Back to #amycoders Homepage

Circledrawing-Compo

The mission

Here comes the logical predecessor of the Linedrawing-Compo: The **Circledrawing-compo**. The mission was to create a routine drawing a circle with variable radius to a chunky-screen. There were both a **fastest** and **shortest** compo again. The choice of the algorithm was free.
The deadline was: **sunday, october 12st, 1997 at 21.00 CET**

Rules:

- The size of the buffer is 256x256, with a color depth of 256.
- Your circleroutine has to accept following input:
  - D0.l/D1.l: X / Y (middle)
  - D2.l radius (1-127)
  - D3.b color
  - A0: Pointer to chunky buffer.
  - The other registers are in undefined state. (Your routine will be disqualified, if it is using input from any other register, than the mentioned ones)
- All registers may be trashed, except a7.
- The circle has to be closed. (no gaps)
- The circle is not meant to be filled !
- The circle must be symmetrical (not egg-like shapes..)
- no clipping required
- The routine must be working on 68020-68060! When you are sick enough to use selfmodifying code, make sure that it does also work on 040/060.
- The speed-tests will be done on a 68040/40. The time needed to draw an amount of circle at different positions and with different sizes will be measured. (small hint: I added some more info in the compo-machine in the linecompo package.

The Results (shortest):

Nine people competed in this compo. The routines were quite interesting and different. A lot of different ideas were involved. One idea was the "bruteforce" approach - scanning the whole screen for the circle with a pythagoras distance formula. These routines got incredible slow. The slowest routine is more than 11,000 times slower, than the fastest one in this compo :)

But **Trover** whose routine won, did show that it is possible to do both a (relatively) fast and shor routine. He also contributed with a 36 bytes version, which needs only **40** rasterlines to draw the testcircles.

The routines in the "shortest" compo had to draw three circles. Two with a radius of 32 and one with a radius of 64. The testroutine is included in the contribution package, which can be downloaded at the bottom of this page.

Detailed Results:

Place	Handle	Length	Speed	Algorithm
1.	Trevor	34	1458	Sin/Cos
2.	Piru	38	4950	Bruteforce Pythagoras/divs
3.	Nitch	40	2920	Bruteforce Pythagoras/2r
	Raylight	40	5534	Bruteforce Pythagoras/divs
5.	Nao *1	40	2298	Sin/Cos

6.	R.A.Y & Sniper	44	23320 !	Bruteforce Pythagoras/sqrt
7.	Dave	46	5	Bresenham
8.	Chip	72-80	3.5	Bresenham
9.	Blueberry	100	2 !	Bresenham

*1 Routine was moved down a place due to its inaccuracy.

The Results (fastest):

Also 9 routines were contributed here. The circleroutines had to draw 504 Circles with 63 different radii, so always 8 circle with the same radius. This lead to a problem with some routines using a kind of "Bruteforce" approach again, by either using a very huge table or generating 128 subroutines, one for each radius. Since these routines speed depended a lot on the memory usage their speed was dependant of the order, in which the circles were drawn. I did a best case and a worst case timing for these routines. (written in parenthesis)
Using the best-case timing only would have been pretty unfair towards the other routines, since no everyday-application would sort the circles by size before drawing. So I used the average value of best and worst time to compare these special routines with the others.

This competition was won by **Piru**. His routine is generating subroutines for every radius, cleverly gathering pixels to .l and .w writes. Due to this it is very cache dependant. So the very clean and short routine by **Blueberry**, who became second, is faster in a lot of cases.

Detailed Results:

Place	Handle	Length	Speed	Algorithm
1.	Piru	436	av. 492 (547/437)	One routine per radius
2.	Blueberry	100	510	Bresenham *1
3.	EFT	112	518	Bresenham *1
4.	Dave	188	551	Bresenham *1
5.	Hitchhiker	146	557	Bresenham *1
6.	Raylight	162	av. 564 (636/492)	One routine per radius
7.	Mhoram	150	589	Bresenham *1
8.	Piru, second entry	292	av. 662 (723/601)	Big table
9.	Chip	80	945	Bresenham *1

*1 Please note, that I didnt take a too close look to the routines.
So routines marked as using Bresenham might also use a similar algorithm instead.

How does it work ?

I am only describing the best two routines of the "shortest-circleroutine" compo here. Download the whole package of routines below.

The winning routine by **Trevor**. This is the 36 bytes version, which is a lot faster. The 34 bytes version is using a "moveq #-128,d7" at the beginning

This routine is using the parametric equation of a circle: ***X=r*cos(t) ,Y=r*sin(t)***. The sinus and cosinus function is calculated by a recursive formula v 2*pi*256 steps, which is enough to draw a closed circle. Read the Sinustable Generation Tutorial for further info. ("Another Algebraic Approach")

```
;Winning routine by Trevor, 36 bytes version.
        move.w  #256*7,d7       ;steps*2*pi
        lsl     #8,d2           ;d2=x
        moveq   #0,d4           ;d4=y
.hamburger
        move.w  d2,d5
        asr.w   #8,d5
        subx.w  d5,d4           ;y=y-(x/256)
```

```
move.l  d4,d6
asr.w   #8,d6
addx.w  d6,d2              ;x=x+(y/256)

add.b   d1,d6              ;000000000000y+y1
lsl.w   #8,d6              ;00000000y+y10000
move.b  d5,d6              ;00000000y+y1000x
add.b   d0,d6              ;00000000y+y1x+x0
move.b  d3,(a0,d6.l)
dbf     d7,.hamburger      ; :P
```

Pirus 38 byte routine, which became second:

It is using a kind of "bruteforce" approach. It is stepping through the whole 256x256 array of the chunky buffer and calculating the distance from the pythagoras formula. If the distance is between r+1 and r, the pixel is considered to be on the circle. (A rasterized circle has an area in fact.. :))

Since a squareroot-routine , which would be needed for the formula ***d=sqrt(x^2+y^2)***, is extremly slow and not that short Piru used another trick. He ***1;r] r e Z*** then. This is mathematically not entirely correct (figure that out byself.. :)) but it seems to be accurate enough, since the circles drawn wi

```
;Second placer by piru, 38 bytes.
        moveq   #0,d7           init loop counter, also y:x 24:8

.loop   moveq   #0,d4           clear x
        move.l  d7,d5
        move.b  d7,d4           x=offs & $00ff
        lsr.w   #8,d5           y=offs>>8
        sub.w   d0,d4           x=x-xc
        sub.w   d1,d5           y=y-yc

        muls.w  d4,d4           x^2
        muls.w  d5,d5           y^2
        add.l   d5,d4           x^2+y^2
        divu.w  d2,d4           (x^2+y^2)/r
        sub.w   d2,d4           (x^2+y^2)/r-r
        bgt.b   .nopix          (x^2+y^2)/r-r<0 ?
        addq.w  #1,d4           (x^2+y^2)/r-r+1
        blt.b   .nopix          (x^2+y^2)/r-r+1<0 ? move.b d3,(a0) put pixel .nopix addq.w #1,d7 loop 64k times addq.l #1
```

Download

In case you want to see the contributions code. Download the package here ***Please remember that, even if you can download these routines, they are still not public domain. Ask before using any of these routines, or give credits at least.***