Bresenham's Line and Circle Algorithms

Introduction. Graphics images consist of individual *Picture Elements* (*Pixels*), which are a single point in the image. For color images, each pixel has color components for the Red, Green, and Blue parts of the color, which are generally specified individually. To draw a line or a circle on a graphical display, we need an algorithm to do *scan conversion*. That is, for a line we are just given the two endpoints as a pair of (x,y) coordinates, and we must determine which pixels are part of the specified line and set those pixels to the appropriate color. Similarly, for circles we are just given the center point (x,y) and the radius. Here, we will discuss two algorithms for scan converting lines and circles. For further details, both of these algorithms are described in "Computer Graphics" by Jim Foley and Andries van Dam.

Bresenham's Mid–Point Line Algorithm. We could easily design an algorithm to draw a line, using floating point values for the slope of the line, and then rounding to an integer to set the appropriate pixel. However, floating point computation in a CPU is substantially more complex (and takes longer) than integer arithmetic. Given this, J. E. Bresenham developed an algorithm for line drawings that *uses integer arithmetic only*. The algorithm is given below:

```
void MidpointLine(int x0, int y0, int x1, int y1)
{
  int dx = x1 - x0;
  int dy = y1 - y0;
  int d  = 2 * dy - dx;
  int incrE = 2 * dy;          // East increment
  int incrNE = 2 * (dy - dx);  // Northeast increment
  int x = x0;                  // Initial x
  int y = y0;                  // Initial y
  SetPixel(x, y);              // Initial line point
  while(x < x1)
    {
      if (d <= 0)
        {
           d += incrE;         // Move East
        }
      else
        {
           d += incrNE;        // Move Northeast
           y++;                // and advance y
        }
      x++;
      SetPixel(x, y);          // Next pixel in line
    }
}
```

It is **IMPORTANT** to note that this algorithm only works for lines where $x1 > x0$ and for slope (dy/dx) between 0 and 1.0. You must modify the algorithm to work for the other cases as well. Further, take care to handle the case of infinite slope lines ($x1 == x0$).

Bresenham's Midpoint Circle Algorithm. For drawing circles, we could easily develop an algorithm that makes use of trigonometric functions such as sin and cosine to find the points on a circle. However, as in the case of line drawing, efficiency is of importance, and we would like an algorithm that uses simple integer arithmetic as much as possible, and in particular does not use expensive trig functions. Bresenham developed a circle drawing algorithm that does exactly this, using mostly integer arithmetic, as follows. This algorithm assumes the circle center is at (0,0).

```
void MidpointCircle(int r)
{
  int x = 0;
  int y = radius;
  double d = 5.0/4.0 - radius;
  SetPixel(x,y);
  while(y > x)
    {
      if (d < 0)
        {
          d += 2.0 * x + 3.0; // Select East
        }
      else
        {
          d += 2.0 * (x-y) + 5.0; // Select SE
          y--;
        }
      x++;
      SetPixel(x, y);
    }
}
```

It is **IMPORTANT** to note that this algorithm only draws one–eighth of a circle, in the range of 0 to 45 degrees. You must modify the algorithm to draw properly the remaining seven eighth's of the circle.