to Draw a Circle

with a Digital Differential Algorithm

http://csustan.csustan.edu/~ tom

Tom Carter

Computer Science
CSU Stanislaus

tom@csustan.csustan.edu

http://csustan.csustan.edu/~
tom/Lecture-Notes/Graphics/DDA-circle/DDA-circle.pdf

October 9, 2014

Our general topics:

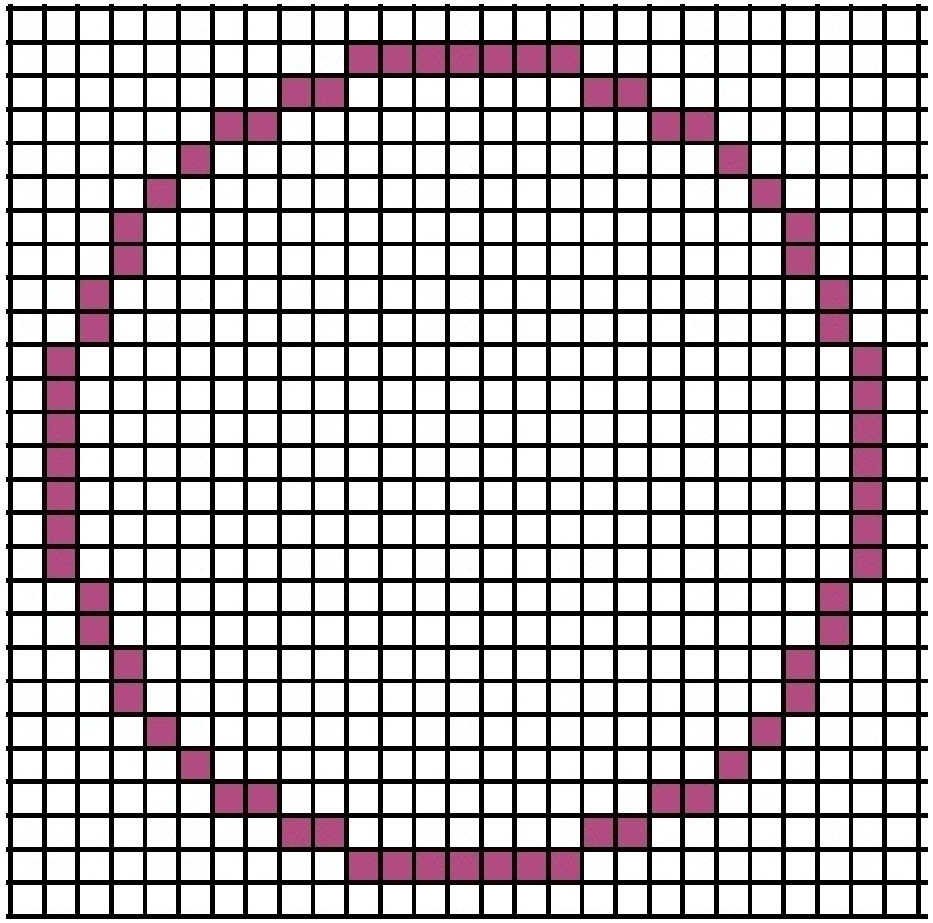
Drawing a Circle in a Raster $\leftarrow$

There are times when we want to draw a circle on a raster device. We'll develop a reasonably optimized algorithm for doing this. The approach we'll take is to develop a *digital differential algorithm*.

The *raster* is a two-dimensional array of pixels. We will assume we are given a center (CX, CY) (in pixel coordinates) and a radius R (also in pixel coordinates).

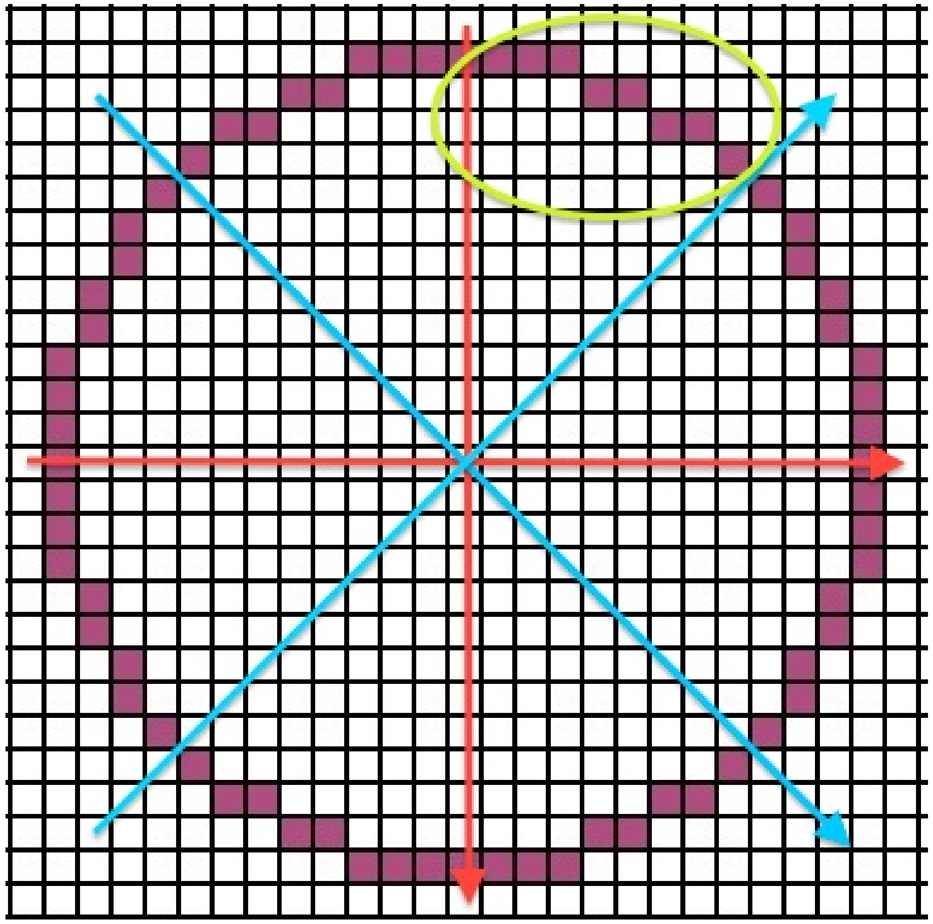
One nice thing about a circle is that it has many symmetries. In fact, we can draw a circle by specifying pixels we want to turn on in just one octant, say $X = 0$ to $X = Y$.

Here's an example of what this will look like:



A circle in a raster

Using symmetries, we only work in one octant:



Symmetries - just one octant

We'll make some simplifying assumptions in the derivation.

1. Our pixel coordinates go from left to right, and bottom to top, the way they do in the first quadrant in mathematics.

2. Our pixels are square (so we don't need to worry about aspect ratio . . .).

3. Our pixels are either off or on (i.e., no gray-scale . . .).

The DDA Circle Algorithm (first steps)

$\leftarrow$

To keep things simple, we'll do our calculations as though center of the circle is at $(0, 0)$. We'll assume we have a procedure (circlePoints(CX, CY, X, Y)) that takes advantage of the symmetries, and turns on 8 (or 4 . . .) pixels once we specify the center of the circle (CX, CY) and the pixel to start with relative to the center.

With the assumptions we have made, our stepwise algorithm will have the outline:

```
X = 0
Y = R
circlePoints(CX, CY, X, Y)
While (X < Y)
     X = X + 1
     determine Y value
     circlePoints(CX, CY, X, Y)
EndWhile
```

Thus, if we can figure out a fast way to determine the Y value to turn on, we will be done.

We'll do this in an iterative fashion. As we step along in the X direction, we can see that either Y stays the same, or Y decreases by 1. All we really need to do is figure out when to decrement Y.

Algorithm, version 2:

```
X = 0
Y = R
circlePoints(CX, CY, X, Y)
While (X < Y)
     X = X + 1
     If we should decrement Y
          Y = Y - 1
     EndIf
     circlePoints(CX, CY, X, Y)
EndWhile
```

We need to develop a *decision function* (or *decision variable*), sometimes called an *indicator variable, that will tell us when to decrement Y.*

Let's start with the function

$$F(x,y) = x^2 + y^2 - r^2$$

For any point (x,y) *on the circle, we have* $F(x,y) = 0$*. If* $F(x,y) < 0$*, the point is inside the circle, and if* $F(x,y) > 0$*, the point is outside the circle.*

So, suppose the current pixel is (X,Y)*. We need to decide whether the next pixel should be* $(X+1,Y)$ *or* $(X+1,Y-1)$*. We'll create a decision variable that looks at the "average" of* *the two pixels we are considering, call it M, and checks whether M is inside or outside the circle:*

$$\begin{aligned} D &= F(M) \\ &= F(X+1, Y-\frac{1}{2}) \\ &= (X+1)^2 + (Y-\frac{1}{2})^2 - R^2 \\ &= X^2 + 2X + 1 + Y^2 - Y + \frac{1}{4} - R^2 \end{aligned}$$

If $D < 0$, then M is still inside the circle, so we don't want to decrement Y yet.

We don't want to calculate D at each step (floating point operations, etc.), but instead want to update the value incrementally.

There are two possibilities.

If we didn't decrement Y, then the new value of D will be

$$\begin{aligned} D_{new} &= (X+2)^2 + (Y - \frac{1}{2})^2 - R^2 \\ &= X^2 + 4X + 4 + Y^2 - Y + \frac{1}{4} - R^2 \end{aligned}$$

and the increment in D will then be

$$\begin{aligned} D_{new} - D_{old} &= X^2 + 4X + 4 + Y^2 - Y + \frac{1}{4} - R^2 \\ &\qquad - (X^2 + 2X + 1 + Y^2 - Y + \frac{1}{4} - R^2) \\ &= 2X + 3 \end{aligned}$$

If we did decrement Y, then the new value of D will be

$$\begin{aligned} D_{new} &= (X+2)^2 + (Y - \frac{3}{2})^2 - R^2 \\ &= X^2 + 4X + 4 + Y^2 - 3Y + \frac{9}{4} - R^2 \end{aligned}$$

and the increment in D will then be

$$\begin{aligned} D_{new} - D_{old} &= X^2 + 4X + 4 + Y^2 - 3Y + \frac{9}{4} - R^2 \\ &\qquad - (X^2 + 2X + 1 + Y^2 - Y + \frac{1}{4} - R^2) \\ &= 2X - 2Y + 5. \end{aligned}$$

The other thing we need to do is establish the starting value for D:

$$\begin{aligned} D_0 &= F(1, R - \frac{1}{2}) \\ &= 1 + (R - \frac{1}{2})^2 - R^2 \\ &= 1 + R^2 - R + \frac{1}{4} - R^2 \\ &= \frac{5}{4} - R \end{aligned}$$

Given that we want to work with integers (not floating point), we will go ahead and use $D_0 = 1 - R$.

Technically, this means that we should do the comparison $D < -\frac{1}{4}$ for our decision variable, but since we are working with integers, this is no different from $D < 0$.

We can thus put together our algorithm as:

```
X = 0
Y = R
D = 1 - R
circlePoints(CX, CY, X, Y)
While (X < Y)
     If D < 0
          D = D + 2 * X + 3
     Else
          Y = Y - 1
          D = D + 2 * (X - Y) + 5
     EndIf
     X = X + 1
     circlePoints(CX, CY, X, Y)
EndWhile
```

The DDA Circle Algorithm ←

Our final algorithm looks like this:

```
void circleDDA(int xCenter, int yCenter, int radius)
{
    int x;
    int y;
    int p;

    x = 0;
    y = radius;
    p = 1 - radius; /**/

    circlePoints(xCenter, yCenter, x, y);
    while (x < y) {
        x++;
        if (p < 0) {
            p += 2*x+3;  /**/
        } else {
            y--;
            p += 2*(x-y)+ 5;  /**/
        }
        circlePoints(xCenter, yCenter, x, y);
    }
}
```

The one other piece is the "circlePoints" procedure, that takes advantage of the symmetries of the circle to plot 8 (or 4) points.

```
void circlePoints(int cx, int cy, int x, int y)
{
    int cxpx = cx + x;
    int cxmx = cx - x;
    int cxpy = cx + y;
    int cxmy = cx - y;
    int cypx = cy + x;
    int cymx = cy - x;
    int cypy = cy + y;
    int cymy = cy - y;
```

```
if (x == 0) {
if ((0 <= cx) && (cx < RDim)
     && (0 <= (cypy)) && ((cypy) < RDim))
            theRaster[cx][cypy] = 1;
if ((0 <= cx) && (cx < RDim)
     && (0 <= (cymy)) && ((cymy) < RDim))
            theRaster[cx][cymy] = 1;
if ((0 <= (cxpy)) && ((cxpy) < RDim)
     && (0 <= (cy)) && ((cy) < RDim))
            theRaster[cxpy][cy] = 1;
if ((0 <= (cxmy)) && ((cxmy) < RDim)
     && (0 <= (cy)) && ((cy) < RDim))
            theRaster[cxmy][cy] = 1;
} else
if (x == y) {
if ((0 <= (cxpx)) && ((cxpx) < RDim)
     && (0 <= (cy + y)) && ((cy + y) < RDim))
            theRaster[cxpx][cy + y] = 1;
if ((0 <= (cxmx)) && ((cxmx) < RDim)
     && (0 <= (cy + y)) && ((cy + y) < RDim))
            theRaster[cxmx][cy + y] = 1;
if ((0 <= (cxpx)) && ((cxpx) < RDim)
     && (0 <= (cymy)) && ((cymy) < RDim))
            theRaster[cxpx][cymy] = 1;
if ((0 <= (cxmx)) && ((cxmx) < RDim)
     && (0 <= (cymy)) && ((cymy) < RDim))
            theRaster[cxmx][cymy] = 1;
} else
if (x < y) {
if ((0 <= (cxpx)) && ((cxpx) < RDim)
     && (0 <= (cy + y)) && ((cy + y) < RDim))
            theRaster[cxpx][cy + y] = 1;
```

```
if ((0 <= (cxmx)) && ((cxmx) < RDim)
     && (0 <= (cy + y)) && ((cy + y) < RDim))
            theRaster[cxmx][cy + y] = 1;
if ((0 <= (cxpx)) && ((cxpx) < RDim)
     && (0 <= (cymy)) && ((cymy) < RDim))
            theRaster[cxpx][cymy] = 1;
if ((0 <= (cxmx)) && ((cxmx) < RDim)
     && (0 <= (cymy)) && ((cymy) < RDim))
            theRaster[cxmx][cymy] = 1;
if ((0 <= (cx + y)) && ((cx +y) < RDim)
     && (0 <= (cypx)) && ((cypx) < RDim))
            theRaster[cx + y][cypx] = 1;
if ((0 <= (cxmy)) && ((cxmy) < RDim)
     && (0 <= (cypx)) && ((cypx) < RDim))
            theRaster[cxmy][cypx] = 1;
if ((0 <= (cx + y)) && ((cx + y) < RDim)
     && (0 <= (cymx)) && ((cymx) < RDim))
            theRaster[cx + y][cymx] = 1;
if ((0 <= (cxmy)) && ((cxmy) < RDim)
     && (0 <= (cymx)) && ((cymx) < RDim)) /**/
            theRaster[cxmy][cymx] = 1;
}
}
```

To top ←