Drawing Circles with the NS32GX32

National Semiconductor
Application Note 644
Dave Rand
Adapted by Ariel Faigon
for the NS32GX32
September 1989

1.0 INTRODUCTION

In this application note, a method for high-speed circle generation is described, using an optimized version of Bresenham's circle algorithm.

2.0 DESCRIPTION

A circle can be described by the center coordinates (xc, yc), the radius (r), and the width (w). With the Pythagorean theorem, pixels along the path described by the equation:

$$(x - xc)^2 + (y - yc)^2 = r^2$$

can be set for a width of w perpendicular to the tangent of the arc.

This, however, involves substantial computation for each point on the line. Even taking advantage of the symmetry of circles, a large number of instructions must be executed to calculate the path.

Bresenham's circle algorithm works by determining which of two pixels are nearer the actual circle at each step. Then, using symmetry, eight points on the circle's path can be determined. Applying the width (w) to each of these eight points yields a displayed (or imaged) circle. For the actual derivation of Bresenham's algorithm, see *Reference 1*, and *Reference 2*. This derivation was done by J. Michener.

Bresenham's algorithm can be implemented in the following manner:

1. Select the first position for display as
   $$(x_1, y_1) = (0, r)$$
2. Calculate the first parameter as
   $$p_1 = 3 - 2r$$
   If $p_1 < 0$, the next position is $(x_1 + 1, y_1)$. Otherwise, the next position is $(x_1 + 1, y_1 - 1)$.
3. Continue to increment the x coordinate by unit steps, and calculate each succeeding parameter p from the preceding one. If for the previous parameter we found that $p_i < 0$ then
   $$p_{i+1} = p_i + 4x_i + 6$$
   Otherwise (for $p_i \geq 0$),
   $$p_{i+1} = p_i + 4(x_i - y_i) + 10$$
   Then, if $p_{i+1} < 0$ the next point selected is $(x_i + 2, y_{i+1})$. Otherwise, the next point is $(x_i + 2, y_{i+1} - 1)$. The y coordinate is $y_{i+1} = y_i$, if $p_i < 0$ or $y_{i+1} = y_i - 1$, if $p_i \geq 0$.
4. Repeat the procedures in step 3 until the x and y coordinates are equal.

3.0 IMPLEMENTATION

With the path of the circle described, the pixels along the path can be set using the basic symmetry of the circle. Following is an example of Bresenham's circle algorithm in the C language, based on Michener's derivation.

```
circle(xc,yc,radius,width)
register int xc,yc,radius,width;
{
    register int y, x, p;
    x = 0;
    y = radius;
    p = 3 - 2 * radius;
    while (x < y) {
        setgrp(xc,yc,x,y,width);
        if (p < 0)
            p += 4 * x + 6;
        else {
            p += 4 * (x - y) + 10;
            y--;
        }
        x++;
    }
    if (y == x)
        setgrp(xc,yc,x,y,width);
}

setgrp(xc,yc,x,y,width)
register int xc,yc,x,y,width;
{
    if y-x<=(width/2)){
        hset(xc + y, yc + x,width);
        hset(xc - y, yc + x,width);
        hset(xc + y, yc - x,width);
        hset(xc - y, yc - x,width);
        vset(xc + x, yc + y,width);
        vset(xc - x, yc + y,width);
        vset(xc + x, yc - y,width);
        vset(xc - x, yc - y,width);
    }
    vset(xc + y, yc + x,width);
    vset(xc - y, yc + x,width);
    vset(xc + y, yc - x,width);
    vset(xc - y, yc - x,width);
    hset(xc + x, yc + y,width);
    hset(xc - x, yc + y,width);
    hset(xc + x, yc - y,width);
    hset(xc - x, yc - y,width);
}
```

 TL/EE10563

RRD-B30M75/Printed in U. S. A.

The *setgrp* routine in the previous example uses symmetry to set eight points of the circle. *Setgrp* has a special case to handle the boundaries of the eight sections. When the distance between the boundaries is less than half the width of the circle, both vertical and horizontal lines are imaged for each section. The *vset* routine sets *width* pixels vertically in the image, centered around the second argument. The *hset* routine sets *width* pixels horizontally, centered around the first argument.

The NS32GX32 implementation is very much like the C version, but is optimized for speed. Note the use of the *ADDR* instruction to do the two p_i computations, each in one line of 32000 assembly code.

Figure 1 shows this algorithm 'at work'. 20 circles of radius 350 pixels, and widths of 1 to 20 pixels are shown. A full listing of this test program is shown in Appendix A.

4.0 TIMING

The execution speed of this algorithm is dependent on the radius and the width of the circle, the test program presented here which draws 20 circles while progressively increasing the width in memory executes in 0.49 seconds on a NS32GX32 at 30 MHz. The time can be further reduced by using macros for the VLINE and HLINE routines and by using a better algorithm for drawing horizontal lines (i.e., not by emulating the 'sbits' NS32CG16 instruction).

5.0 CONCLUSIONS

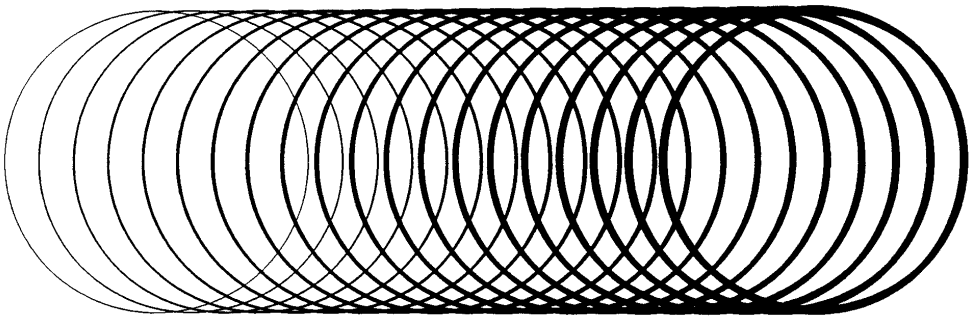
The NS32GX32's high code density and fast execution make it ideal for intensive graphics processing.

This algorithm does, however, show an apparent 'thinning' on the 45° boundaries, when the width of the circle is greater than five pixels. An alternate algorithm will be presented in a future application note. This algorithm is optimized for speed.

6.0 REFERENCES

1. Hearn, Donald and M. Pauline Baker, (1986). *Computer Graphics*, Englewood Cliffs, N.J., Prentice-Hall, 65–69.

2. Foley, James D. and Van Dam, Andries, (1982). *Fundamentals of Interactive Computer Graphics*, Reading, Massachusetts, Addison-Wesley, 441–446.



TL/EE/10563–1

FIGURE 1

APPENDIX A

```
#
#       circle.s - circle drawing program
#
        .set    xdim,2544       # number of bits in X-dimension
        .set    ydim,800        # number of bits in Y-dimension

#       The following should contain the address of the bitmap area in RAM
#       where the circle is to be drawn. It is supposed to be allocated by
#       the caller function _before_ calling the circle-drawing routine
#       '_test'. The area may be allocated by using the declaration + call:
#               short * bitmap;
#               bitmap = calloc(xdim*ydim/16, sizeof(short))
#
        .data
        .globl  _bitmap
hlfwdth:
        .double 0

        .text
#
#  Test is a 'C' callable function that creates figure 1.
#
#  The calling format of this function from 'C' is:
#
#    test(xstart, ystart, radius, ncircles);
#
        .globl  _test
_test:  save    [r3,r4,r5,r6,r7]
        movd    8(fp),r0        # x-coordinate of 1st circle center -› r0
        movd    12(fp),r1       # y-coordinate of 1st circle center -› r1
        movqd   1,r2            # width = 1 -› r2
        movd    16(fp),r3       # radius -› r3
        movd    20(fp),r7       # number of circles to draw -› r7
loop:   bsr     circle          # do a circle
        addr    80(r0),r0       # x-coordinate of center += 80
        addqd   1,r2            # width += 1
        acbd    -1,r7,loop      # loop for all circles
        restore [r3,r4,r5,r6,r7]
        ret     0
```

TL/EE/10563–2

```
#
#  Bresenham's circle algorithm, as expressed in
#  "Computer Graphics" by Donald Hearn & M. Pauline Baker
#  (1986. Prentice-Hall, ISBN 0-13-165382-2)
#
#       Inputs:
#               r0 = x coordinate of center of circle
#               r1 = y coordinate of center of circle
#               r2 = width  (in pixels)
#               r3 = radius (in pixels)
#
#       Outputs:
#               no registers altered
#               circle drawn in RAM
#
#  Notes:
#       This routine uses two special-case line drawing routines
#               a horizontal case (called 'hline')
#               a vertical case (called 'vline')
#       If the line is to have a width of > 25 pixels, the BIGSET
#       algorithm must be added to the 'hline' routine.
#
circle:
#       save    [r4,r5,r6,r7]
        movd    r4,tos
        movd    r5,tos
        movd    r6,tos
        movd    r7,tos
        movd    r2,r7           # get current width
        lshd    $-1,r7          # divide by 2
        movd    r7,hlfwdth      # and restore it away
        movqd   0,r4            # x1 = 0
        movd    r3,r5           # y1 = radius
        movqd   3,r6            # p = 3 - (radius * 2)
        subd    r3,r6
        subd    r3,r6
        br      circ_test
        .align  4
circ_loop:
        bsr     setgrp          # set a group of points
        cmpd    0,r6            # is p < 0 ?
        blt     pge0            # no, it is not. skip.
        addr    6(r6)[r4:d],r6  # p += 4 * x1 + 6
        addqd   1,r4            # x1++

circ_test:
        cmpd    r4,r5           # is x1 <= y1 ?
        ble     circ_loop       # it is. Loop.
        br      circ_out

        .align  4
pge0:   movd    r4,r7           # t = x1
        subd    r5,r7           # t = x1 - y1
        addr    10(r6)[r7:d],r6 # p += 4 * (x1 - y1) + 10
        addqd   -1,r5           # y1--
```

TL/EE/10563–3

```
#
#  Bresenham's circle algorithm, as expressed in
#  "Computer Graphics" by Donald Hearn & M. Pauline Baker
#  (1986. Prentice-Hall, ISBN 0-13-165382-2)
#
#       Inputs:
#               r0 = x coordinate of center of circle
#               r1 = y coordinate of center of circle
#               r2 = width  (in pixels)
#               r3 = radius (in pixels)
#
#       Outputs:
#               no registers altered
#               circle drawn in RAM
#
#  Notes:
#       This routine uses two special-case line drawing routines
#               a horizontal case (called 'hline')
#               a vertical case (called 'vline')
#       If the line is to have a width of > 25 pixels, the BIGSET
#       algorithm must be added to the 'hline' routine.
#
circle:
#       save    [r4,r5,r6,r7]
        movd    r4,tos
        movd    r5,tos
        movd    r6,tos
        movd    r7,tos
        movd    r2,r7           # get current width
        lshd    $-1,r7          # divide by 2
        movd    r7,hlfwdth      # and restore it away
        movqd   0,r4            # x1 = 0
        movd    r3,r5           # y1 = radius
        movqd   3,r6            # p = 3 - (radius * 2)
        subd    r3,r6
        subd    r3,r6
        br      circ_test
        .align  4
circ_loop:
        bsr     setgrp          # set a group of points
        cmpd    0,r6            # is p < 0 ?
        blt     pge0            # no, it is not. skip.
        addr    6(r6)[r4:d],r6  # p += 4 * x1 + 6
        addqd   1,r4            # x1++

circ_test:
        cmpd    r4,r5           # is x1 <= y1 ?
        ble     circ_loop       # it is. Loop.
        br      circ_out

        .align  4
pge0:   movd    r4,r7           # t = x1
        subd    r5,r7           # t = x1 - y1
        addr    10(r6)[r7:d],r6 # p += 4 * (x1 - y1) + 10
        addqd   -1,r5           # y1--
```

TL/EE/10563–4

```
        addd    r4,r0           # x += xl       ### (5)
        addd    r5,r1           # y += yl
        bsr     hline           # do a horizontal-line
        movd    r6,r0           # restore x and y
        movd    r7,r1
        addd    r4,r0           # x += xl       ### (6)
        subd    r5,r1           # y -= yl
        bsr     hline
        movd    r6,r0           # restore x and y
        movd    r7,r1
        subd    r4,r0           # x -= xl       ### (7)
        addd    r5,r1           # y += yl
        bsr     hline
        movd    r6,r0           # restore x and y
        movd    r7,r1
        subd    r4,r0           # x -= xl       ### (8)
        subd    r5,r1           # y -= yl
        bsr     hline
        movd    r6,r0           # restore x and y
        movd    r7,r1

        movd    tos,r7          # pop the two saved registers
        movd    tos,r6
        ret     0

sgl:    movd    r7,r1           # restore y
        addd    r4,r0           # x += xl       ### (1)
        addd    r5,r1           # y += yl
        bsr     hline
        bsr     vline
        movd    r6,r0           # restore x and y
        movd    r7,r1
        addd    r4,r0           # x += xl       ### (2)
        subd    r5,r1           # y -= yl
        bsr     hline
        bsr     vline
        movd    r6,r0           # restore x and y
        movd    r7,r1
        subd    r4,r0           # x -= xl       ### (3)
        addd    r5,r1           # y += yl
        bsr     hline
        bsr     vline
        movd    r6,r0           # restore x and y
        movd    r7,r1
        subd    r4,r0           # x -= xl       ### (4)
        subd    r5,r1           # y -= yl
        bsr     hline
        bsr     vline
        movd    r6,r0           # restore x and y
        movd    r7,r1

        addd    r5,r0           # x += yl       ### (5)
        addd    r4,r1           # y += xl
        bsr     vline
        bsr     hline
        movd    r6,r0           # restore x and y
```

TL/EE/10563–5

```
        movd    r7,r1
        addd    r5,r0           # x += y1       ### (6)
        subd    r4,r1           # y -= x1
        bsr     vline
        bsr     hline
        movd    r6,r0           # restore x and y
        movd    r7,r1
        subd    r5,r0           # x -= y1       ### (7)
        addd    r4,r1           # y += x1
        bsr     vline
        bsr     hline
        movd    r6,r0           # restore x and y
        movd    r7,r1
        subd    r5,r0           # x -= y1       ### (8)
        subd    r4,r1           # y -= x1
        bsr     vline
        bsr     hline
        movd    r6,r0           # restore x and y
        movd    r7,r1

        movd    tos,r7          # pop the two saved registers
        movd    tos,r6
        ret     0

#
#       vline: A vertical line drawing algorithm
#
#       Inputs:
#               r0 = x coordinate of line
#               r1 = centerpoint of y-coordinate of line
#               r2 = line length
#
#       Outputs:
#               None: line drawn in memory
#
        .align  4
vline:
# save  [r0,r1,r2,r3]   # save working registers
        movd    r0,tos
        movd    r1,tos
        movd    r2,tos
        movd    r3,tos

        subd    hlfwdth,r1      # y -= half of width to center vline
        movd    $xdim,r3        # bit offset = y * (xdim) + x
        muld    r3,r1
        addd    r0,r1
        movd    _bitmap,r0      # bitmap address in r0

# --- SBITPS (NS32CG16 microprocessor instruction) emulation code start
        .align  4
sbloop: sbitd   r1,0(r0)        # set required bit
        addd    r3,r1           # add the bit warp
        acbd    -1,r2,sbloop    # loop for the r11
# --- SBITPS emulation code end
        # restore       [r0,r1,r2,r3]   # restore registers
```

TL/EE/10563–6

```
        movd    tos,r3
        movd    tos,r2
        movd    tos,r1
        movd    tos,r0
        ret     0


#
#       hline: A horizontal line drawing algorithm
#
#       Inputs:
#               r0 = centerpoint of x coordinate of line
#               r1 = y-coordinate of line
#               r2 = line length
#
#       Outputs:
#               None: line drawn in memory
#
        .align  4
hline:
# save  [r0,r1,r3]      # save working registers
        movd    r0,tos
        movd    r1,tos
        movd    r3,tos
        subd    hlfwdth,r0      # x -= half of width to center values
        muld    $(xdim),r1      # bit off = (y * xdim) + x
        addd    r0,r1
        movd    _bitmap,r0      # bitmap address in r0
# ---   addr    sbitstab,r3     # address of sbits table
# --- SBITS emulation code start
        movqd   7,r3
        andd    r1,r3
        lshd    $5,r3           # * 32
        addd    r2,r3
        ashd    $-3,r1
        ord     sbitstab[r3:d],0(r0)[r1:b]
# --- SBITS emulation code end
        # restore       [r0,r1,r3]
        movd    tos,r3
        movd    tos,r1
        movd    tos,r0
        ret     0

#       The following table is used for emulation of the 'sbits'
#       instruction of the NS32CG16 microprocessor
#
        .data
sbitstab:
        .double h'00000000, h'00000001, h'00000003, h'00000007
        .double h'0000000f, h'0000001f, h'0000003f, h'0000007f
        .double h'000000ff, h'000001ff, h'000003ff, h'000007ff
        .double h'00000fff, h'00001fff, h'00003fff, h'00007fff
        .double h'0000ffff, h'0001ffff, h'0003ffff, h'0007ffff
        .double h'000fffff, h'001fffff, h'003fffff, h'007fffff
        .double h'00ffffff, h'01ffffff, h'03ffffff, h'07ffffff
        .double h'0fffffff, h'1fffffff, h'3fffffff, h'7fffffff
```

TL/EE/10563–7

```
.double h'00000000, h'00000002, h'00000006, h'0000000e
.double h'0000001e, h'0000003e, h'0000007e, h'000000fe
.double h'000001fe, h'000003fe, h'000007fe, h'00000ffe
.double h'00001ffe, h'00003ffe, h'00007ffe, h'0000fffe
.double h'0001fffe, h'0003fffe, h'0007fffe, h'000ffffe
.double h'001ffffe, h'003ffffe, h'007ffffe, h'00ffffe
.double h'01ffffe, h'03ffffe, h'07ffffe, h'0ffffffe
.double h'1ffffffe, h'3ffffffe, h'7ffffffe, h'ffffffe

.double h'00000000, h'00000004, h'0000000c, h'0000001c
.double h'0000003c, h'0000007c, h'000000fc, h'000001fc
.double h'000003fc, h'000007fc, h'00000ffc, h'00001ffc
.double h'00003ffc, h'00007ffc, h'0000fffc, h'0001fffc
.double h'0003fffc, h'0007fffc, h'000ffffc, h'001ffffc
.double h'003ffffc, h'007ffffc, h'00ffffc, h'01ffffc
.double h'03ffffc, h'07ffffc, h'0ffffffc, h'1ffffffc
.double h'3ffffffc, h'7ffffffc, h'ffffffc, h'ffffffc

.double h'00000000, h'00000008, h'00000018, h'00000038
.double h'00000078, h'000000f8, h'000001f8, h'000003f8
.double h'000007f8, h'00000ff8, h'00001ff8, h'00003ff8
.double h'00007ff8, h'0000fff8, h'0001fff8, h'0003fff8
.double h'0007fff8, h'000ffff8, h'001ffff8, h'003ffff8
.double h'007ffff8, h'00ffff8, h'01ffff8, h'03ffff8
.double h'07ffff8, h'0ffffff8, h'1ffffff8, h'3ffffff8
.double h'7ffffff8, h'fffffff8, h'fffffff8, h'fffffff8

.double h'00000000, h'00000010, h'00000030, h'00000070
.double h'000000f0, h'000001f0, h'000003f0, h'000007f0
.double h'00000ff0, h'00001ff0, h'00003ff0, h'00007ff0
.double h'0000fff0, h'0001fff0, h'0003fff0, h'0007fff0
.double h'000ffff0, h'001ffff0, h'003ffff0, h'007ffff0
.double h'00ffff0, h'01ffff0, h'03ffff0, h'07ffff0
.double h'0ffffff0, h'1ffffff0, h'3ffffff0, h'7ffffff0
.double h'fffffff0, h'fffffff0, h'fffffff0, h'fffffff0

.double h'00000000, h'00000020, h'00000060, h'000000e0
.double h'000001e0, h'000003e0, h'000007e0, h'00000fe0
.double h'00001fe0, h'00003fe0, h'00007fe0, h'0000ffe0
.double h'0001ffe0, h'0003ffe0, h'0007ffe0, h'000ffffe0
.double h'001fffe0, h'003fffe0, h'007fffe0, h'00ffffe0
.double h'01ffffe0, h'03ffffe0, h'07ffffe0, h'0ffffffe0
.double h'1ffffffe0, h'3ffffffe0, h'7ffffffe0, h'ffffffe0
.double h'ffffffe0, h'ffffffe0, h'ffffffe0, h'ffffffe0

.double h'00000000, h'00000040, h'000000c0, h'000001c0
.double h'000003c0, h'000007c0, h'00000fc0, h'00001fc0
.double h'00003fc0, h'00007fc0, h'0000ffc0, h'0001ffc0
.double h'0003ffc0, h'0007ffc0, h'000ffffc0, h'001ffffc0
.double h'003fffc0, h'007fffc0, h'00ffffc0, h'01ffffc0
.double h'03ffffc0, h'07ffffc0, h'0ffffffc0, h'1ffffffc0
.double h'3ffffffc0, h'7ffffffc0, h'ffffffc0, h'ffffffc0
.double h'ffffffc0, h'ffffffc0, h'ffffffc0, h'ffffffc0

.double h'00000000, h'00000080, h'00000180, h'00000380
.double h'00000780, h'00000f80, h'00001f80, h'00003f80
.double h'00007f80, h'0000ff80, h'0001ff80, h'0003ff80
.double h'0007ff80, h'000fff80, h'001fff80, h'003fff80
.double h'007fff80, h'00ffff80, h'01ffff80, h'03ffff80
.double h'07ffff80, h'0fffff80, h'1fffff80, h'3fffff80
.double h'7fffff80, h'ffffff80, h'ffffff80, h'ffffff80
.double h'ffffff80, h'ffffff80, h'ffffff80, h'ffffffff
```

TL/EE/10563–8

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998