist.

Der sichtbare Bildschirm erstreckt sich von X = 24 bis 344 und in der Vertikalen reicht er von 50 bis 250. Für X und Y dürfen Werte zwischen 0 und 65535 eingesetzt werden. Bild 1 skizziert auch, bei welchen Koordinatenangaben ein Sprite voll sichtbar ist:

Links oben müssen dazu die Koordinaten (24/50) gewählt werden. Ein normales Sprite ist bei X = 320, ein in X-Richtung gedehntes bei X = 296 gerade noch voll sichtbar. Entsprechend gilt für die Y-Werte:
Normales Sprite bis Y = 229,
in Y-Richtung gedehntes bis Y = 208.

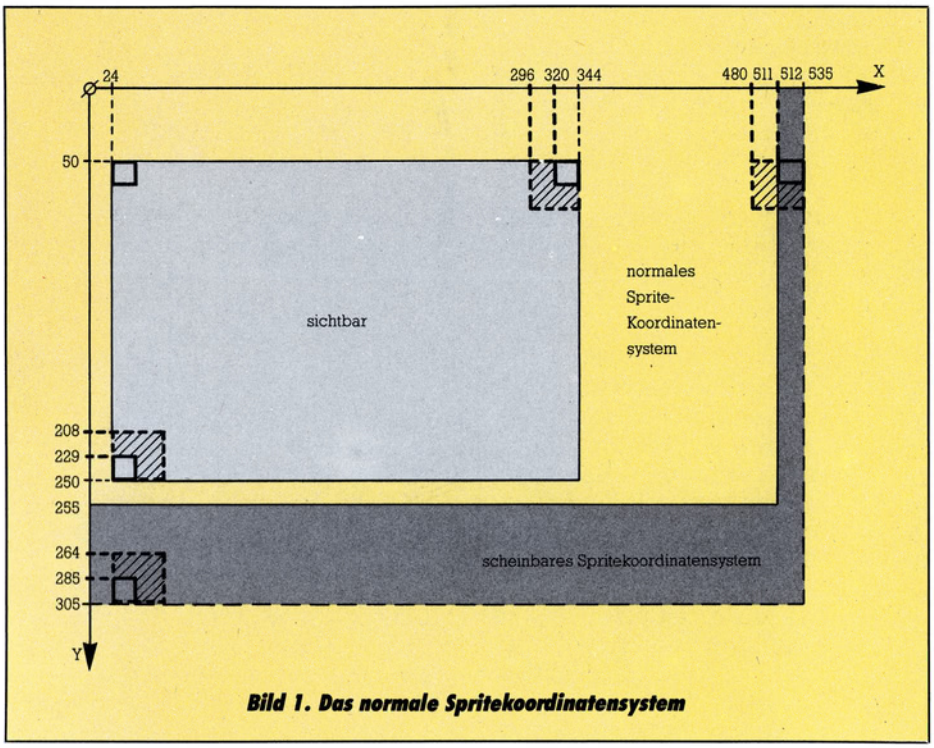

Bild 1. Das normale Spritekoordinatensystem

Größere Koordinatenwerte führen zum Herauswandern aus dem sichtbaren Bereich. Überschreiten die X- oder Y-Werte eine bestimmte Grenze, dann treten die MOBs auf der jeweils gegenüberliegenden Seite wieder ins Bild. Das ist ab X = 513 (bei gedehnten Sprites ab 481) und ab Y = 286 (bei gedehnten Sprites ab 265) der Fall. Eine weitere Erhöhung der Koordinaten führt dann zum Wandern über den Bildschirm, zum erneuten Verschwinden und schließlich Wiederauftauchen auf der anderen Seite und so weiter.

Vorhin war die Rede vom »normalen« Sprite-Koordinatensystem. Daraus kann messerscharf geschlossen werden, daß es auch noch ein anderes gibt, ein »nicht normales«. Das ist tatsächlich der Fall und es handelt sich nicht nur um eines, sondern um eine ganze Menge verschiedener Systeme. Haben Sie nämlich durch den SCALE-Befehl ein neues Hires- oder Multicolor-Bildschirmsystem definiert, dann beziehen sich X und Y auf dieses, was manchmal zu einiger Verwirrung führen kann. Sollte also die Gefahr bestehen, daß ungewollt aus einer früheren Programmphase ein anderes Bildschirmsystem gültig ist, dann kann durch SCALE 0 der normale Zustand hergestellt werden.
Hier noch ein Beispiel:

MOVSPR 1,300,100
setzt Sprite 1 an die Stelle (300/100) im jeweils gültigen Koordinatensystem.
b: MOVSPR n,+/-X,+/-Y

Damit kann man Sprite n relativ um + oder -X (und/oder Y) zur aktuellen Position verschieben. X und Y dürfen wieder Werte bis 65535 annehmen, wobei durch das Vorzeichen nun sogar eine Skalenbreite von -65535 bis +65535 möglich wird. Auch hier beziehen sich X und Y auf das jeweils durch SCALE definierte Koordinatensystem (ohne SCALE-Anwendung also auf das normale Sprite-System aus Bild 1).

Auf diese Weise kann – zum Beispiel in einer Schleife – das Sprite praktisch endlos über den Bildschirm wandern. Es tritt nämlich keine Fehlermeldung auf, wenn die Summe aus alter Position und Verschiebung größer als 65535 wird. Beispielsweise ist es ohne weiteres möglich (ob es sinnvoll ist, wäre eine andere Frage), folgende Kombination zu verwenden:
MOVSPR 8,100,65000:MOVSPR 8,100,+65000

Auch ein Unterlauf unter Null ist auf diese Weise erlaubt.

Wie Sie aus dem Beispiel erkennen können, kann die MOVSPR-Anweisung auch kombiniert verwendet werden:

MOVSPR 1,+10,100 verschiebt Sprite 1 um +10 Einheiten relativ zum vorangegangenen X-Wert und auf die Y-Koordinate 100.

MOVSPR 2,50,-20 läßt Sprite 2 auf die X-Koordinate 50 wandern und

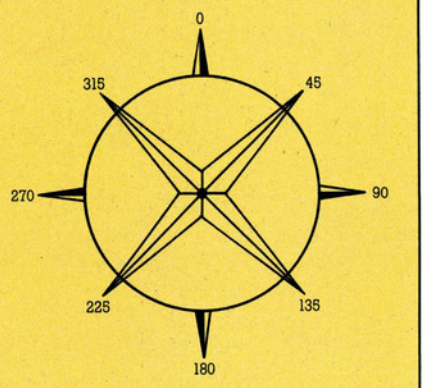

Bild 2. Die Kompaßrose: MOVSPR

gleichzeitig um 20 Y-Einheiten nach oben.
c: MOVSPR n,W#V

Diese Variante des Spritebewegungs-Befehls ist eine feine Sache: Unabhängig vom sonstigen Programmgeschehen wird das Sprite mit der Nummer n mit einem Richtungswinkel W und der Geschwindigkeit V über den Bildschirm gesteuert. Der Winkel W wird in Grad angegeben. Bild 2 zeigt die möglichen Richtungen, die der Kompaßrose entsprechen:

Ohne Fehlermeldung sind Eingaben möglich zwischen -65535 und +65535. Sinnvoll scheint das aber allenfalls in einer Schleife, die ständige Richtungswechsel durchführt, zumal negative Werte und Angaben größer als 33024 keine vernünftige Reaktion ergeben.

Auch für die Geschwindigkeit V sind Angaben zwischen -65535 und 65535 möglich. Als sinnvoll erweisen sich hier nur Werte zwischen 0 und 15. 0 stoppt die Bewegung und 15 ist die höchste erreichbare Geschwindigkeit. Für höhere V-Werte ergibt sich die Geschwindigkeit dann zu modulo(16). (Damit ist der Rest gemeint, der bei Division der eingegebenen Zahl durch 16 verbleibt. So ergibt V=30 dieselbe Geschwindigkeit wie V=14)

Eine Eigenart dieser MOVSPR-Variante ist es, daß die Wirkung dieses Befehls anhält, bis die Bewegung ausdrücklich durch MOVSPR n,W#0 auf Null gesetzt wird. Auch ein abgeschaltetes Sprite wandert weiter, was Sie leicht mal ausprobieren können.

Der Befehl
MOVSPR 1,90#15
läßt Sprite 1 mit Höchstgeschwindigkeit horizontal nach rechts über den Bildschirm jagen.

SPRSAV

Das ist ein sehr interessanter Befehl, dessen Anwendung wir später noch detailliert untersuchen werden. Zwei mögliche Varianten sind vorgesehen:
a: SPRSAV n,A$

Einem String (hier also A$) wird das Bitmuster des Sprite n zugeordnet.
Mittels SPRSAV 1,A$(2)
ordnet man das Bitmuster des Sprite mit der Nummer 1 dem Stringarrayelement A$(2) zu.
b. SPRSAV A$,n

Das ist der umgekehrte Weg: Das in A$ enthaltene Bitmuster definiert nun das Sprite mit der Nummer n.

Durch SPRSAV A$(2),8 wird das in A$(2) gespeicherte Muster in das Sprite Nummer 8 gelesen.

Anstelle von A$ kann jede Stringvariable - auch ein Array-Element - verwendet werden. Einige Möglichkeiten, die auf diese Weise gegeben sind: "Klonen" eines Sprites, Definieren von mehr als acht Sprites (jeweils acht sind gleichzeitig auf dem Bildschirm aktivierbar), schneller Austausch von Spritemustern.

COLLISION

Was im C64-Modus dem Assembler-Programmierer vorbehalten bleibt oder nur durch aufwendige POKE- und PEEK-Operationen realisiert werden kann, ist im C 128-Modus mit diesem und dem folgenden Befehl möglich:
COLLISION A,NNNN
fängt drei Typen von Ereignissen ab und setzt die Programmbearbeitung in Zeile NNNN fort. A ist dabei eine Typkennung:
1: Sprite/Sprite-Kollision
2: Sprite/Text-Kollision
3: Lichtgriffelaktivierung

Das Serviceprogramm ab NNNN ist wie ein Basic-Unterprogramm zu behandeln, also auch durch RETURN abzuschließen. Die weitere Bearbeitung geschieht dann an der Stelle, wo bei dem auslösenden Ereignis unterbrochen worden ist.

COLLISION stellt lediglich fest, welcher Typ stattgefunden hat. Bei Spritezusammenstößen kann also damit nicht die Spritenummer identifiziert werden. Es ist erlaubt, mehrere Kollisionstypen gleichzeitig zu aktivieren. Eine Aktivierung innerhalb eines Unterbrechungs-Service-Unterprogrammes ist nicht möglich: Es findet nur die Bearbeitung eines Typs zur Zeit statt. Falls also mehrere aktiviert sind, sollte einer der ersten Befehle im Unterprogramm das Abschalten aller Kollisionsabfragen sein mittels COLLISION A (ohne Zeilennummer). Am Ende der Routine kann dann COLLISION A,NNNN wieder eingeschaltet werden.

Nicht immer ist es sinnvoll, so zu verfahren (also die Kollisionsabfrage am Ende der Routine wieder einzuschalten): Falls beispielsweise der Typ 1 (Sprite/Sprite-Zusammenstoß) als Auslöser definiert ist, muß bedacht werden, daß die Überlappung zweier Sprites einige Zeit andauert und deshalb das Unterprogramm mehrfach angesteuert wird. Im einem Testprogramm (Listing 2) ist dieser Effekt deutlich an den vielen Textwiederholungen zu erkennen.

Unter welchen Umständen wird eine Kollision erkannt? Immer dann, wenn sich sichtbare Teile von Sprites gegenseitig oder mit Bildschirmobjekten (bei Typ 2) überlagern. Abgeschaltete Sprites werden nicht berücksichtigt, wohl aber Zusammenstöße außerhalb des sichtbaren Bereiches (das Handbuch behauptet das Gegenteil).

Probieren Sie doch einmal, in Zeile 40 von Listing 1 die Spritepositionierung:
40 MOVSPR 7,345,100:MOVSPR 8,345,200

Nach dem Start ist keines der beiden Sprites mehr auf dem Bildschirm zu sehen, Kollisionen werden aber gemeldet.

Zwei Aspekte verdienen noch Erwähnung: Zum einen darf man ein Programm, in dem der COLLISION-Befehl verwendet wird, nicht durch STOP unterbrechen. Tut man es trotzdem, dann funktioniert nach einem CONT die Kollisionsabfrage nicht mehr. Zum anderen: Natürlich können alle Sprite-Befehle auch ohne 40-Zeichen-Bildschirm betrieben werden, man sieht nur nichts. Das kann bei COLLISION ganz rätselhafte Effekte erzeugen. So wären Spiele denkbar, die ein Sprite per Joystick durch Hindernisse hindurchbewegen. Das ganze geschieht im Dunkeln (also mit abgeschaltetem oder nicht vorhandenem 40-Zeichen-Bildschirm) und lediglich die Reaktion auf eine Kollision erfolgt auf dem 80-Zeichen-Bildschirm. Der Phantasie sind keine Grenzen gesetzt.

BUMP

Der Name sagt's bereits: Auch hier geht es um Zusammenstöße. Mit dem BUMP-Befehl können wir einfach feststellen, welche Sprites in eine Kollision verwickelt sind:
BUMP(A)

erschließt über die Typkennung A zwei Sorten von Zusammenstößen.

A = 1: Sprite/Sprite-Kollision
A = 2: Sprite/Text-Kollision.

Der Befehl hat lediglich die Aufgabe, den Inhalt eines speziellen Kollisionsregisters auszulesen. Das allerdings macht er so gründlich, daß dieses Register hinterher gelöscht ist. Es empfiehlt sich daher, diesen Wert sogleich in eine Variable zu speichern:

V = BUMP(1)

Die Zahl, die auf diese Weise erhalten wird, muß allerdings erst entschlüsselt werden (siehe da, ein alter Bekannter aus dem C64-Modus), denn jedem Sprite ist ein Bit zugeordnet: Sprite 1 hängt mit Bit 0, Sprite 2 mit Bit 1 zusammen und so weiter bis zu Sprite 8, welches mit Bit 7 geht. Findet nun die Kollision statt, dann schalten die Bits der kollidierten Sprites auf »1«. Bild 3 zeigt Ihnen diese Zusammenhänge und einige Rechenbeispiele.

Bits:	7	6	5	4	3	2	1	0	
Bitwerte:	128	64	32	16	8	4	2	1	
Sprites:	8	7	6	5	4	3	2	1	
Beispiele:	0	0	0	0	0	0	1	1	= 3, also Sprites 1 und 2
	0	0	0	0	0	1	0	1	= 5, also Sprites 1 und 3
	0	0	0	0	1	0	0	1	= 9, also Sprites 1 und 4
	1	0	0	0	0	0	0	1	= 129, also Sprites 1 und 8
	1	1	0	0	0	0	0	0	= 192, also Sprites 7 und 8

Bild 3. Das Ergebnis der BUMP(A)-Abfrage

Ein Supercrash unter Beteiligung aller acht Sprites würde dann also die Zahl 255 ergeben. Ein kleines Testprogramm (Listing 3) soll das Vorgehen dabei illustrieren.

Besonders die Zeilen 90 und 100 sind sicherlich für Sie interessant, weil man mit ihnen die Nummern der beteiligten Sprites feststellen kann.

Auch hier gilt, was schon bei COLLISION bemerkt wurde: Auch Zusammenstöße außerhalb des sichtbaren Bildschirmes werden registriert und das auch ohne 40-Zeichen-Bildschirm. Ebenso wie dort muß beachtet werden, daß ein Zusammenstoß einige Zeit in Anspruch nimmt, also die Reaktion darauf meistens mehrfach erfolgt.

Im Gegensatz allerdings zu COLLISION, das interrupt-gesteuert abläuft, muß die Abfrage des BUMP-Wertes mehrfach erfolgen. Der Befehl ist sowohl ohne als auch mit COLLISION einsetzbar. Im letzteren Fall dient er als Ergänzung im Unterprogramm zur Feststellung der Spritenummern.

RSPCOLOR

Damit sind Informationen zu den aktuellen Multicolor-Register-Inhalten zu bekommen:

RSPCOLOR(A)

liefert bei

A = 1: Farbcode in Multicolorregister 1 (das ist das, welches der Bitkombination 01 zugeordnet ist)

A = 2: Hier wird der Inhalt des anderen Multicolorregisters ausgegeben, das zu der Bitkombination 11 gehört.

RSPPOS

Sollte uns die aktuelle Position oder Geschwindigkeit eines Sprites interessieren, dann können wir das durch

RSPPOS(n,A)

erfahren. Dabei ist n die Spritenummer und A wieder eine Kennung mit folgender Zuordnung:

A = 0: aktuelle X-Koordinate
A = 1: Y-Koordinate
A = 2: gerade vorhandene Geschwindigkeit

Leider gibt es da eine kleine Unstimmigkeit gegenüber dem Befehl MOVSPR: Gleichgültig, welches Sprite-Koordinatensystem wir dort durch SCALE festgelegt haben, RSPPOS liefert immer die Werte des normalen Systems. RSPPOS(0) liegt daher immer zwischen 0 und 511, RSPPOS(1) immer zwischen 0 und 255. Sollten Sie bei normalem System mittels MOVSPR höhere Werte eingegeben haben, dann wird hier immer modulo(512) für die X- und modulo(256) für die Y-Koordinate ausgegeben.

Auch die Sache mit der aktuellen Geschwindigkeit hat einen Haken: Falls Sie nämlich den Sprite mal auf andere Weise als durch MOVSPR n,W#V bewegen, erhalten Sie den Wert 0 als Ausgabe von RSPPOS(2).

RSPRITE

Hier haben wir das Pendant zum SPRITE-Befehl vorliegen. Alle dort verwendeten Parameter können hier durch

RSPRITE(n,A)

abgefragt werden. n ist wieder die Spritenummer, A eine Kennung mit den Zuordnungen:

A = 0: Sprite ein- (=0) oder ausgeschaltet (=1)
A = 1: Farbcode des Sprites (Im Multicolormodus Farbe der Bitkombination 10)
A = 2: Sprite vor (=0) oder hinter (=1) Text.
A = 3: In X-Richtung gedehnt (=1) oder nicht (=0).
A = 4: In Y-Richtung gedehnt (=1) oder nicht (=0).
A = 5: Multicolormodus (=1) oder normales Hires-Sprite (=0)

JOY

Dieser Befehl hat zwar nicht unbedingt etwas mit den Sprites zu tun, wird aber häufig zum Steuern der Sprites eingesetzt und kommt daher an diese Stelle. Man kann damit auf einfache Weise den Zustand der Joystickports abfragen:

JOY(A)

fragt bei A = 1 den Port 1, und bei A = 2 den Port 2 ab.

Bevor wir uns die Bedeutung der Abfrageergebnisse ansehen, noch eine Warnung: Offensichtlich entspricht der Feuerknopf in Port 1 der Taste F8, so daß im Direktmodus immer der Maschinensprachemonitor angesprungen wird, wenn man den Feuerknopf drückt. Beim Port 2 gibt es dieses Problem nicht.

Die mittels JOY ausgelesenen Werte haben folgende Bedeutung:

0 = Ruhestellung
1–8 = Richtungen (siehe Bild 4)
128 = Ruhestellung, Feuerknopf gedrückt
129–136 = Richtungen mit gedrücktem Feuerknopf (siehe Bild 4)

Auf diese Weise können wir bequem Sprites mit dem Joystick steuern:

100 ON JOY(2) GOSUB 300,310,320,330,...

Im Testprogramm (Listing 4) ist eine mögliche Variante dazu gezeigt.

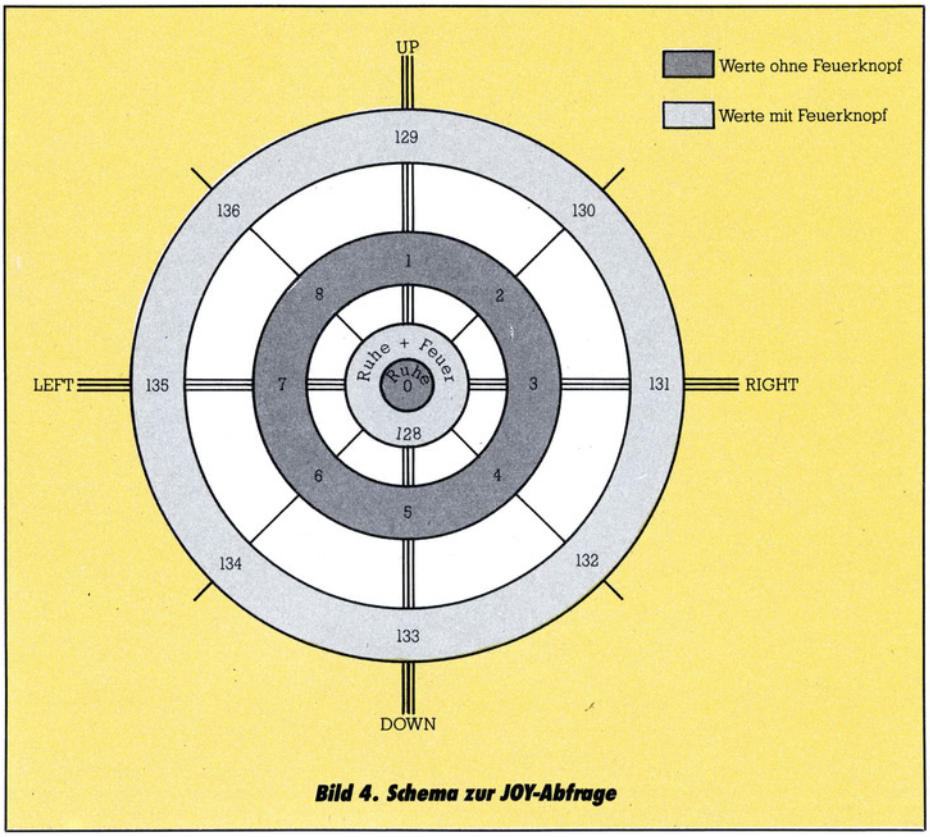

Bild 4. Schema zur JOY-Abfrage

Damit haben wir alle im Basic 7.0 enthaltenen Sprite-Befehle behandelt. Allerdings sind noch einige Fragen offen geblieben, die uns nun weiter beschäftigen werden.

Wie kann man Sprites der Nachwelt erhalten?

1. Sprites im Speicher

a. Sprite-Daten-Speicher

Schon beim SPRDEF-Befehl haben wir diesen Speicherbereich zwischen 3584 und 4095 ($0E00 bis 0FFF) erwähnt, in dem die Spritedaten landen, wenn ein Sprite durch Shift-Return gespeichert wird. Pro MOB befinden sich dort - in der Reihenfolge der Spritenummern - je 63 Byte und eine Endmarkierung (das ist ein 0-Byte). Verfügbar bleiben diese Daten, bis sie überschrieben oder der Computer abgeschaltet wird.

b. Sprite-Strings

Noch flüchtiger ist die Existenz der Daten im String, der durch SPRSAV gebildet wird. Schon ein CLR macht der Kunst den Garaus. In jeweils 67 Byte (die letzten vier enthalten immer die Werte 23, 0, 20, 0) finden sich dort die Bitmuster. Sollten Sie einmal versuchen, so einen String (z.B. A$) durch PRINT A$ auf dem Bildschirm zu zeigen, dann müssen Sie sich auf allerhand Überraschungen gefaßt machen: Alle Bytewerte von 0 bis 255 können auftreten und wirken so, als wären sie mittels CHR$(Wert) zur Ausgabe aufgerufen worden. Wenn also der Wert 147 zufällig dabei sein sollte, dann wird der Bildschirm gelöscht, etc. Jedes RUN führt übrigens ebenfalls ein CLR aus. Auch das ist daher nicht die richtige Methode, die mühselig konstruierten Sprites etwas länger am Leben zu erhalten.

Sowohl im Speicher als auch im String haben die Sprite-Daten genau dasselbe Format (bis auf die letzten Bytes, die im einen Fall aus einer 0, im anderen aus den Zahlen 23,0,20,0 bestehen). Das können Sie selbst nachprüfen am Programm "VERGLEICH SPRDT" (Listing 5).

Nach dem Start werden zunächst die Spritedaten von der Diskette geladen und dann in Strings eingelesen. Anschließend erscheinen auf dem Bildschirm Sprite für Sprite die Bytewerte sowohl aus dem Speicher als auch aus dem String.

2. Sprites auf Diskette und Kassette.

a. BSAVE

Eine elegante Methode zur Verewigung unserer Sprites ist das Speichern des Spritespeichers mittels des BSAVE-Befehls. Man schreibt dazu:

BSAVE"Name",ON B0,P3584 TO P4095

Damit sind dann alle acht Sprites erfaßt. Leider arbeitet dieser Befehl nicht mit der Kassettenstation, denn es sind nur Gerätenummern von 4 bis 15 zulässig. Das Wiedereinladen geschieht dann (wie in den hier vorgestellten Beispielprogrammen) mittels:

BLOAD"Name",ON B0

b. Vom Monitor aus

Damit ist es nun auch Benutzern der Datasette möglich, den Spritespeicher auf Kassette zu sichern. Der Monitor verfügt über ein Kommando S zum Speichern beliebiger Speicherbereiche. Mittels MONITOR oder der Funktionstaste F8 schalten Sie den Monitor an, dann verwenden Sie:

S"Name",01,+3584,+4095
(Kassette) oder
S"Name",08,+3584,+4995
(Diskette).

Mit Hilfe des L-Kommandos im Monitor kann solch ein File dann problemlos wieder geladen werden:

L"Name",01(oder 08),+3584

c. Als sequentielles File

Hat man die Sprites in Strings abgelegt (das dürfen dann auch mehr als acht sein), dann kann man sich der üblichen Techniken zur Speicherung in sequentiellen Dateien bedienen. Allerdings kann es manchmal dabei Schwierigkeiten mit bestimmten Byte-Inhalten geben. So ist es durchaus möglich, daß ein Byte zufällig den Inhalt 13 (also einem RETURN entsprechend) hat, was zu Störungen beim Wiedereinlesen führen kann. In solchen Fällen könnte man die einzelnen Bytes (wie im Programm VERGLEICH SPRDT geschehen) in die ASCII-Zahlen wandeln und in dieser Form als SEQ-File speichern. Diese Möglichkeit werden wir nicht weiter beschreiben, weil die Speicherung per Monitor oder BSAVE schneller und effektiver erscheint.

3. Für ein Listing

Viele Programme werden nicht in Form von Disketten- oder Kassettenfiles, sondern einfach auf dem Papier weitergegeben. Dieses Problem lösen bessere Sprite-Editoren durch eine Funktion, die alle Werte in DATA-Zeilen an ein Programmende anhängt. In SPRDEF existiert diese Möglichkeit leider nicht, was uns dazu zwingt, selbst die Initiative zu übernehmen.

a. Aus dem Speicher in DATAs

Dazu haben wir ein kleines Programm »SPRITEDATAS« gestrickt (Listing 6), das Sie mit der MERGE-Funktion an Ihr eigenes fertiges Spriteprogramm anhängen können. Noch eine Warnung: Wenn Sie SPRITDATAS abgetippt haben, speichern Sie es unbedingt vor

einem RUN ab, denn es verabschiedet sich am Ende des Programmlaufes aus Ihrem Basicspeicher. Nun also zur Verwendung des Programmes. Man startet es durch RUN63000. Das Auslesen des Sprite-Datenspeichers geschieht im FAST-Modus (der 40-Zeichen-Bildschirm verabschiedet sich vorübergehend), danach erscheinen jeweils zwei Programmzeilen (mit Zeilennummern ab 63020) und darunter ein GOTO 63009. Außerdem meldet sich ein BREAK und READY. Mittels der HOME-Taste und 3 aufeinanderfolgenden RETURNs übernehmen Sie die Zeilen ins Programm und erzeugen den nächsten Bildschirm, wo das Spielchen dann ebenso weitergeht. Insgesamt machen wir das achtmal (für 8 Sprites). Zu guter Letzt wird in Zeile 63040 noch eine Einleseschleife übernommen und es meldet sich der Befehl DELETE 63000-63011. Haben Sie auch das mit HOME und zwei RETURNs übernommen, dann ist der DATA-Generator gelöscht und an Ihr Programm wurde die gesamte DATA-Sequenz angehängt. Als letzte Zeile finden Sie beim Listen noch die Einleseschleife, zu der Sie nun nur noch an passender Stelle Ihres Programmes mittels GOSUB 63040 springen müssen.

Es gibt sicherlich elegantere Möglichkeiten, DATA-Zeilen zu generieren. So könnten wir auch den Tastaturpuffer verwenden. Das aber überlassen wir Ihrer Schöpferkraft.

b. Aus den Strings in DATAs

Auch das Übertragen der Stringinhalte in DATA-Zeilen ist eine Lösungsmöglichkeit unseres Problems. Sie ist sogar sehr verlockend, weil wir damit auf einen Streich mehr als acht Sprites ins Listing schreiben könnten. Die Programmierung wird aber etwas komplizierter, weil hier nicht mehr der ganze Inhalt des Sprite-Strings in einen Zeilen-String gelesen werden kann. Es gibt eine Reihe von Strategien um das Problem zu lösen (mehr oder weniger einfache, was teilweise mit dem Auslesen der DATAs zusammenhängt), auch hier sind Sie gefordert!

Koppeln von Sprites

Sollten Sie ein komplexeres grafisches Gebilde erzeugen wollen als es mit einem Sprite möglich ist, dann können Sie auch mehrere Sprites koppeln. Am Beispiel von vier Sprites zeigt Ihnen Bild 5 die dazu nötigen Koordinatenwerte:

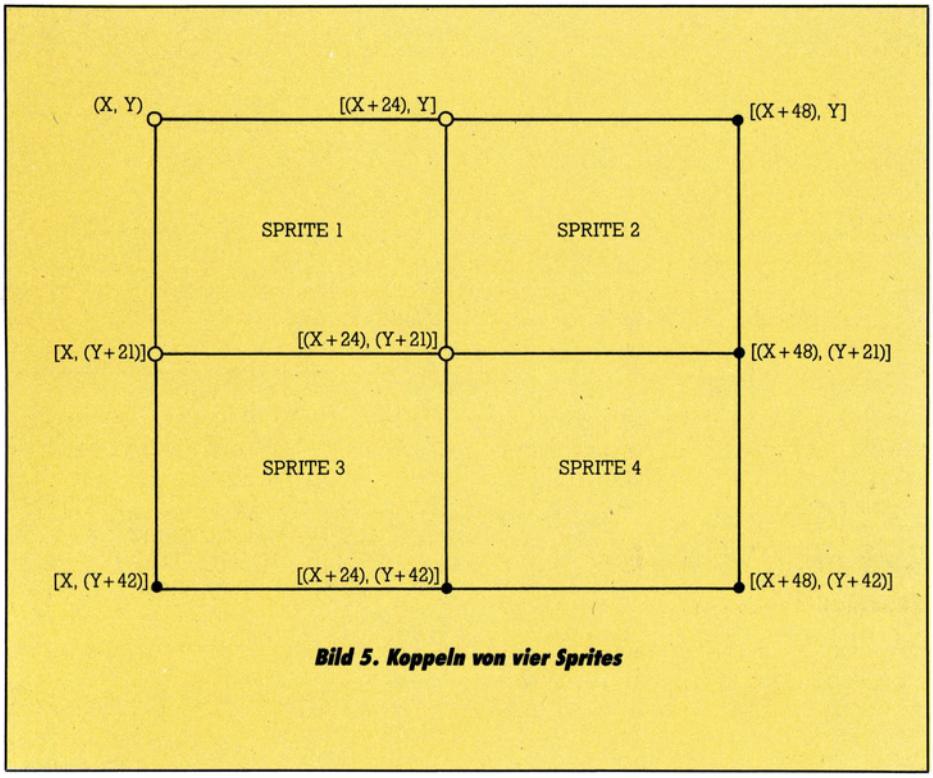

Bild 5. Koppeln von vier Sprites

Das Prinzip ist einfach: Weil jedes Sprite in X-Richtung 24 Bit breit ist, ergibt sich für die X-Koordinate des Nachbarsprites bei nahtloser Verbindung ein um 24 größerer Wert. 21 Bit mißt ein MOB in der Vertikalen, wodurch sich Y+21 für ein unterhalb des ersten gelegenes Sprite berechnet als Y-Koordinate. Das Programm »VIER SPRITES« (Listing 7) demonstriert die Verhältnisse.

Zunächst wächst ein grafisches Objekt aus vier Sprites zusammen. Dieses kann dann mit dem Joystick in Port 2 (nach Druck auf den Feuerknopf) über den Bildschirm gesteuert werden (Zeile 300). Ein weiterer Druck auf den Feuerknopf beendet das Programm. Wenn Sie bei bestimmten Richtungen genau hinsehen, werden Sie eine leichte Unstimmigkeit in der Bewegung feststellen. Bei noch größeren Objekten scheint Basic für diese Steuerung doch etwas zu langsam zu werden und man muß auf Maschinensprache umsteigen um eine flüssige Bewegung zu erreichen.

Mehr als acht Sprites

Hier soll nicht die Rede sein von trickreichen Anwendungen der Unterbrechungsprogrammierung per Assembler, die es tatsächlich zuläßt, mehr als acht Sprites GLEICHZEITIG auf dem Bildschirm zu zeigen. Vielmehr geht es uns um die schnelle, nacheinander ausgeführte Abbildung. Sehen Sie sich dazu zuerst einmal das Programm »SPRITE-TRICK« (Listing 8) an. Hier liegen acht Spritemuster vor, die nacheinander – mit programmierter Verzögerung, weil's sonst zu schnell ginge – auf derselben Bildschirmstelle gezeigt werden. Es ergibt sich ein kleiner Trickfilm.

Nun sind acht Teilbilder ein etwas ärmlicher Bewegungseffekt. Um wirklich längere Passagen zu zeigen, müßten wir anders verfahren. Das soll im Programm »24 SPRITES« demonstriert werden.

Das Geheimnis – Sie haben es sicherlich schon längst vermutet – liegt im SPRSAV-Befehl begründet. 24 Spritemuster wurden hier in ein Stringarray eingelesen (Zeilen 20 bis 80). Während die Sprites auf dem Bildschirm aktiv sind (die Aktivierung erfolgt schon in den Zeilen 130 bis 150), wechseln wir durch die Zuordnung anderer Spritemuster aus den Strings ihre Erscheinungsformen. Das geschieht in der DO...LOOP Schleife ab Zeile 170. Hier geschieht das – wegen des hübschen Mustereffektes – zyklisch (durch Drücken von »←« können Sie das Programm beenden), man kann sich aber ohne weiteres vorstellen, wie ein oder mehrere große Arrays gebildet und dann fortlaufend deren Spritemuster verwendet werden können. Gleichzeitig sind die sich ändernden Sprites noch beweglich...

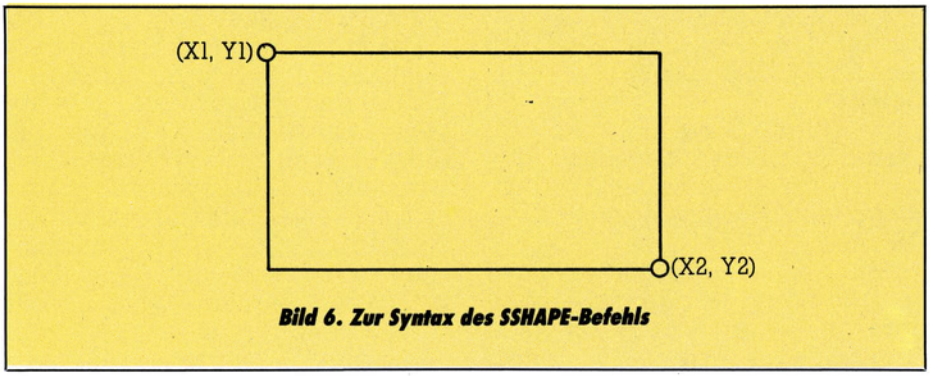

Bild 6. Zur Syntax des SSHAPE-Befehls

Sie sehen, daß kaum Grenzen gesetzt scheinen, allerhand Interessantes auf dem Bildschirm ablaufen zu lassen.

Noch ein paar Sprite-Besonderheiten

1. Sprites und Splitscreens

Beim C 128 haben wir ja in den Grafik-Modi 2 und 4 die Möglichkeit, sowohl hochauflösende (oder aber Multicolor-) Grafik als auch Text auf einem Bildschirm gleichzeitig darzustellen. Damit ist nicht etwa der CHAR-Befehl gemeint, sondern durch eine spezielle Technik, die den Rasterzeileninterrupt ausnutzt, wird der Bildschirm tatsächlich in zwei verschiedenen Modi betrieben.

Wie verhalten sich nun die Sprites auf solch einem Bildschirm? Wie es Kobolden ansteht, scheren sie sich überhaupt nicht darum! Probieren Sie mal das Programm VIER SPRITES mit einer kleinen Änderung aus: In die Zeile 130 fügen Sie noch ein GRAPHIC 2,0,12 (oder für den Multicolormodus GRAPHIC 4,0,12) und starten dann. Ohne Reaktion von Seiten der Sprites können Sie diese über die in der Bildmitte liegende Modusgrenze hinwegsteuern. Allerdings reagiert der Bildschirm mit einem penetranten Flattern der Grenzlinie, wenn das Sprite genau darauf liegt.

Falls Sie dasselbe mit dem Grafik-Modus 4 (also GRAPHIC 4,0,12) probieren, würde es uns interessieren, ob auch Ihr Computer nach einiger Zeit mit irgendeiner Fehlermeldung aussteigt. Beim Modell des Autors hat er sich so verhalten. Wenn er sich dann die als fehlerhaft gemeldete Zeile listen ließ, befanden sich allerlei merkwürdige – anscheinend zufällig erzeugte – Fehler darin. Zuvor aber war die Zeile fehlerfrei gewesen! Man sollte offensichtlich diese Möglichkeit mit etwas Vorsicht gebrauchen.

2. Sprites und Windows.

Ebensowenig wie sich Sprites um den Splitscreen kümmern, scheren sie sich um Windows. Probieren Sie es mal aus, indem Sie in Zeile 130 des Programmes »VIER SPRITES« noch ein Window definieren (zum Beispiel mit WINDOW 5,5,20,10). Unbeirrt von allen Einschränkungen, denen unser Text unterworfen ist (hier die Aufforderung zum Benutzen des Port 2), läßt sich unser Spritegebilde kreuz und quer verschieben.

Die Sprites an sich sind damit zunächst einmal durchleuchtet. Legen wir sie also für eine Weile beiseite und wenden wir uns den anderen Grafikobjekten des C 128 zu, den Shapes.

Was sind Shapes?

Wie so vieles aus der Computerkultur stammt auch dieses Wort aus dem angelsächsischen: Shape heißt ins Deutsche übersetzt soviel wie Form, Gestalt, Umriß. Hier bezeichnet Shape einen genau definierten Bildschirmausschnitt, der gespeichert und wiederverwendet werden kann. Wie das mit dem C 128 geschieht, soll nun erklärt werden.

SSHAPE

Dies ist der Befehl, mit dem ein Bildschirmbereich in einen String eingelesen werden kann. Seine Syntax ist

SSHAPE A$,X1,Y1,X2,Y2

A$ ist hier dann eine beliebige Stringvariable, auch ein Array kann Verwendung finden. X1 bis Y2 bezeichnet die Eckkoordinaten des Rechteckes, dessen Inhalt als Shape definiert werden soll. Bild 6 soll das etwas verdeutlichen:

(X1,Y1) gehören zum linken oberen und (X2,Y2) zum rechten unteren Eckpunkt. Es ist auch möglich, X2 und Y2 wegzulassen. In diesem Fall werden durch den SSHAPE-Befehl einfach die aktuellen Koordinaten des Grafik-Cursors eingesetzt.

Falls es noch unklar sein sollte: Im Gegensatz zu Sprites, die jedem Bildschirm-Modus trotzen (also sowohl als Hires- oder Multicolorsprites im Text-, als auch im Hochauflösungs- und im Multicolormodus auftauchen können), sind Shapes fest mit dem aktuellen Modus verbunden.

Ein Shape ist ein Grafikobjekt, kann daher auch nur in den Grafikmodi auftreten und definiert werden. Also in denen, die durch GRAPHICn erzeugt werden, wobei n von 1 bis 4 geht. Außerdem ist ein Multicolorshape auch nur im Multicolormodus ein solches. Schaltet man beispielsweise mittels GRAPHIC1 um in den normalen Hochauflösungsmodus, wird auch unser Shape nur als hochaufgelöstes sichtbar.

Wie wir uns aus all dem schon fast denken können, ist das Koordinatensystem, auf das sich die Angaben X1 bis Y2 beziehen, das im jeweiligen Modus gültige. Im Normalfall also im Hires-Modus X von 0 bis 319 und Y von 0 bis 199. Im Multicolormodus geht X nur von 0 bis 159.

A$ ist ein String. Für Strings gilt beim C 128 die Begrenzung auf maximal 255 Byte. Ein durch SSHAPE in einen String zu schreibendes Gebiet darf also eine gewisse Größe nicht überschreiten. Bezeichnen wir als

DX = X2 – X1

und als

DY = Y2 – Y1

dann gilt für die Bytemenge im String im Hochauflösungsmodus die Formel:

Lh = INT((ABS(DX) + 1)/8 + .99)*(ABS(DY) + 1) + 4

Im Multicolormodus gilt stattdessen die Gleichung:

Lm = INT((ABS(DX) + 1)/4 + .99)*(ABS(DY) + 1) + 4

Die Addition von 4 am Ende der Formeln ergibt sich durch vier Byte, die den Schluß des Shape-String bilden und Steuergrößen enthalten.

Natürlich können Sie jedesmal, wenn Sie ein Shape erstellen, von dem Sie argwöhnen, es könne zu groß sein für den String, unsere Formeln anwenden. Eine Hilfe soll Ihnen Listing 9 »SSHAPE-TABELLE« geben. In der vorliegenden Form druckt es eine Liste aus, in deren linker Spalte Werte für eine Y-Differenz stehen. Rechts daneben sind die maximal zulässigen DX-Werte angegeben sowie die sich ergebende Stringlänge. Auf diese Weise können Sie – ausgehend von DY – mit einem Blick erkennen, wie groß DX sein darf. Es empfiehlt sich, noch drei weitere Tabellen auszudrucken. Da ist zunächst einmal

interessant, wie der Maximalwert von DY bei gegebenem DX aussieht. Zu diesem Zweck ist lediglich die Reihenfolge des Ausdruckes und der Rechenoperationen zu ändern:

In Zeile 20 kommt dann X-DIFF vor Y-DIFF, Zeile 40 enthält X=0, in Zeile 50 schreiben wir DO UNTIL X=320, in der nächsten Zeile 60 steht: X=X+1:Y=0. Zeile 70 wird zu DO UNTIL Y=200 und Zeile 80 zu Y=Y+1. Schließlich wird auch in Zeile 90 noch die Reihenfolge des Ausdruckes umgestellt zu PRINT #1,X1,Y1,Z.

Diese beiden Tabellen beziehen sich auf Shapes im Hochauflösungsmodus. Zwei weitere Tabellen für den Multicolormodus bekommen Sie, wenn Sie in Zeile 30 die vorhin angegebene Berechnungsfunktion zu diesem Modus einfügen. Nur die Zahl 8 muß dazu in eine 4 verändert werden.

Sollte dann, trotz all dieser Berechnungsmöglichkeiten – man sieht ja nicht jedesmal in die Tabellen – der zu speichernde Shape größer werden als 255 Byte, dann meldet unser Computer einen STRING TOO LONG ERROR.

Einen Aspekt zum SSHAPE-Befehl müssen wir noch besprechen, weil es da einige Verwirrung geben kann: Wie reagiert SSHAPE auf ein durch SCALE verändertes Bildschirmsystem? Die Koordinatenangabe bei SSHAPE muß dann ebenfalls in den aktuell gültigen Werten geschehen. Man muß sich also nicht um die Organisation des Koordinatensystems kümmern, denn durch SSHAPE wird automatisch die Umrechnung auf den normalen Bildschirm vorgenommen. Lediglich bei der Berechnung der Stringlänge müssen einige Korrekturen eingefügt werden.

Nehmen wir an, wir hätten durch SCALE 1,XM,YM ein System definiert, dann liegen (im Hochauflösungsmodus) die Vergrößerungsfaktoren

FX = XM/320
FY = YM/200

vor und die Beziehungen für die Stringlänge lauten dann:

Hochauflösungsmodus:

```
Lh=INT((ABS(DX*320/XM)+1)/8
+.99)* (ABS(DY*200/YM)+1)+4
```

Multicolormodus:

```
Lm=INT((ABS(DX*160/XM)+1)/4
+.99)* (ABS(DY*200/YM)+1)+4
```

DX und DY sind dabei im aktuellen Koordinatensystem geltende Werte.

Begnügen wir uns nun mit dem SSHAPE-Befehl (es gäbe noch einiges zu untersuchen) und wenden wir uns dem Gegenstück, nämlich dem GSHAPE-Befehl zu.

GSHAPE

Ein durch SSHAPE im String gespeichertes Bild kann durch diesen Befehl nun wieder auf dem Bildschirm dargestellt werden:

GSHAPE A$,X,Y,M

A$ ist der uns nun schon bekannte Speicherstring, der durch SSHAPE definiert wurde. Es kann jede Stringvariable oder ein String-Array-Element verwendet werden.

X,Y sind die Koordinaten im aktuellen System, an die die linke obere Ecke unseres Shapes gelegt werden soll.

Interessant ist die Angabe von M. Hier dreht es sich um einen Darstellungsmodus, von dem es hier vier Möglichkeiten gibt:

M = 0: Unser Shape wird genauso abgebildet, wie es definiert wurde. Schon an dieser Stelle auf dem Bildschirm vorhandene Objekte werden überdeckt. Dasselbe ergibt sich, wenn wir diesen Parameter einfach weglassen.

M = 1: Das Shape wird invertiert abgebildet. Auch hier überdeckt das Shape vorhandene Objekte.

M = 2: OR-Modus: Bildpunkte werden sichtbar, wenn sie zum Shape oder zu vorhandenen Objekten gehören. Beide Bilder überlagern sich so.

M = 3: AND-Modus:Bildpunkte werden nur dann gesetzt, wenn sie sowohl zum Shape als auch zum Bildschirmobjekt gehören.

M = 4: EOR-(oder XOR-)Modus: Bildpunkte werden nur dann gesetzt, wenn das entsprechende Bit für Shape und Bildschirmobjekt ungleich ist.

Die Modi 2, 3, 4 sind etwas schwer vorstellbar. Zur Verdeutlichung finden Sie hier ein kleines Demonstrationsprogramm namens »GSHAPE-MODI« (Listing 10). Hier wird zuerst ein Shape definiert, dann der gesamte Bildschirm quergestreift und schließlich das soeben erstellte Shape in allen fünf Modi darauf dargestellt. Die Wirkung ist auf diese Weise ganz gut zu erkennen, besonders bei den Modi 2, 3 und 4.

Der Modus 4 ist aufgrund der EOR-Behandlung interessant: Wird nämlich auf dieselbe Stelle noch einmal das Shape im Modus 4 gezeichnet, dann verschwindet es ganz und der Bildschirmhintergrund ist wieder vorhanden. Will man Shapes bewegen, ohne Bildschirmobjekte zu zerstören, dann ist diese Darstellungsweise ganz gut dazu geeignet.

Die Frage, wieviele Shapes man definieren und auch darstellen könne, erübrigt sich fast: Beliebig viele, solange der Speicherplatz für die Strings reicht. Und davon haben wir eine ganze Menge! Man könnte sich also eine ganze Bibliothek von Hochauflösungs- oder Multicolorobjekten zulegen und diese dann bei Bedarf abrufen. Sollte für ein Objekt der String zu kurz sein, können - ebenso wie wir es bei den Sprites schon praktiziert haben - mehrere Shapes gekoppelt werden. Sie erkennen schon: Auch hier sind die Möglichkeiten sehr breit.

Zusammenspiel von SSHAPE und GSHAPE.

Eine Frage haben wir noch offen gelassen, die eine recht verlockende Konsequenz in sich birgt: Wenn man mittels der SCALE-Anweisung die Systeme bei SSHAPE (also beim Aufbau des Shapes) und bei GSHAPE (bei der Abbildung) unterschiedlich wählt, kann man dann Shapes verkleinern oder vergrößern?

Die Antwort ist leider nein. Bei GSHAPE werden nur die Koordinaten eines (des linken oberen) Eckpunktes angegeben. Lediglich auf den Darstellungsort hat somit ein unterschiedliches Koordinatensystem eine Wirkung, nicht aber auf die Shape-Größe.

Bewegen von Shapes

Im Gegensatz zu den Sprites ist das Bewegen von Shapes eine langweilige Angelegenheit. Jedem Shape-Aufbau durch GSHAPE kann man ganz geruhsam zusehen. Zwar ist es durch Sequenzen wie:

```
FOR I = 0 TO 200
   GSHAPE A$,I,I,4
   GSHAPE A$,I,I,4
NEXT I
```

möglich, Shapes ohne Schaden für den Bildschirminhalt über den Sichtbereich ziehen zu lassen. Das ganze ähnelt aber bei weitem nicht der Sprite-Bewegung. Aus diesem Grund ist es anzuraten, bei der Bewegung von Bildschirmobjekten – wann immer möglich – Sprites zu wählen.

Shapes der Nachwelt erhalten

Wir stehen bei den Shapes demselben Problem gegenüber wie bei den Sprites. Auf welche Weise kann ein einmal erstelltes Shape für weiteren Gebrauch beiseitegelegt werden?

1. Shapes im Speicher

a. Der Shape-String

Durch SSHAPE wurde ein String geschaffen, der die gleiche Lebensdauer hat wie jede andere

Variable: RUN und CLR löschen ihn.

b. Shape im Sprite-Speicher

Wir werden später eine Möglichkeit kennenlernen, die es unter gewissen Umständen erlaubt, ein Shape zum Sprite zu machen. Auf diese Weise gelangt das Muster dazu dann in den Speicherraum, der den Sprites vorbehalten ist und überlebt, bis der Computer abgeschaltet oder der Speicher überschrieben wird.

2. Shapes auf Diskette oder Kassette

Damit wird es nun wirklich interessant. Der Massenspeicher kann Shapes »auf ewig« festhalten.

a. Shape-Strings direkt

Im Grunde genommen sollte es möglich sein, den durch SSHAPE definierten String direkt als sequentiellen File auf Diskette oder Kassette abzulegen und zu lesen. Allerdings ergeben sich hier – wie auch schon bei den Sprite-Strings – Schwierigkeiten beim Lesen. Weil im Prinzip alle Bytewerte von 0 bis 255 auftreten können, also auch ein Wert, der einem RETURN oder einem Anführungszeichen entspricht, wird der String nicht immer vollständig und fehlerfrei gelesen. Dazu kommt das Problem, daß die Shape-Strings unterschiedliche Längen aufweisen können. Wie bestimmt man das Ende eines solchen Files?

b. Als ASCII-Werte speichern

Ebenfalls bei den Sprites wurde diese Möglichkeit schon angedeutet: den Shape-String auseinander zu nehmen und jedes Byte in Form seines ASCII-Wertes zu speichern. Eine Variante dieses Verfahrens zeigt Ihnen das Listing 11 »SHAPE SPEICHERN«. Hier wird zuerst ein Shape (einfach ein Kreis) erzeugt und in A$ abgelegt. Ein Unterprogramm druckt auf dem Textbildschirm (für Benutzer von 2 Bildschirmen sofort, für andere erst später sichtbar) die ASCII-Werte des Strings aus. In Zeile 40 wird der alte File SHAPE1 gelöscht und dann ab Zeile 50 der neue auf Diskette abgelegt. Benutzen Sie eine Datasette, dann sind lediglich in den Zeilen 60, 80, 120 und 180 die DOPEN und DCLOSE gegen OPEN und CLOSE mit den entsprechenden Gerätenummern auszutauschen. Nun sind wir eigentlich schon fertig: In Zeile 100 löschen wir alle Variablen, den Grafikbildschirm und starten den zweiten Teil des Programmes, der nur zur Kontrolle den File einliest und unser Shape abbildet. Zusätzlich werden auf dem Textbildschirm wieder die Stringlänge und die eingelesenen ASCII-Werte abgebildet. Beide Ausdrucke lassen sich gut vergleichen (Benutzer eines Bildschirmes kommen in diesen Genuß, wenn sie nach Ende des Programmes durch GRAPHIC0 den Textbildschirm einschalten).

c. Eine unorthodoxe Lösung

Etwas exotisch und daher hier nur kurz skizziert ist die folgende Vorgehensweise: Mittels des POINTER-Befehls (hier also POINTER(A$)) sucht man in BANK 1 den Stringdescriptor. Die dort genannte Adresse und Länge verhilft uns dazu, den Speicherbereich, in dem wir unseren String stehen haben, durch BSAVE abzuspeichern.

Beim Wiedereinladen müßte man zunächst einen String definieren, dann durch BLOAD dessen Inhalt an eine beliebige Speicherstelle packen und den Stringdesciptor darauf richten.

Das erscheint im ersten Moment ziemlich kompliziert zu sein, könnte aber – besonders bei Verwendung vieler Shapes – schneller funktionieren. Damit würde beispielsweise ein festgelegter Speicherbereich für alle Shapes en bloc ein- oder auslesbar werden. Probieren Sie's doch mal!

3. Shapes für ein Listing

a. Als Zeichenvorschrift

Häufig werden Shapes durch einige wenige Grafik-Befehle erstellt. Die einfachste Methode – der wir uns auch bisher bedient haben in den Beispielprogrammen – ist es daher, die Zeichenvorschrift ins Programm einzuarbeiten. Manchmal stößt man damit aber an Grenzen: Wenn beispielsweise der Ausschnitt eines Multicolorbildes als Shape verwendet werden soll, das auf dem Grafiktablett erstellt wurde oder wenn es sich um ein Teil eines Fractals handelt, das 25 Stunden bis zur Fertigstellung dauert etc.

b. Shapes in DATA-Zeilen

Ähnlich wie wir bei Sprites einen DATA-Generator verwendet haben, der sich nach seiner Verwendung selbsttätig löscht, geschieht das nun hier auch. Im Programm »SHAPE DATA« (Listing 12) ist außerdem noch ein kleiner Testteil enthalten.

Wie gehabt, wird hier zuerst ein Shape (ein Kreis) erzeugt, dann endet das Programm an einem STOP in Zeile 6. Wenn Sie mit dem 40-Zeichen-Bildschirm arbeiten, dann fügen Sie bitte vor das STOP-Kommando noch ein GRAPHIC 0 ein, denn alles, was nun folgt, ist Text. Das STOP-Kommando wurde eingebaut, um Ihnen die Möglichkeit zu geben, an dem Shape vor dem Aufruf des DATA-Generators noch etwas zu verändern. Ist alles in Ordnung, dann geben Sie bitte CONT ein. Der DATA-Generator erzeugt nun acht Zeilen mit DATAs und dazu noch das Einleseunterprogramm sowie die Löschanweisung DELETE 63000-63012.

Die erste Zahl in den DATAs ist die Stringlänge, die als A später beim Auslesen und Zusammenfügen des Shape-Strings eine wichtige Rolle spielt. Ist das Programm mit dem Ausdrucken der neuen Zeilen fertig, dann bricht es ab. Durch die HOME-Taste gelangen Sie in die erste Zeile. Mittels mehrfach wiederholtem RETURN können nun alle neuen Zeilen ins Programm übernommen und der Generator gelöscht werden. Hinterher sieht unser Programm dann aus wie in »SHAPE DATA 2« (Listing 13). Zur Kontrolle lassen Sie doch mal dieses Programm durch RUN 10 starten. RUN löscht ja auch die Strings, so daß das nunmehr gezeichnete Shape aus den DATA-Zeilen wiederauferstanden ist.

Die Vorgehensweise bei diesem Programm ist dieselbe wie beim DATA-Generator für die Sprites. Allerdings werden hier die Werte nicht aus dem Speicher, sondern aus einem String geholt, der deshalb auch schon durch einen – zumindest teilweisen – Programmlauf definiert worden sein muß.

Wenn Sie mehrere Shapes auf diese Weise festhalten wollen, ist das Programm relativ einfach umzubauen, am besten mittels eines String-Arrays und einer weiteren Schleife.

Wenden wir uns nun dem Zusammenspiel von Sprites und Shapes zu:

Shapes und Sprites

Erzeugen von Sprites aus Shapes.

Anstatt mit dem SPRDEF-Befehl können Sprites auch regelrecht gezeichnet werden durch Grafik-Befehle im Hochauflösungs- oder im Mehrfarbenmodus. Die Nahtstelle zwischen Sprite und Shape ist dabei der String, in den unsere Zeichnung zuerst als ein Shape (mittels SSHAPE) eingeschrieben wird. Von dort ist es dann durch SPRSAV in den Sprite-Speicher zu übertragen.

Eine Regel ist dabei zu beachten:

Der Sprite-String liegt in seiner Länge fest. Im Normalmodus müssen wir immer von einem 24 mal 21 (das ist X mal Y) Raster ausgehen, im Mehrfarbenmodus vom 12 mal 21 Muster. Eine SSHAPE-Anweisung muß daher - falls wir ein Sprite konstruieren wollen - im Normalmodus lauten:
SSHAPE A$,X,Y,X+23,Y+20
und im Multicolormodus:
SSHAPE A$,X,Y,X+11,Y+20

X und Y sollen die Koordinaten der linken oberen Ecke des gewünschten Bildausschnittes sein.

Ist SSHAPE nicht in diesem Raster erstellt worden, kann sich allerlei Unsinn auf dem Bildschirm abspielen.

Ein kleines Demoprogramm (Listing 13) namens »SHAPE UND SPRITE« spielt diese Option durch. Zunächst wird ein Multicolorshape erzeugt und in Sprite 1 geschrieben. Eine Umstellung vom Multicolor- in den Hochauflösungsmodus zeigt, daß das Sprite seine Farben behält, das Shape dagegen nicht. Ein zweites Shape erzeugt Sprite 2, die beide über den Bildschirm bewegt werden. Falls Sie nach Programmende mittels GRAPHIC 0 noch auf den Textbildschirm (40 Zeichen) umschalten, sehen Sie noch einen Unterschied zwischen Shapes und Sprites: Letztere bleiben auch hier aktiv.

Shapes aus Sprites bauen

Natürlich ist auch der umgekehrte Weg möglich: Nämlich Shapes aus Sprites zu konstruieren. Fügen Sie einfach mal an unser letztes Programm aus Listing 13 »SHAPE UND SPRITE« folgende Zeilen an:

```
130 CLR :REM LOESCHEN DER STRINGS
150 SPRSAV 1,A$:SPRSAV 2,B$
160 GSHAPE A$,150,100:GSHAPE B$,200,100
```

Über den Sinngehalt solch einer Zuordnung läßt sich natürlich streiten. Denkbar wäre es, das anzuwenden bei Speicherung eines Shape-Musters im Spritespeicher und anschließendem Zurücklesen dieser Information. Darüber hatten wir oben bei Aufbewahrung von Shapes für die Nachwelt schon kurz nachgedacht.

Wenn Shapes und Sprites zusammenstoßen

Kollisionen von Sprites mit Shapes werden vom Computer ebenso registriert, als wären es Zusammenstöße mit Textbestandteilen. Sowohl der COLLISION- als auch der BUMP Befehl können verwendet werden. In einer erweiterten Version unseres Programmes »SHAPE UND SPRITE« namens »SHAPE/SPR-KOLL« (Listing 14) werden alle Möglichkeiten, die wir in den letzten Abschnitten besprochen haben, ausprobiert. Dabei findet der COLLISION-Befehl Anwendung und Sprite 1 ist durch einen Joystick in Port 2 steuerbar. Bei jedem Zusammenstoß wechselt das Sprite die Farben.

Wann Shapes und wann Sprites?

Jeder von Ihnen kann eigentlich nun die Frage beantworten, wann welches der beiden Grafik-Objekte sinnvoll einzusetzen ist: Shapes und Sprites.

- Bewegte Objekte sollten Sprites sein
- Objekte, die in verschiedenen Grafikmodi (auch im 40-Zeichen-Textmodus) in gleicher Gestalt auftreten sollen, können nur Sprites sein.
- Statische Objekte in beliebiger Anzahl sind als Shapes gut zu verwirklichen
- Objekte, bei denen kein starr festgelegter Raster zugrundegelegt wird, können Shapes sein.

Nun sind Ihrer Phantasie keine Grenzen gesetzt: Bauen Sie sich ein Menü, in dem mit einem Sprite in Pfeil- oder Handform auf Symbole (Shapes) gedeutet wird, oder basteln Sie sich Schaltpläne aus vorgefertigten Einzelteilen (Shapes) zusammen, die Sie mit einem Kran-Sprite aussuchen und plazieren oder, oder.

(H.Ponnath/og)

```
10 rem ***** data-lader fuer alle sprites *****
15 fast
20 gosub100:bsave"sprites1",onb0,p3584top4095
30 gosub100:bsave"sprites2",onb0,p3584top4095
40 gosub100:bsave"scene1",onb0,p3584top4095
50 gosub100:bsave"scene2",onb0,p3584top4095
60 gosub100:bsave"scene3",onb0,p3584top4095
65 slow
70 printchr$(147)"alle sprite-files sind gespeicherte"
80 end
100 rem *** daten in sprite-speicher schreiben ***
110 fori=3584to4095:reada:pokei,a:nexti
120 return
63000 rem ******* sprite-daten *********
63010 rem *** datas von file sprites1 ***
63020 data000,000,000,000,000,000,170,000,001,254,000,002,001,000,004,000,128,004,204,128,004,000,128,060,048,240,036,000,144,036,132
63021 data144,036,120,144,034,001,016,001,254,000,000,132,000,000,132,000,000,132,000,000,132,000,003,135,000,000,000,000,000
63022 data000,000,000,000,000,000,000,170,000,001,254,000,002,001,000,004,000,128,004,204,128,004,000,128,060,048,224,036,000,160,036,132
63023 data160,036,120,160,034,001,032,001,254,032,000,132,032,000,130,000,000,129,000,000,128,128,000,128,096,003,128,000,000,000,000,000
63024 data000,000,000,000,000,000,000,170,000,001,254,000,002,001,000,004,000,128,132,204,128,132,000,128,252,048,224,004,000,160,004,132
63025 data160,004,120,160,002,001,032,001,254,034,000,132,034,000,067,254,000,064,000,000,064,000,000,064,000,001,192,000,000,000,000,000
63026 data000,000,000,000,000,000,000,170,000,001,254,000,002,001,008,004,000,136,004,204,136,132,000,136,252,048,248,132,000,136,004,132
63027 data136,004,120,142,002,001,000,001,254,000,000,128,000,000,064,000,000,032,000,000,032,000,000,224,000,000,000,000,000
63028 data000,000,000,000,000,192,000,170,096,001,254,048,002,001,024,004,000,140,132,204,139,132,000,136,252,048,248,004,000,128,004,132
63029 data128,004,120,128,002,001,000,001,254,000,000,128,000,000,064,000,000,032,000,000,032,000,000,224,000,000,000,000,000
63030 data000,000,000,000,000,000,255,170,000,161,254,000,162,001,000,036,000,192,036,204,160,036,000,144,060,048,248,004,000,128,004,132
63031 data128,004,120,128,002,001,000,001,254,000,000,128,000,000,064,000,000,032,000,000,032,000,000,224,000,000,000,000,000
63032 data000,000,000,000,000,000,000,170,000,001,254,000,002,001,000,004,000,192,004,204,160,004,000,144,060,048,248,036,000,128,036,132
63033 data128,164,120,128,098,001,000,033,254,000,016,128,000,008,064,000,004,003,002,000,002,032,000,004,032,000,008,224,000,000,000,000,000
63034 data000,000,000,000,000,000,000,170,000,001,254,000,002,001,000,004,000,128,004,204,128,004,000,128,028,048,224,020,000,160,020,132
```