General Information#

Author: Donald E. Glover
Language: ACTION!
Compiler/Interpreter: ACTION
Published: ANALOG Computing #20

Stars 3-D#

I was looking for something to do with my shiny new Action! cartridge when I ran across the article **Stars 3-D** by Craig Patchett in **ANALOG** No. 16. To become familiar with the new language, I decided to translate this demonstration program into Action!, a job I thought would take one or two hours. The task eventually took much longer, due to a number of strange quirks associated with the Action! language. I hope this discussion of my problems will save other Action! programmers some hair pulling and nail chewing.

My first task was to find a place for the display list (DLIST) and screen memory (STRLIN). I wanted to put them in a safe location, while allowing easy access from Action!-generated code and in-line machine code. I finally decided to put them in Action! arrays whose starting addresses were defined such that the display list and screen memory started on 1K boundaries in high memory. (The Atari cannot easily deal with a display list which crosses a 1K boundary or screen memory which crosses a 4K boundary.)

Calculations to generate the display list required that the address of screen memory be divided by 256 to obtain the high byte of the address. Performing this division on addresses greater than 32767, unfortunately, gives the wrong answer, since Action! multiplications and divisions always assume they are acting on signed numbers. Try typing:

```
X PRINTCE(32768/256)
```

in the Action! monitor and see what you get. After figuring out the problem, I replaced the division by 256 with "RSH 8" (shift cardinal number right 8 bits). The use of this technique can be seen by examining the procedures STRINI() and DLSINI().

The next problem was to insert the addresses of the arrays STRTPH, STRTPL, and STRPOS into the machine language procedure SCROLL(). My initial attempt to do this involved inserting the address during the compilation phase. Using this method, the first instruction in the procedure SCROLL() would be:

```
$B0 STRTPL; LDA STRTPL,X
```

To my horror, the addresses of arrays compiled into the code by this technique frequently (but not always) differed from those observed after compilation. Apparently, the addresses of arrays change during the compile phase, and the compiler cannot modify addresses inserted into machine code. The solution was to "POKEC" the addresses into the machine language routines during run time [see the procedure MAIN90].

I believe everything else in the listing is understandable, because I kept the names of all routines and most of the comments the same as those in the original assembly language listing. A word of warning: this program is designed to work with a machine having 48K of memory. If your machine has less memory, you will have to change the starting address of the arrays DLIST and STRLIN. The place to do this is clearly marked in the listing.

Before finishing, I should mention another couple of Action! peculiarities.

1. Negative FOR loops do not work. Try:

```
FOR COUNTER=5 TO 0 STEP -1
```

in a sample test procedure. It won't work. 2. You cannot initialize a variable to a negative number.

```
TEMP1=[-1]
```

will not work. However

```
temp1=[[65535]
```

will accomplish the same thing. 3. The example on page 123 of the manual doesn't work (at least with Version 3.5 of Action!). PRINTCE(rec. idnum) prints the wrong answer. For some reason, PRINTCE(rec. idnum*1) gives the right answer. I suspect Action! will never be used to write commercial arcade-style games, because it is just not as efficient at "bit-twiddling" as machine code (try writing the procedure scroll() in Action!). It also does not produce code as compact as that produced by a good assembly language hacker, a definite consideration when trying to stuff a game in an 8K or 16K ROM cartridge. Nevertheless, I feel that the language (perhaps with the help of a few machine language routines) can be used to write games similar in quality to the machine language games found in **ANALOG** in a much shorter time than usually required. Games written in Action! would also be easier for novices to analyze and understand.

Action! Listing

```
; ********************
; *   STARS 3D DEMO   *
; * BY CRAIG PATCHETT *
; * ACTION! VERSION   *
; *  BY DON GLOVER    *
; ********************
; system equates
CARD sdmctl=$022F,
     nmien=$D40E,
     random=$D20A,
     wsync=$D40A,
     sdlstl=$0230,
     vdslst=$0200,
     colpf0=$d016,
     color4=$02C8,
     xitvbl=$E462
BYTE index ;used to index into strcol
CARD indrct=$D6 ;pointer for indirect addressing
BYTE ARRAY timer=[8 8 6 6 5 5 3 3 2 2 1 1], ; timers for scrolling
timary=[8 8 6 6 5 5 3 3 2 2 1 1], ;values to reset timers
mancol=[$22 $24 $26 $28 $2A $2C $2E $22], ; star colors
; ***careful-requires a 40k machine to work
dlist(600)=35840, ;display list
strlin(768)=36864, ;star(screen) memory
; ************************************
strtpl(16), ;addresses of beginning of
strtph(16), ;each star line
strpos(16), ;position of star on line
strcol(192)    ;color of each line
CARD ARRAY strtp(16) ;temp. hold addresses to be
                    ;transferred into strtpl,strtph
PROC setvbv=$E45C(BYTE command,vblankhigh,vblanklow)
; *********************************
PROC SCROLL(BYTE dummy,starindex) ; star scroll routine
```

```
; ACTION! is too much trouble to use here with all the bit twiddling
[
$BD $FFFF    ; LDA STRTPL,X
$85 INDRCT   ; STA INDRCT
$BD $FFFF    ; LDA STRTPH,X
$85 INDRCT+1 ; STA INDRCT+1
$BC $FFFF    ; LDY STRPOS,X
$B1 INDRCT   ; LDA (INDRCT),Y
$0A          ; ASL
$0A          ; ASL
$91 INDRCT   ; STA (INDRCT),Y
$90 $0F      ; BCC SCRBR1
$A9 $01      ; LDA #1
$88          ; DEY
$C0 $FF      ; CPY =255
$D0 $02      ; BNE SCRBR2
$A0 $2F      ; LDY #47
; SCRBR2
$91 INDRCT   ; STA (INDRCT),Y
$98          ; TYA
$9D $FFFF    ; STA STRPOS,X
; SCRBR1
]
RETURN
; **********************************
PROC cntdwn() ; timer routine
   BYTE starcounter
   scroll(0,13) scroll(0,13) ; move fastest star
   scroll(0,12) scroll(0,12) ; and its twin
   ; now do rest of stars
   FOR starcounter=0 to 11
   DO
      timer(starcounter)==-1
      IF timer(starcounter)=0 THEN
         scroll(0,starcounter) ; scroll if ready
         timer(starcounter)=timary(starcounter) ; reset timer
      FI
   OD
RETURN
; **********************************
PROC vblank() ; vblank routine
     cntdwn()
     index=0 ; back to scan line zero
     [
     $4C xitvbl ; jmp xitvbv
     ]
RETURN ; never gets here
; **********************************
PROC strini() ; initialize stars
   BYTE counter
   ; set up screen memory address tables
   ; first use dummy array for simplicity
   FOR counter=0 to 15 DO
      strtp(counter)=strlin+48*counter
      strpos(counter)=0 ;all stars start at pos. zero
   OD
   ; now transfer addresses to byte arrays
   FOR counter=0 to 16 DO
      strtpl(counter)=strtp(counter)&255
```

```
      ; cannot divide numbers>32767 and get correct results
      strtph(counter)=strtp(counter) RSH 8
   OD
   ; clear star memory
     ZERO(strlin,768)
   ; give each line a star
   FOR counter=0 to 15 DO
      strlin(48*counter)=64
   OD
   RETURN
; *********************************
PROC dlsini() ; initialize display list
   BYTE startype,offset,counter
   CARD TEMP
   ; do for each scan line
   dlist(0)=$70 dlist(1)=$70 dlist(2)=$F0
   FOR counter=0 to 191
   DO
      dlist(3*(counter+1))=$CE ; graphics 7+
      ; now pick star type
      startype=RAND(8)
      ; set color for each line
      strcol(counter)=mancol(startype)
      ; randomize offset into memory
      offset=RAND(48) ;offset into line
      ; now put address of line into display list
      dlist(3*(counter+1)+1)=(strtp(2*startype)+offset)&255
      ; cannot divide numbers>32767 and get correct results
      dlist(3*(counter+1)+2)=(strtp(2*startype)+offset) RSH 8
   OD
   ; now finish up display list
   dlist(579)=$41 ; jump instruction
   dlist(580)=dlist&255
   dlist(581)=dlist RSH 8
   sdlstl=dlist ;tell antic where display list is
   sdmctl=$23 ;wide screen
RETURN
; *********************************
PROC dli() ; display list interrupt routine
; too little time to even think about using ACTION!
   [
   $AE INDEX  ; LDX INDEX
   $BD $FFFF  ; LDA STRCOL,X
   $8D WSYNC  ; STA WSYNC
   $8D COLPF0 ; STA COLPF0
   $EE INDEX  ; INC INDEX
   $40        ; RTI
   ]
RETURN
; *********************************
PROC MAIN()
   ; fix up ml routines
   ; by poking in addresses of various arrays
   POKEC(dli+4,strcol)
   POKEC(scroll+7,strtpl)
   POKEC(scroll+12,strtph)
   POKEC(scroll+17,strpos)
   POKEC(scroll+40,strpos)
   POKEC(DLI+1,@index)
```

```
   strini() ;set up stars
   dlsini() ;set up display list
   color4=0 ;set background color
   SETVBV(7,vblank/256,vblank&255) ; set up vblank
   vdslst=dli  nmien=192 ; get dli's going
   ; now just let things run
   DO

   OD
RETURN
```

PDF: Stars in 3D/stars3dinaction.PDF