Back to #amycoders Homepage

Starfield-compo

The mission

Another size optimizing compo! This time the aim is to reduce a 3D-Starfield routine by size. 3D starfields should be wellknown from amiga-demos in the early 90ies.
Since the aim should not be reached by cutting down system compatibility to gain some extra bytes, a source is provided which does all system-init, doublebuffering and screenclearing for you. You just have to add your own optimized code at the given place. More precisely this are two places - one init-routine, which is called once and can be used by you to calculate tables etc. and the main-routine. The main routine is getting a pointer to the bitplane it has to drawn on in A0.

Deadline was: **saturday, june 28, 1997** at 21.00 CET.

Rules:

- You may use as much BSS-space as you want - its not added to the overall length.
- 512 Stars have to be calculate per frame. (And most of them should be visible)
- Registers must be assumed to be in an undefined state , when the Init is called.
- A star is in this case defined as a pixel on one bitplane.
- Perspective projection and clipping at the screen borders..
- The stars are moving towards the spectator (Eyepoint, Z-direction)
- When a stars passed the eyepoint it has to be set to a random position in 3D-space. (No repeating patterns)
- The position (0,0,0) in 3D-space is at (160;128) on the screen!
- No system calls, no direct hardware access. (No $DFFxxx)
- No usage of the content of any memory area not included in the program to generate random values. (This includes reading from the Kickstart ROM)

btw thanks a lot to **Morbid** for providing then Init-Source ! :)

The results

This time 12 people took part in the compo - and unlike at the SQRT-compo there are nearly no contributions with equal length.
Sounds like easy judgeing, but this time I had problems of another kind. 3 of the 10 contributions didnt take care of all rules. Two of them didnt do proper perspective - I didnt disqualify them, since they werent very close of winning. The problem was **Axis** contribution. He sent me one with 62 bytes using some nice tricks, but it had a repeating pattern. The stars repeated every 256 steps. He simply

cycled the Z-values instead of generating new, individual random-positions. I told it to him and he later replied , that he fixed it needing another 6 bytes. But I didnt get his new routine till today. So I will assume that his routine is 68 Bytes.

You thought that was it ? Well, me too ! But on the day after the first (earlier) deadline I got another contribution by **Raylight** , which matched all the rules and had a length of only **60** bytes. Since it was in time for the later, original deadline I decided to let in contribute anyways. So we have a new winner!
Yet another thing happened: I suddenly saw, how to cut my 58 bytes routine down to **54** bytes! Can anyone find another tweak ?

Detailed results:

1.	Raylight	60 bytes
2.	Azure	64 bytes
	Chip	64 bytes
3.	Axis	68 bytes
4.	Dave	74 bytes
5.	Grey	86 bytes
6.	Accede	94 bytes
7.	Dark Angel	96 bytes
8.	Shin *1	102 bytes
9.	Kaneda *2	104 bytes
10.	Wind	108 bytes
11.	Flynn	128 bytes

*1 not doing real perspective transformation.
*2 not doing real perspective transformation, stars are only in 256x256 range.

So - and you thought 64 bytes (my original contribution, which was finished before I got any others contribution) is the minimum ? It isnt ! After getting Axis contribution I saw he was using a quite nice trick to shorten the amount of bytes used for perspective transformation. I substituted my perspective-code with his and managed to shorten Axis perspective transformation code by even another 4 bytes. Ending up with a **60 bytes** routine ! Using another Trick I even made it to **58 bytes**, without violating any of the rules - I cycled the random-numbers. Check out the code for more info :) Graham later told me, that this was, what Axis originally wanted to do in his routine.
The routine is now down to even **54** bytes..

How does it work ?

Explaining every single routine is a bit too time consuming. I will just publish some routines at this page. Download a package with all contributions here. (Zip-File)

The winning 60 byte-routine by **Raylight**: (no doubt, this guy is using GoldED for his sources. :))

```
*** Starfield by Raylight , 54 bytes. See the full source for comments
```

```
          moveq   #-128,d5
          lea     -512*3*2-32(a7),a1
  .lop    movem.w (a1)+,d1-d3
          subq.w  #7,-(a1)
          bgt.b   .ok
          sub.l   (a1),d0
          add.w   d0,(a1)
          eor.l   d0,-(a1)
          addq.l  #4,a1
  .ok     divs.w  (a1),d1
          divs.w  (a1)+,d2
          cmp.w   #160,d1
          bge.b   .skip
          add.w   #160,d1
          blt.b   .skip
          cmp.w   d5,d2
          ble.b   .skip
          add.w   d5,d2
          bgt.b   .skip
          muls.w  #-320,d2
          add.w   d1,d2
          bfset   (a0){d2:1}
  .skip   sub.w   d5,d6
          bvc.b   .lop
```

The winning 64 byte-routine by **Azure**:

```
**** Starfield by Azure, 64 Bytes. Check out the full source for details.

Starfield:
        lsr.w   #7,d6

        lea     StarBuffer+6,a1

.newpos
        ror.l   #7,d7           ;random number generator
        add.w   d6,d7

;       move    d7,-(a1)        ;y-pos now random number
;       move    d2,-(a1)        ;x-pos now former y-pos
;       move    d1,-(a1)        ;z-pos now former x-pos
                                ;using this trick the random-numbers
                                ;are cycled through the coordinate
                                ;parts
        movem.w d1/d2/d7,-(a1)
.lop1
        addq.w  #SpeedOfStars,(a1)
        movem.w (a1)+,d0-d2     ;z x y (movem doesnt alter flags)
        bpl.s   .newpos         ;illegal position (behind viewer or at z=0)

        divs.w  d0,d1
        divs.w  d0,d2           ;perspective
                                ;higher words have been cleared in the
                                ;previous run
        add.w   #160,d1
        add.w   #128,d2
        ext.l   d1
        mulu.w  #40,d2          ;clears upper word of d1/d2 for next
                                ;run

        cmp.w   #320,d1
```

```
        bcc.s   .newpos
        cmp.w   #256*40,d2
        bcc.s   .newpos

        bfset   (a0,d2){d1:1}   ;d2 is row offset
                                ;d1 is x-pos
.out
        dbf     d6,.lop1
        rts
```

This is **Axis** 62 byte routine, which was disqualified. Check it out anyways, since it is quite interesting.

```
**** Starfield by Axis, 62 Bytes. Check out the full source for details.

Starfield:
c               moveq   #$01,d5
                subq    #$01,d2
                lsr.w   #$07,d0
.l0:            move.w  d0,d6
                add.b   d2,d6

                addq    #$01,d6
                move.w  d5,d1
                divu.w  #$4433,d5
                move.w  d5,d7

                moveq   #$21,d3
                moveq   #$7f,d4

.l3:            extb.l  d7

                lsl.l   #$06,d7 ;this one can be left out, but then it
                                ;starts to get really ugly (au=1). :o)

                divs.w  d6,d7

                cmp.w   d4,d7
                bge.b   .l2
                add.w   d4,d7
                bmi.b   .l2

                add.w   d3,d4
                exg     d7,d1
                lsr.w   #$04,d3
                bne.b   .l3

                mulu.w  #$28,d7
                ext.l   d1
                bfset   $28(a0,d7.w){d1:01}
.l2:            dbra    d0,.l0
                rts
```

This is **Azures** (some ideas by **Axis**) 54 byte routine, which didnt take part in the compo:

```
**** Starfield by Azure , 54 Bytes. Check out the full source for details.

Starfield:
        rol.l   d5,d5
        addq    #5,d5
        move.l  d5,d7
.lop1
        addq    #2,d6
        move.w  d7,d1
        rol.l   d7,d7
        addq    #5,d7
        move.w  d7,d2

        moveq   #127,d3
        moveq   #33,d4
.lop2
        ext.l   d2
        divs.w  d6,d2           ;optimized version of axis persp-trick
        cmp.w   d3,d2           ;(4 bytes shorter)

        bge.s   .out
        add.w   d3,d2
        bmi.s   .out
        exg.l   d1,d2
        add.b   d4,d3           ;this will loop once
        bvs.s   .lop2

        mulu.w  #320,d2
        add     d1,d2
        bfset   (a0){d2:1}      ;d2 is row offset
                                        ;d1 is x-pos
.out
        and     #$3fe,d6
        bne.s   .lop1
        rts
```

Last change: 16.01.2001
