Bresenham’s Circle Drawing Algorithm

A Quick Comparison

There are lots of great articles about Bresenham’s Circle Drawing Algorithm. Here, for instance. So I won’t describe the theory in this article. Instead, I’ll show a Plain English implementation of the same, and compare it with this C version from the aforementioned article:

```
#include (some obscure library name)
#include (some obscure library name)
#include (some obscure library name)

// Function to put pixels
// at subsequence points
void drawCircle( int xc,  int yc,  int x,  int y)
{
     putpixel(xc+x, yc+y, RED);
     putpixel(xc-x, yc+y, RED);
     putpixel(xc+x, yc-y, RED);
     putpixel(xc-x, yc-y, RED);
     putpixel(xc+y, yc+x, RED);
     putpixel(xc-y, yc+x, RED);
     putpixel(xc+y, yc-x, RED);
     putpixel(xc-y, yc-x, RED);
}

// Function for circle-generation
// using Bresenham's algorithm
void circleBres( int xc,  int yc,  int r)
{
     int x = 0, y = r;
     int d = 3 - 2 * r;
     while (y >= x)
     {
         // for each pixel we will
         // draw all eight pixels
         drawCircle(xc, yc, x, y);
         x++;
```

```
          // check for decision parameter
          // and correspondingly
          // update d, x, y
          if (d > 0)
          {
              y--;
              d = d + 4 * (x - y) + 10;
          }
          else
              d = d + 4 * x + 6;
          drawCircle(xc, yc, x, y);
          delay(50);
      }
}


// driver function
int main()
{
     int xc = 50, yc = 50, r2 = 30;
     int gd = DETECT, gm;
     initgraph(&gd, &gm,  "" );   // initialize graph
     circleBres(xc, yc, r);     // function call
     return 0;
}
```

Now here's the Plain English version:

```
To run:
Start up.
Clear the screen.
Draw a circle using the screen's center and 3 inches.
Refresh the screen.
Wait for the escape key.
Shut down.

To draw a circle with a center spot and a radius:
Put the radius and 0 into a spot.
Put the radius times -2 plus 1 pixel with 1 pixel into a change pair.
Loop.
If the spot's y is greater than the spot's x, break.
Plot eight spots given the center and the spot.
Add 1 pixel to the spot's y.
Add the change's y to some error twips.
Add 2 pixels to the change's y.
If the error times 2 plus the change's x is less than or equal to 0, repeat.
Subtract 1 pixel from the spot's x.
Add the change's x to the error.
Add 2 pixels to the change's x.
Repeat.
```

```
To plot eight spots given a center spot and a spot:
Privatize the spot.
Loop.
Plot the center plus the spot.
Negate the spot's x. Plot the center plus the spot.
Negate the spot's y. Plot the center plus the spot.
Negate the spot's x. Plot the center plus the spot.
Negate the spot's y.
Flip the spot. If a counter is past 1, break.
Repeat.
```

Lines of code: Plain English, 33; C, 56.
Almost 70% more in the C version.

Characters: Plain English, 1048; C, 1209.
About 15% more in the C version.

Now please don't get me wrong. C is a great language if you don't mind memorizing (and/or looking up) how to type things like:

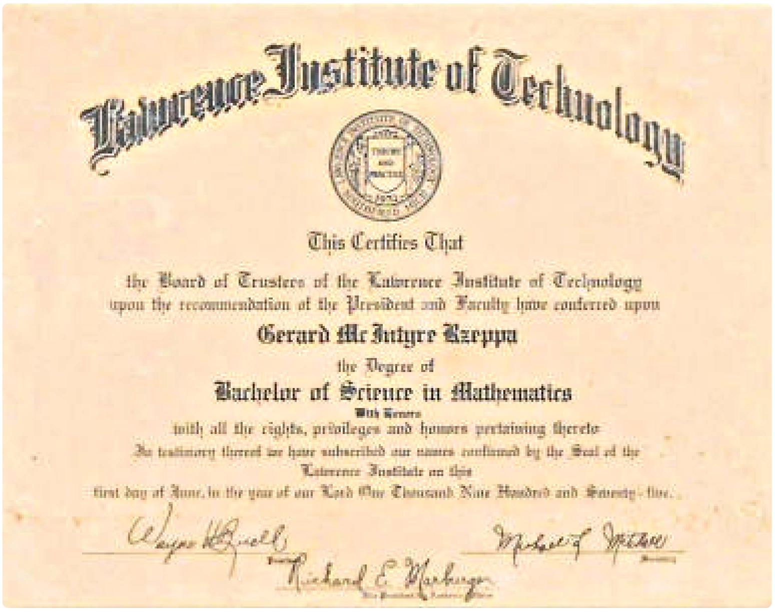
```
initgraph(&gd, &gm, "");
```

It just seems kind of *foreign* and *unnatural* to me. After all, my native tongue is English, and when I'm thinking about a program, I typically think in English "pseudo-code." The cool thing about Plain English is that the pseudo-code I had in mind actually runs.

Bear with me, please. This next part is really interesting.

Mathemagician Pie

As you may know from a previous article of mine, I joined the Mathemagician's Club in my youth. This is my membership card:



Lawrence Institute of Technology

This Certifies That
the Board of Trustees of the Lawrence Institute of Technology
upon the recommendation of the President and Faculty have conferred upon
Gerard McIntyre Rzeppa
the Degree of
Bachelor of Science in Mathematics
With Honors
with all the rights, privileges and honors pertaining thereto
In testimony thereof we have subscribed our names confirmed by the Seal of the
Lawrence Institute on this
first day of June, in the year of our Lord One Thousand Nine Hundred and Seventy-five.

[illegible] [illegible]
Richard E. Marburger

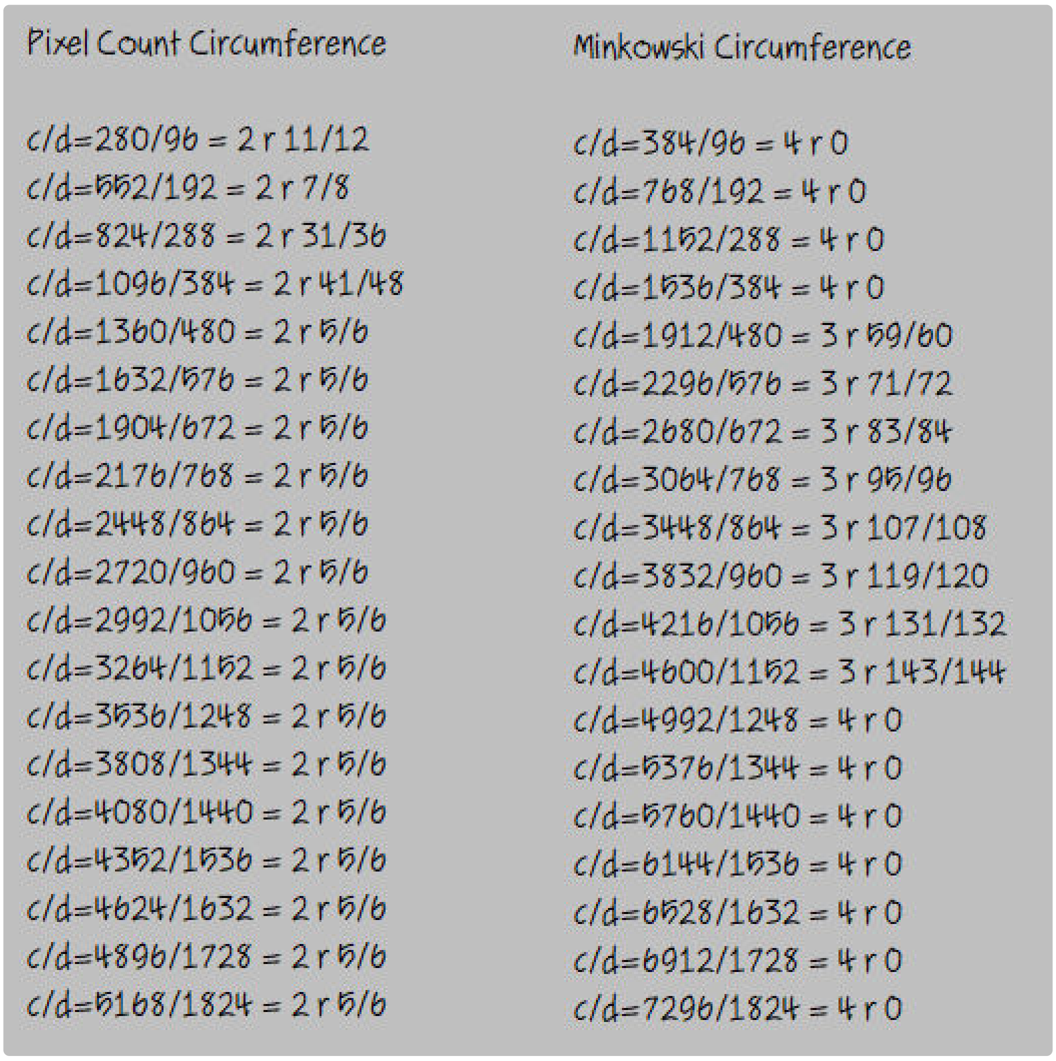
But I quickly became an outcast for taking Kronecker's side in the Kronecker/Cantor kerfuffle. Which is why, decades later, Plain English is still an integer-only language. Baby steps toward a more *rational* and *natural* mathematics. And with the algorithms above, we now have what we need to take another baby step.

Pi, in the mathemagician's world, is defined as the circumference of a circle, c, divided by the circle's diameter, d, thus:

```
Pi = c/d
```

It is a nasty, transcendental (ie, unreal) number that can't be fully calculated by anyone or anything. But using the routines above, we *are* able to calculate precise values for c/d, because we're dealing, not with mathemagical abstractions, but with real-life pixels that we can actually see and count (using whole numbers).

The question is, How should we measure the circumference of a circle? I know of two, obvious, whole-number methods: (1) we can count the pixels we draw, or (2) we can calculate the sum of the Minkowski distances between those pixels. These are the figures we get for circles of increasing diameter using each method:



Pixel Count Circumference	Minkowski Circumference
c/d=280/96 = 2 r 11/12	c/d=384/96 = 4 r 0
c/d=552/192 = 2 r 7/8	c/d=768/192 = 4 r 0
c/d=824/288 = 2 r 31/36	c/d=1152/288 = 4 r 0
c/d=1096/384 = 2 r 41/48	c/d=1536/384 = 4 r 0
c/d=1360/480 = 2 r 5/6	c/d=1912/480 = 3 r 59/60
c/d=1632/576 = 2 r 5/6	c/d=2296/576 = 3 r 71/72
c/d=1904/672 = 2 r 5/6	c/d=2680/672 = 3 r 83/84
c/d=2176/768 = 2 r 5/6	c/d=3064/768 = 3 r 95/96
c/d=2448/864 = 2 r 5/6	c/d=3448/864 = 3 r 107/108
c/d=2720/960 = 2 r 5/6	c/d=3832/960 = 3 r 119/120
c/d=2992/1056 = 2 r 5/6	c/d=4216/1056 = 3 r 131/132
c/d=3264/1152 = 2 r 5/6	c/d=4600/1152 = 3 r 143/144
c/d=3536/1248 = 2 r 5/6	c/d=4992/1248 = 4 r 0
c/d=3808/1344 = 2 r 5/6	c/d=5376/1344 = 4 r 0
c/d=4080/1440 = 2 r 5/6	c/d=5760/1440 = 4 r 0
c/d=4352/1536 = 2 r 5/6	c/d=6144/1536 = 4 r 0
c/d=4624/1632 = 2 r 5/6	c/d=6528/1632 = 4 r 0
c/d=4896/1728 = 2 r 5/6	c/d=6912/1728 = 4 r 0
c/d=5168/1824 = 2 r 5/6	c/d=7296/1824 = 4 r 0

If we figure the circumference using a pixel count circumference, we get a rational value for Pi that quickly converges to 2 and 5/6. On the other hand, if we calculate the circumference by tracing a Minkowski path through those pixels, we get a rational value for Pi that is very close to 4. This is exactly what we'd expect. The Pixel Count Pi is a little less than the mathemagician's value, and the Minkowski Pi is a little more. But both are rational numbers based on real-world pixels.

A "pixelated" circle, I hope you can see, is *not* an approximation of a mathemagician's "ideal" circle; that's putting the cart before the horse. It's the mathemagician's *imaginary* circle that approximates the real-life pixelated circle. Which, as we've just seen, can be efficiently drawn, and measured, and analyzed, in Plain English, using whole numbers only.

Call me crazy, but don't call me late for dinner!