General Information
Author: Peter Finzel
Language: ACTION!
Compiler/Interpreter: ACTION!
Published: ATARI Magazin #1 (01/02-87), ACTION! Center 1

Blitzschnelle Vektoren#

Dieses erste Action!-Center befasst sich mit Grafikanimation auf den 8-Bit-Ataris.#

Obwohl die Softwareflut bei den 8-Bit-Ataris längst nicht so hohe Wellen wie bei Schneider-Computer oder dem C64 schlägt, gibt es doch einige "Software-Perlen", um die uns die Besitzer der obigen Computertypen mehr als beneiden. Eine davon ist die Programmiersprache Action!. Dies ist eine strukturierte Compilersprache nach Art von Pascal bzw. C, die zu purer 6502-Maschinensprche übersetzt wird und damit Laufzeiten erreich, die sonst nur Assembler-Programmen vorbehalten sind.

Dabei ist Action! so einfach zu programmieren wie Basic, nur eben ein wenig anders, da es sich um eine strukturierte Sprache handelt. Mit anderen Worten, es gibt kein GOTO, dafür aber eine Reihe von Verzweigungs- und Schleifenstrukturen, wodurch der Sprungbefehl mehr als überflüssig wird. Lassen Sie sich durch das Fehlen von Zeilennummern und dem Einrücken der Zeilen nicht abschrecken. Letzteres ist nur ein Stilmittel zur Hervorhebung der Struktur. Natürlich könnte man Action!-Programme auch in bester Basic-Manier schreiben (möglichst viel in einer Zeile). Doch bringt dies nicht; das kompilierte Programm wird weder kürzer noch schneller.

Wo soviel Licht ist, dar natürlich ein wenig Schatten auch nicht fehlen. Sicherlich der negativste Aspekt an Action! ist der hohe Anschaffungspreis, der immerhin in der Größenordung eines neuen 800XL liegt. Die Programme sollen nur mit eingestecktem Action!-Modul laufen, doch glücklicherweise haben Handbücher nicht immer recht, und es gibt eine Anzahl von Tricks, mit denen man Action!-Programme vom Modul unabhängig macht. Wer meine Assemblerecke in der Zeitschrift CK-Computer Kontakt (6-7/86) verfolgt hat, weiß bereits Bescheid.

Genug der Vorrede, es wird höchste Zeit, dass wir uns mit einem konkreten Beispiel beschäftigen. wie Sie nun wissen, ist Action! enorm schnell und daher genau die richtige Sprache zum Programmieren von flinker Grafik. Nein, wir wollen nicht schon wieder die Player-Missiles strapazieren, sondern diesmal Animation in hochauflösender Grafik produzieren. Genauer ausgedrückt wird es sich um Vektorgrafik handeln. Was versteht man nun darunter?

In normaler (Raster-) Grafik ist ein Objekt (Shape, Sprite, Bob usw.) aus einzelnen Punkten aufgebaut. Bei Vektorgrafik dagegen wird ausschließlich mit Linien Vektoren gezeichnet. Zur Definition eines Objekts gibt man die Anfangs- und Endpunkte aller dazu benötigten Linien an. Der enorme Vorteil liegt darin, dass Vektorobjekte praktisch stufenlos vergrößert und verdreht werden können, indem man die Koordinaten der Anfangs- und Endpunkte mathematisch umrechnet. Im Beispiel werden wir uns ein Programm näher ansehen, das solche Objekte zeichnen und vergrößern kann.

Wer Ab und zu in eine Spielhalle geht, kennst sicherlich "Star Wars" oder das beinahe schon historische "Asteroids"; beide arbeiten mit reiner Vektorgrafik. die Automaten verfügen über spezielle Hardware (Vektor-Displays), die nur auf diese Grafik ausgelegt sind.

So etwas steht uns im Atari zwar nicht zur Verfügung, doch können wir es leicht nachahmen. Man benötigt dazu nur hochauflösende Grafik und ein Programm, das sehr schnell Linien zeichnen kann. Beides ist für den Atari kein Problem, denn schließlich besitzt er den ANTIC-Chip für Grafik und Action! für flotte Programme.

Wir dürfen allerdings nicht den Action!-Befehl DRAWTO zum Linienzeichnen verwenden. Er arbeitet nämlich mit einer Routine des Betriebssystems, die auch von Basic benutzt wird und nicht gerade schnell ist. Wesentlich besser ist es, die DRAWTO-Routine selbst in Action! zur schreiben (im Programm ist das die Routine LineTo). Es ist kaum zu glauben, aber sie ist tatsächlich schneller als ihr in Assembler programmiertes Gegenstück im ROM des Betriebssystems.

Jetzt zu den Vektorobjekten. Ein solches wird im Programm in einem Byte-Array abgelegt. Das erste von je drei aufeinanderfolgenden Bytes bestimm, ob ein neuer Anfangspunkt festgelegt oder eine Linie gezogen werden soll. Das kann man sich bildlich so vorstellen, dass das Objekt auf kariertes Papier gezeichnet würde. Bei einer Null wird der Stift abgehoben und nur der neue Punkt angesteuert; eine Eins hingegen bedeutet, dass eine Linie vom letzten zum neuen Punkt gezogen wird. Ein Wert von $FF zeigt an, dass das Objekt fertig gezeichnet ist. Das zweite und dritte Byte geben die Koordinaten des neuen Punktes (zuerst X, dann Y) an.

Im Programm wurde as Beispiel das Atari-Logo (im Array Atari_L) abgelegt. Zum Entwurf eines Objektes zeichnet man es auf kariertes Papier und überträgt die Koordinaten nach der oben geschilderten Methode in ein Byte-Array. Beim Entwurf sollte man sich an einer Grüße von 10 x 12 Kästchen (Höhe x Breite) halten, da sonst die Mitte des Objektes nicht richtig berechnet wird (oder die DEFINEs Mitte_X und Mitte_Y ändern)

Die Prozedur Draw() zeichnet ein gesamtes Objekt in einen HiRes-Bildschirm. Sie können dabei noch im Parameter VERGR angeben, um wie viel das Objekt vergrößert oder verkleinert werden soll. Ein Wert von 0 bis 9 verkleinert. 10 bildet die Originalgröße ab, während es bei höheren Werden vergrößert wird. Daneben können noch Werte zur Verschiebung in horizontaler und vertikaler Richtung angegeben werden.

Animation kann mit der Routine Draw() erzeugt werden, indem man das Objekt zeichnet, dann den Bildschirm löscht und es an einem anderen Platz oder mit anderer Vergrößerung neu zeichnet. Der hässliche Nachteil dieser Methode ist nur, dass durch das dauernde Löschen und Neuzeichnen ein unruhiges und flimmerndes Bild entsteht, wodurch die Animation kaum mehr als solche erkennbar ist.

Daher wurde im Programmbeispiel zu einer List gegriffen. Man verwendet zwei Bildschirme, von denen einer zur sehen ist, während der andere gelöscht und neu gezeichnet wird. Die Umschaltung der beiden Bildschirme erledigt die Prozedur Switch_Screen(). Durch diese Technik Wird die Animation fließend.

Das Hauptprogramm nützt die Möglichkeiten der Draw()-Routine, um zwei Atari-Logos nebeneinander abwechselnd zu verkleinern und zu vergrößern. Der Effekt ist recht plastisch; je eines der beiden Fuji-Symbole scheint in der Tiefe des Raumes zu verschwinden, um danach wieder neu aufzutauchen.

Das Programm verwendet einige Tricks und Kniffe, die Sie auch in eigenen Programmen gewinnbringend einsetzten können. Da wäre zunächst die Definition einiger Variablen in der Zero-Page. Mit zwei SET-Anweisungen wird Action! instruiert, die Definition von row bis hin zu yf in der Zero-Page ab der Speicherzelle $F0 vorzunehmen. So kann Action! diese Variable (vor allem dem Array-Zeiger row) wesentlich schneller ansprechen. Nach ihrer Definition wird die Adresse des Objectcodes durch zwei weitere SET-Befehle auf die Adresse $7000 verlegt.

Mit den SET-Anweisungen kann man die Code-Erzeugung wie bei einem Assembler-Programm mit ORG (bzw. mit "*"=) steuern. Normalerweise würde das von Action! erzeugte Objektfile direkt nach dem Textfile abgelegt. Diese Methode würde sich beim vorliegenden Programm jedoch nicht anbieten, da eine Display-List definiert wird, die durch gewisse Einschränkungen des ANTIC-Bausteins keine 1 KByte-Grenze überschreiten darf. Legt man jedoch die Anfangsadresse fest auf $7000 und definieren die Display-List am Anfang des Programms wird dieses Problem vermieden.

Die Display-List wird einfach durch mehrere BYTE- und CARD-Definitionen erzeugt. Verwendet wird ein Modus, der GRAPHICS 6 entspricht (Auflösung 160 mal 96, zwei Farben). Natürlich könnte man in diesem Falle auch einen GRAPHICS-6-Aufruf benutzen, aber da wir später mit Page-Flipping arbeiten wollen, ist die erste Methode eleganter. Es genügt dann, der Variablen LMS einen anderen Wert zuweisen, um die Adresse des Videospeichers zu verändern.

Anschließend folgen einige Byte-Arrays, die das Objekt sowie eine Adresstabelle enthalten. Letztere ist für die Fast_Plot()-Routine nötig, damit die Adresse der Zeilenanfänge möglichst schnell herausgefunden werden können. Auch hier ein Trick. Anstatt ein großes CARD-Feld zu verwenden, werden LSBs und MSBs der Adressen in getrennten Byte-Feldern aufbewahrt. Auf diese Weise kann der Zugriff viel schneller erfolgen.

Nun folgt die Fast_Plot()-Routine, in der auch einige wirkungsvolle Tricks versteckt sind. Schreibt man nämlich zwischen Namen und Parameterklammer ein "=*", so verzichtet Action! darauf, die Parameter in lokale Variablen abzulegen. Man kann das dann selbst mit einem kleinen Codeblock erledigen. Die beiden Parameter werden aus dem X- und Y-Register in einen reservierten Zero-Page-Speicherbereich von Action! gebracht. Da es nun nicht mehr weiß, wo die beiden zu finden sind, legt man zwei adressierte Variablen (im vorliegenden Fall X und Y) darauf. Der indirekte Zugriff auf den Videospeicher geschieht über den Array-Zeiger row, der mittels Adresstabellen aus LSB und MSB zusammengesetzt wird.

Die LineTo()-Routine stammt aus dem Programm "View 3D" von Paul Chabot (Antic 6/85). Sie ist sehr schnell, da nur Additionen und Subtraktionen und die Fast_Plot()-Routine benutzt werden. die Prozedur Graphic_Init() aktiviert die Display-List und bereitet Adresstabelle und das Video-RAM vor, während Screen_Switch() zwischen den beiden Bildschirmen (die übrigens bei Adresse $8000 bzw. $8800 beginnen) hin- und herschalten kann.

Damit wären wir am Ende des ersten Action!-Centers angelangt. Ich hoffe, es hat Ihnen gefallen und Sie haben einige neue Anregungen bekommen. Im nächsten Heft werden wir besprechen, wie man Interrupts in Action! programmieren kann. Ich würde mich freuen, wenn Sie wieder mit von der Partie sind.

```
;**********************************
;      VEKTORGRAPHIK IN ACTION!
;
;P. FINZEL                        1986
;**********************************

DEFINE VRAM1  ="$8000", ;Screen 1
       VRAM2  ="$8800", ;Screen 1
       VRLen  ="1920",  ;Laenge Screen
       VMax   ="40",    ;max. Vergroesserung
       Mitte_X="6",     ;Mitte des
       Mitte_Y="5",     ;Objektes
       MODE   ="$B$B$B$B$B"
;
;Zero-Page Variable
;
SET $E=$F0 SET $F=0
BYTE ARRAY row
BYTE rowl=row, rowh=row+1
BYTE t,a,b,Xnow,Ynow
BYTE dx,dy,xf, yf
;
;Programm ab $7000 ablegen
;=========================
;
```

```
SET $E  =$7000
SET $491=$7000
;
;
;Variablen und Daten
;===================
;
CARD dlist =560 ;Display-List Zeiger
BYTE color0=708 ;Schattenreg. Farbe 1
;
CARD Wrk  =[ $8000 ] ;Zeiger auf bearbeiteten
BYTE Wrkh = Wrk+1    ;Screen (Wrkh int MSB)
BYTE scr  =[0]       ;momentaner Screen

;-----------------------------------
; Die Display-List:
;-----------------------------------
BYTE DLST0=[ $70 $70 $70 $4B ]
CARD LMS  =[ $8000 ]
BYTE DLST1=[ MODE MODE MODE MODE
             MODE MODE MODE MODE
             MODE MODE MODE MODE
             MODE MODE MODE MODE
             MODE MODE MODE $41 ]
CARD DJMP =[ 0 ]

;-----------------------------------
; Adresstabelle
;-----------------------------------
BYTE ARRAY
  adrl(96),adrh(96),
mask8(0)=[128 64 32 16 8 4 2 1]

;-----------------------------------
;Objekt in 10x12 Raster:
;-----------------------------------
BYTE ARRAY ATARI_L=[
0  3  0:1  4  0:1 4 5:1 1 10:1 0 10
1  3  5:1  3  0:0 5 0:1 7  0:1 7 10
1  5 10:1  5  0:0 8 0:1 9  0:1 9  5
1 12 10:1 11 10:1 8 5:1 8  0:$FF$0$0]
;
;-----------------------------------
; Gaphikpunkt setzen
;-----------------------------------
PROC Fast_Plot=*(BYTE x1,y1)
  BYTE X=$A0, Y=$A1
  BYTE xb=$A2, Xr=$A3

  [ $85 $A0 $86 $A1 ]

  IF Y<96 THEN
      rowl=adrl(y)
      rowh=adrh(y)+wrkh
      xb=x RSH 3:xr=x AND 7
      row(xb)== % mask8(xr)
  FI
RETURN
```

```
;-----------------------------------
;Graphik-Linie ziehen
;-----------------------------------
PROC LineTo(BYTE x,y)
BYTE i
Fast_Plot(xnow,ynow)
IF x=xnow AND y=ynow THEN RETURN FI
IF x>xnow THEN
       dx=x-xnow:xf=1
ELSE
       dx=xnow-x:xf=$FF
FI
IF y>ynow THEN
       dy=y-ynow:yf=1
ELSE
       dy=ynow-y:yf=$FF

FI
x=xnow:y=ynow
IF dx>dy THEN
  a=dy+dy:t=a-dx:b=t-dx
  FOR i=1 TO dx
    DO
    x==+xf
    IF t>127 THEN
       t==+a
    ELSE
       t==+b:y==+yf
    FI
    Fast_Plot(x,y)
  OD
ELSE
  a=dx+dx:t=a-dy:b=t-dy
  FOR i=1 TO dy
    DO
    y==+yf
    IF t>127 THEN
       t==+a
    ELSE
       t==+b:x==+xf
    FI
    Fast_Plot(x,y)
  OD
FI
xnow=x:ynow=y
RETURN

;-----------------------------------
;Page-Flipping: Screen wechseln
;-----------------------------------
PROC Switch_Screen=*()
  IF scr=0 THEN
       Lms=Vram2
       Wrk=Vram1
  ELSE
       Lms=Vram1
       Wrk=Vram2
  FI
  scr==+1&1
```

```
  Zero(Wrk,VRLen)
RETURN

;-----------------------------------
;Display-List & Adresstabelle anlegen
;-----------------------------------
PROC Graphic_Init()
  BYTE i

  row=0
  FOR i=0 TO 95
    DO
    adrl(i)=rowl
    adrh(i)=rowh
    row==+20
  OD
  Zero(Vram1,Vrlen)
  Zero(Vram2,Vrlen)
  scr=0
  Lms=Vram1
  Wrk=Vram2
  DJMP=@Dlst0
  Dlist=@Dlst0
RETURN

;-----------------------------------
;Graphik-Koerper zeichnen
;-----------------------------------
PROC Draw(BYTE ARRAY Def,BYTE vergr,
                         INT xrel,yrel)
BYTE i
INT X,Y

IF Vergr=0 THEN RETURN FI
i=0
WHILE Def(i)<>$FF
       DO
       X=Def(i+1) X==+xrel-Mitte_X
       Y=Def(i+2) Y==+yrel-Mitte_Y
       X==*Vergr/10+79
       Y==*Vergr/10+48
       IF Def(i)=0 THEN
               Xnow=X Ynow=Y
       ELSE
               LineTo(X,Y)
       FI
       i==+3
       OD
RETURN

;-----------------------------------
;Das Hauptprogramm
;-----------------------------------
PROC Vektorgraphik()
  BYTE i
  Graphic_Init()
  DO
    FOR i=0 TO VMax STEP 3
      DO
```

```
      DRAW(ATARI_L,i,7,0)
      DRAW(ATARI_L,Vmax-i,0-7,0)
      Switch_Screen()
    OD
    FOR i=0 TO VMax STEP 3
      DO
      DRAW(ATARI_L,Vmax-i,7,0)
      DRAW(ATARI_L,i,0-7,0)
      Switch_Screen()
    OD
  OD
RETURN
```

PDF: Schnelle Vektoren in ACTION/vektoreninaction.PDF