Bresenham Algorithm: Implementation and Analysis in Raster Shape

Ford Lumban Gaol
Bina Nusantara University, Jakarta
Indonesia
Email: fgaol@binus.edu

***Abstract*—One of the most important aspect that have to solve in raster objects is to describe the structure of the individual objects and their coordinate locations within the scene. We required to implement the graphics output primitives. The Output primitives are very important since the performance of the graphics depend on the Primitives. Point positions and straight-line segments are the simplest geometric primitives. We focus on these parts in this research. In this paper we will implement and analysis how accurate and efficient raster line-generating algorithm, develop by Bresenham, that uses only incremental integer calculations. The implementation will be expanded to display circles and other curves. The analysis will be focus on numerical results, error produced, computation speed, and display. The language that used in this implementation is C++ with OpenGL.**

***Index Terms*—Bresenham Algorithm, numerical result, error produced, computation speed, display, OpenGL**

I. Introduction

A general software package for graphics applications, sometimes referred to as a computer-graphics application programming interface (CG API), provides a library of functions that we can use within a programming languages such as C++ to create pictures (Haque *et al* 2006; Ring Che 2003] . One the first thing to do when creating a picture is to describe the component parts of the scene to be displayed. Picture components could be trees and terrain, furniture and walls, storefronts, and street scenes, automobiles and billboards, atoms and molecules, or stars and galaxies. For each type of scene, we need to describe the structure of the individual objects and their coordinate locations within the scene. Those functions in graphics package that we use to describe the structure of the individual objects and their coordinate locations within the scene (Prabukumar 2007; Ray, B.K 2006; Stucki 1991; Staunton 1989).

These functions in a graphics package that we use to describe the various picture components are called the graphics output primitives, or simply primitives. The output primitives describing the geometry of objects are typically referred to as geometric primitives. Point positions and straight-line segments are the simplest geometric primitives. Additional geometric primitives that can be available in a graphics package included circles, and other conics sections, quadratic surfaces, spline curves and surfaces, and polygon color areas. (Ammeraal 1987; Bresenham 1971; Bresenham 1977)

The rest of this paper organized as follows: Part 2 will discussed about Bresenham algorithm for line, circle, and polygon. next the implementation and conclusion.

II. Brensenham's Algorithm for Line, Circle, and Polygon

In this section, the accuracy and efficiency of raster-line generating algorithm, developed by Bresenham, that uses only incremental integer calculations will be elaborated. The Bresneham's line algorithm will be adapted to display circles and other curves. (Mersereau 1979; Rappoport 1991; Pitteway 1985; Klassen 1991).

To illustrate Bresenham's approach, we first consider the scan-conversion process for lines with positive slope less than 1.0. Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left endpoint (x_0, y_0) of a given line, we step to each successive columns (x position) and plot the pixel whose scan-line y value is closet to the line path. Assuming that the pixel at (x_k, y_k) is to be displayed, next to be decide which pixel to plot in column $x_{k+1} = x_k + 1$. The choices are the pixels at positions (x_k+1, y_k) and (x_k+1, y_k+1).

At sampling position xk+1, label the vertical pixel separations from the mathematical line path as d_{lower} and d_{upper} The y coordinate on the mathematical in at pixel column position x_k+1 is calculated as

$$y = m(x_k + 1) + b \tag{1}$$

Then

$$\begin{aligned} d_{lower} &= y - y_k \\ &= m(x_k + 1) + b - y_k \end{aligned} \tag{2}$$

and

$$\begin{aligned} d_{upper} &= (y_k + 1) - y \\ &= y_k + 1 - m(x_k + 1) - b \end{aligned}$$

To determine which of the two pixels is closet to the line path, we can set up an efficient test that is based on the difference between the two pixel separations:

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1 \tag{3}$$

A decision parameter p_k for the k-th step in the line algorithm can be obtained by rearranging (3) so that it involves only integer calculations. We accomplish this by substituting $m = \frac{\Delta y}{\Delta x}$, where Δy and Δx are the vertical and

doi:10.4304/jcp.8.1.69-78

horizontal separations of the endpoints positions, and define the decisions parameter as

$$p_k = \Delta x(d_{lower} - d_{upper}) = 2\Delta y.x_k - 2\Delta x.y_k + c \quad (4)$$

The sign of p_k is the same as the sign of $d_{lower} - d_{upper}$, since $\Delta x > 0$. Parameter c is constant and has the value $2\Delta y + \Delta x(2b-1)$, which is independent of the pixel positions and will be eliminated n the recursive calculations for p_k. if the pixel at y_k is "closer" to the line path than the pixel at y_k+1 (that is $d_{lower} < d_{upper}$), the decision parameter p_k is negative. In that case, we plot the lower pixel; otherwise we plot upper pixel.

Coordinate changes along the line occur in unit steps in either the x or y directions. Therefore, we can obtain the values of successive decision parameters using incremental integer calculations. At step k+1, the decision parameter is evaluated from (4) as

$$p_{k+1} = 2\Delta y.x_{k+1} - 2\Delta x.y_{k+1} + c \quad (5)$$

Subtracting (4) from the preceding equation, we have

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

But $x_{k+1} = x_k + 1$ so that

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k) \quad (6)$$

where the term yk+1 – yk is either 0 or 1, depending on the sign of parameter p_k.

This recursive calculation of decision parameters is performed at each integer x position, starting at the left coordinate endpoints of the line. The first parameter, p0 is evaluated from (4) at the starting pixel position (x_0, y_0) and with m evaluated as $\frac{\Delta y}{\Delta x}$:

$$p_0 = 2\Delta y - \Delta x \quad (7)$$

Herewith the summarizing of Brensehma line drawing for a line with a positive slope less than 1 in the following outline of the algorithm. The constant $2\Delta y$ and $2\Delta y - 2\Delta x$ are calculated once for each line to be scan converted, so the arithmetic involves only integer addition and subtraction of these constants.

Algorithm Bresenham's Line-Drawing Algorithm for $|m| < 1.0$

1. Input the two-line endpoints and store the left endpoint in (x_0, y_0).
2. Set the color for frame-buffer position (x_0, y_0); i.e. plot the first point.
3. Calculate the constants Δx, Δy, $2\Delta y$ & $2\Delta y$-$2\Delta x$, and obtain the starting value for the decision parameter as: $p_0 = 2\Delta y - \Delta x$
4. At each x_k along the line, starting at k = 0, perform the following test. If $p_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and $p_{k+1} = p_k + 2\Delta y$. Otherwise, the next point to plot is $(x_k + 1, y_k+1)$ and $p_{k+1} = p_k + 2\Delta y - 2\Delta x$.
5. Perform step 4 Δx –1 times.

As in the raster line algorithm, we sample at unit intervals and determine the closet pixel position to the specified circle path at each step. For a give radius r and screen center positions (x_c, y_c), we can first set up our algorithm to calculate pixel positions around a circle path centered at the coordinate origin (0,0). Then each calculated position (x, y) is moved to its proper screen position by adding x_c to x and y_c to y. Along the circle section from x =0 to x = y in the first quadrant, the slope of the curve varies from 0 to –1.0. Therefore, we can take unit steps in the positive x direction over this octant and use a decision parameter to determine which of the two possible pixel positions in any column is vertically closer to the circle path. Positions in the other seven octants are then obtained by symmetry.

To apply the midpoint method, we define the circle function as

$$f_{circle} = x^2 + y^2 - r^2 \quad (8)$$

Any point (x,y) on the boundary of the circle with radius r satisfies the equation $f_{circle}(x, y) = 0$. If the point is in the interior of the circle, the circle function is negative. An dif the point is outside the circle, the circle function is positive. To summarize, the relative position of any point (x,y) can be determined by

$$f_{circle}(c, y) = \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the circle boundary.} \\ = 0, & \text{if } (x, y) \text{ is on the cirle boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the circle boundary} \end{cases} \quad (9)$$

checking the sign of the circle function:

The test on (9) are performed for the midpoints between pixels near the circle path at each sampling step. Thus, the circle function is the decision parameter in the midpoint algorithm, and we can set up incremental calculations for this function as we did in the line algorithm.

Assuming that we have just plotted the pixel at (xk, yk), we next need to determine whether the pixel at position (x_k+1, yk) or the one at position $(x_k +1, y_k - 1)$ is closer to the circle. Our decision parameter is the circle function (8) evaluated at the midpoint between these two pixels:

$$p_k = f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$p_k = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 \quad (10)$$

If $p_k < 0$, this midpoint is inside the circle and the pixel on the scan line y_k is closer to the circle boundary. Otherwise, the midposition is outside or on the circle boundary, and we select the pixel on the scan line y_k –1.

Successive decision parameters are obtained using incremental calculations. We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1}+1 = x_k + 2$;

$$p_{k+1} = f_{circle}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

$$= [(x_k + 1) + 1)]^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2$$

or

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1 \quad (11)$$

where y_{k+1} is either y_k or y_k-1, depending on the sign of p_k.

Increments for obtaining p_{k+1} are neither $2x_{k+1}$+1 (if p_k is negative) or $2x_{k+1}$ +1-$2y_{k+1}$. Evaluation of the terms 2xk+1 and 2yk+1 can also be done incrementally as

$$2x_{k+1} = 2x_k + 2$$

$$2y_{k+1} = 2y_k + 2$$

At the start position (0,r), these two terms have the value 0 and 2*r*, respectively. each successive value for the $2x_{k+1}$ term is obtained by adding 2 to the previous value, and each successive value for the $2y_{k+1}$term is obtained by subtracting 2 from the previous value.

The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$:

$$p_0 = f_{circle}\left(1, r - \frac{1}{2}\right)$$

$$= 1 + \left(r - \frac{1}{2}\right)^2 - r^2 \quad (12)$$

or

$$p_0 = \frac{5}{4} - r$$

Below the summarize of the steps in the midpoint circle algorithm as follows:

Midpoint Circle Algorithm

[1] Input radius r and circle center (x_c, y_c), then set the coordinates for the first point on the circumference of a circle centered on the origin as:

$$\boldsymbol{(x_0, y_0) = (0, r).}$$

[2] Calculate the intial value of the decision parameter as

$$\boldsymbol{p_0} = \frac{5}{4} - r$$

[3] At each x_k position, starting at k =0, perform the following test. If $p_k < 0$, the next point along the circle centered on (0,0) is (x_{k+1}, y_k) and $\boldsymbol{p_{k+1} = p_k + 2x_{k+1} + 1}$

Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ and $\boldsymbol{p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}}$

where $\boldsymbol{2x_{k+1} = 2x_k + 2}$ and $\boldsymbol{2y_{k+1} = 2y_k - 2.}$

[4] Determine sysmetri points in the other seven points.

[5] Move each calculated pixel position (x, y) onto the circular path centered at (x_c, y_c) and plot the coordinate values:

$x = x + x_c$ $\qquad$ $y = y + y_c$

6. Repeat steps 3 through 5 until $x \geq y$

The approach for midpoint ellipse algorithm is similar to that used in displaying a raster circle. Given parameter r_x, r_y and (x_c, y_c), we determine curve positions (x, y) for an ellipse in standard position centered on the origin, the we shift all the points using a fixed offset so that the ellipse is centered at (x_c, y_c).

We define an ellipse function with $(x_c, y_c) = (0,0)$ as

$$f_{ellipse}(x, y) = r_y^2 x_x^2 + r_x^2 r^2 - r_x^2 r_y^2 \quad (13)$$

which has the following properties:

$$f_{circle}(c, y) = \begin{cases} < 0, & \text{if } (x, y) \text{ is inside the ellipse boundary.} \\ = 0, & \text{if } (x, y) \text{ is on the ellipse boundary} \\ > 0, & \text{if } (x, y) \text{ is outside the ellipse boundary} \end{cases} \quad (14)$$

Thus, the ellipse function $f_{ellipse}(x,y)$ serves as the decision parameter in the midpoint algorithm. At each sampling position, we select the next pixel along the ellipse path according to the sign of the ellipse function evaluated at the midpoint between the two candidate pixels.

Starting at $(0,r_y)$ we take unit steps in the x direction until we reach the boundary between region 1 and region 2. Then we switch to unit steps in the y direction over the remainder of the curve in the first quadrant. At each step we need to test the value of the slope of the curve. The ellipse slope is calculated from (14) as

$$\frac{dy}{dx} = \frac{2r_y^2 x}{2r_x^2 y} \quad (15)$$

Midpoint Ellipse Algorithm

1. Input r_x, r_y and ellipse center (x_c, y_c), and obtain the first the first point on an ellipse centered on the origin as $(x_0, y_0) = (0, r_y)$.
2. Calculate the initial value of the decision parameter in region 1 as $p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$
3. At each x_k position in region 1, starting at $k = 0$, perform the following test. If p1k < 0, the next point along the ellipse centered on (0,0) is (x_{k+1}, y_k) and

$$p1_{k+1} = p1_x + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the ellipse is (x_k+1, y_k-1) and

$$p1_{k+1} = p1_x + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

with

$$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2$$

$$2r_x^2 y_{k+1} = 2r_x^2 y_k + 2r_x^2$$

and continue until $2r_y^2 x \geq 2r_x^2 y$.

4. Calculate the initial value of decision parameter in region 2 as

$$p2_0 = r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

where (x_0, y_0) is the last position calculated in region 1.

5. At each y_k position in region 2, starting at k =0, perform the following test. If $p2_k > 0$, the next point along the ellipse centered on (0,0) is $(x_k, y_k - 1)$ and

$$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$

Otherwise, the next point along the ellipse is (x_k+1, y_k-1) and

$$p2_{k+1} = p2_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

using the same incremental calculations for x and y as in region 1. Continue until y =0

6. For both regions, determine symmetry points in the other three quadrants.
7. Move each calculated pixel position (x,y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:

$x = x + x_c$ $\quad\quad$ $y = y + y_c$

III. Implementation of Bresenham' Algorithm

This Bresenham Algorithm was used using Visual Studio 2008 language and OpenGL Graphics tool where the computer was CPU Core Duo 1.83GHz and memory was 4 GB.

In this part we will implement Brensenham' Algorithm for

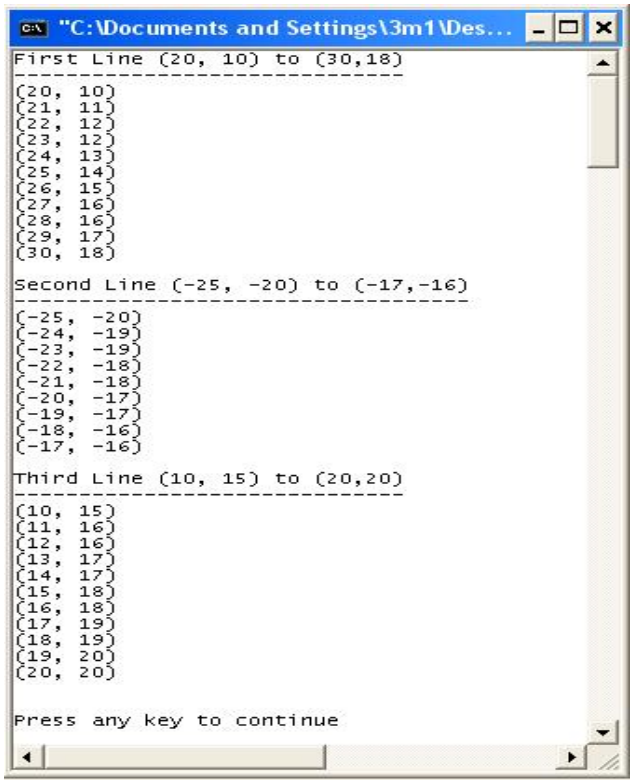
1. Line, with these points <(20, 10), (30, 18)>, <(-17, -16), (-25, -20)>, & <(10, 15), (20, 20)>.
2. Circle, with radius: r = 10, r = 25, & r = 49.
3. Ellipse with parameters: (r_x = 8 and r_y = 6), (r_x = 12 and r_y = 16) & (r_x = 4 and r_y = 6).

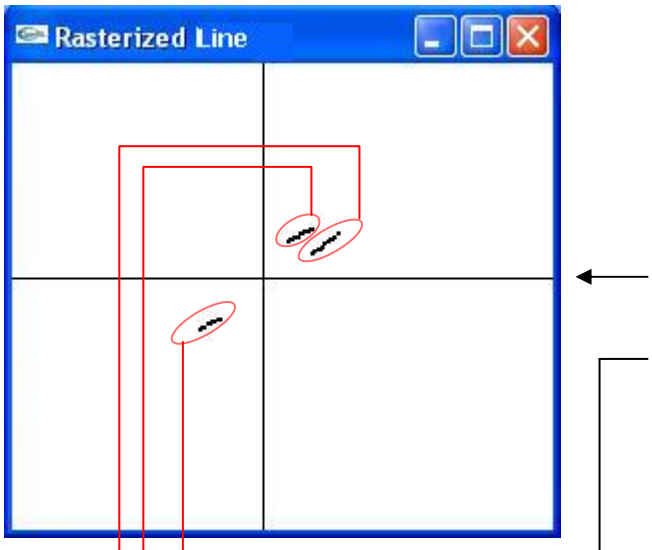
A. Line

Below is the listing program and the result:

Program Modification :

```
#include <stdlib.>
#include <math.h>
#include <windows.h>
#include <gl/Gl.h>
#include <gl/glut.h>


  void lineBres (int x0, int y0, int xEnd, int yEnd)
  {
    int dx = fabs (xEnd - x0),  dy = fabs(yEnd - y0);
    int p = 2 * dy - dx;
    int twoDy = 2 * dy,  twoDyMinusDx = 2 * (dy - dx);
    int x, y;

    if (x0 > xEnd)
            {
      x = xEnd; y = yEnd; xEnd = x0;
    }
    else {
      x = x0;
      y = y0;
    }
    glVertex2i(x, y);

    while (x < xEnd) {
      x++;
      if (p < 0)
        p += twoDy;
      else {
        y++;
        p += twoDyMinusDx;
      }
      glVertex2i  (x, y);
    }
  }

  void myInit(void)
 {
   glClearColor(1.0,1.0,1.0,0.0);       // (1)
   glColor3f(0.0f, 0.0f, 0.0f);         // (2)
    glPointSize(2.0);                   // (3)
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void ellipseMidpoint (int xCenter, int yCenter, int Rx, int Ry);

void myDisplay(void)
{
     glClear(GL_COLOR_BUFFER_BIT);
     glBegin(GL_LINES);
          glVertex2i  (0, 100);
          glVertex2i  (800, 100);
     glEnd();
     glBegin(GL_LINES);
          glVertex2i  (100, 0);
          glVertex2i  (100, 600);
     glEnd();
     glBegin(GL_POINTS);
          lineBres (20+100, 10+100, 30+100, 18+100);
          lineBres (-25+100, -20+100, -17+100, -16+100);
          lineBres (10+100, 15+100, 20+100, 20+100);
     glEnd();
     glFlush();
}

void main(int argc, char** argv)
{
     glutInit(&argc, argv);
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
     glutInitWindowSize(800,600);
     glutInitWindowPosition(100, 50);
     glutCreateWindow("Rasterized Line");
     glutDisplayFunc(myDisplay);
     myInit();
     glutMainLoop();
}
```





The second line is located at the fourth quadrant. So to make it visible. We need to relocate the origin point (0,0) from the bottom left corner to some where at the middle. In this case, we choose (100,100) as a new (0,0) or origin point.

This is tho new X axis. The real value is at Y = 100. Now we consider it as Y = 0.

This is the new Y axis. The real value is at X = 100. Now we consider it as X = 0.

Another modification is everytime the coordinate is generated (set pixel), this modified code will call an OpenGL function (glvertex2i) to draw that points.

Second Line (-25,-20) to (-17,-16)
Third Line (10, 15) to (20, 20)
First Line (20, 10) to (30, 18)

Figure 1. Bresenham Implementation for Line

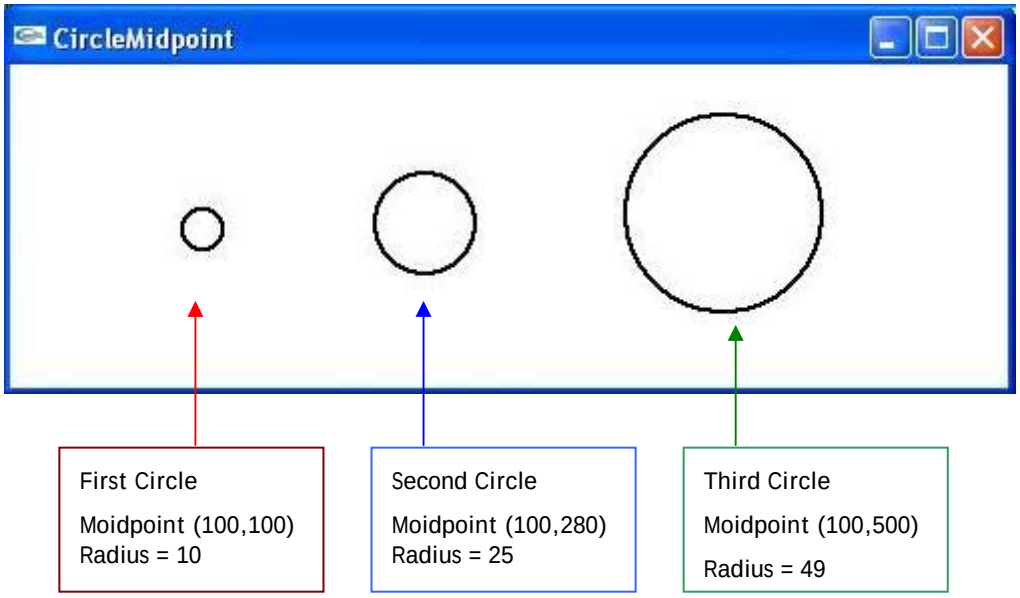
B. Circle

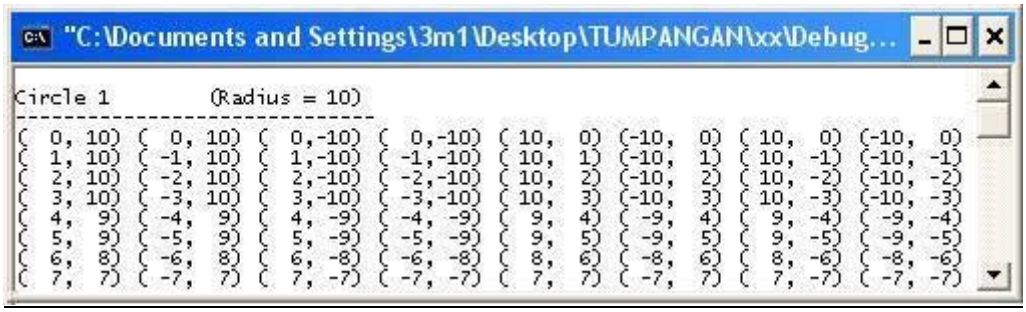
The implementation for Circle, with radius: r = 10, r = 25, & r = 49.

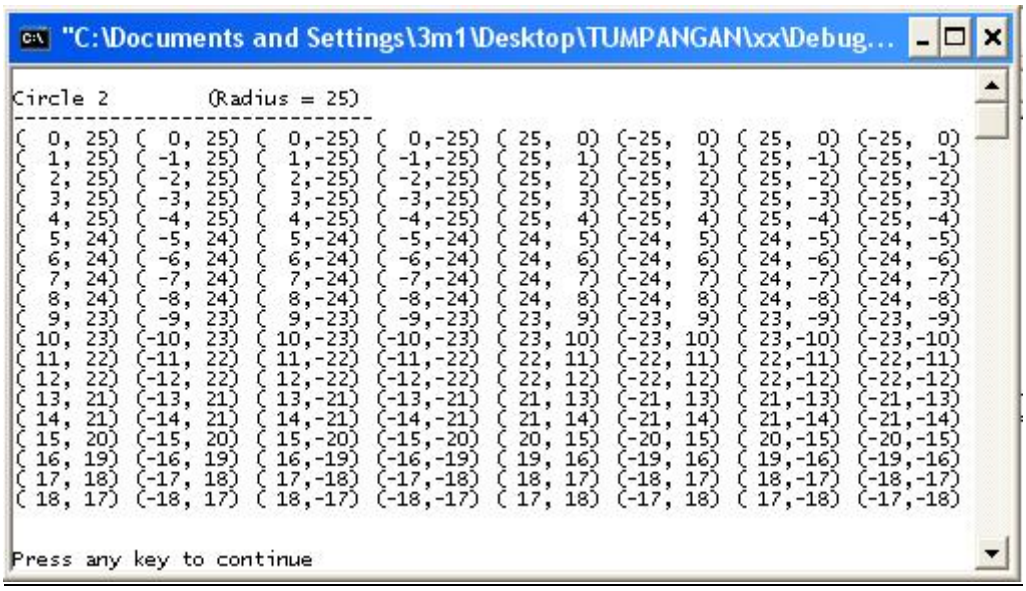
Program Modification :

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
#include <iostream.h>
```

```
struct scrPt
{GLint x, y;};

GLsizei winWidth = 600, winHeight = 500;

void init (void)
{
  glClearColor (1.0, 1.0, 1.0, 1.0); glMatrixMode
(GL_PROJECTION);
  gluOrtho2D (0.0, 200.0, 0.0, 150.0);}

void setPixel (GLint x, GLint y)
{
          glBegin (GL_POINTS);

        glVertex2i (x, y);

          glEnd ( );}

void circleMidpoint (scrPt circCtr, GLint radius)
{
  scrPt circPt;

  GLint p = 1 - radius;

  circPt.x = 0; circPt.y = radius;

  glColor3f (0.0, 0.0, 0.0);
  glPointSize(2);

  void circlePlotPoints (scrPt, scrPt);

  circlePlotPoints (circCtr, circPt);

  while (circPt.x < circPt.y) {
    circPt.x++;
    if (p < 0)
      p += 2 * circPt.x + 1;
    else {
      circPt.y--;
      p += 2 * (circPt.x - circPt.y) + 1;
    }
    circlePlotPoints (circCtr, circPt);}}

void circlePlotPoints (scrPt circCtr, scrPt circPt)
{
  setPixel (circCtr.x + circPt.x, circCtr.y + circPt.y);
  setPixel (circCtr.x - circPt.x, circCtr.y + circPt.y);
  setPixel (circCtr.x + circPt.x, circCtr.y - circPt.y);
  setPixel (circCtr.x - circPt.x, circCtr.y - circPt.y);
  setPixel (circCtr.x + circPt.y, circCtr.y + circPt.x);
  setPixel (circCtr.x - circPt.y, circCtr.y + circPt.x);
  setPixel (circCtr.x + circPt.y, circCtr.y - circPt.x);
  setPixel (circCtr.x - circPt.y, circCtr.y - circPt.x);}

void displayFcn (void)
{
   scrPt circCtr; GLint radius;
        glClear (GL_COLOR_BUFFER_BIT);

   circCtr.x = 100; circCtr.y = 400; radius = 10;
   circleMidpoint (circCtr, radius);
   circCtr.x = 280; circCtr.y = 400; radius = 25;
   circleMidpoint (circCtr, radius);
   circCtr.x = 500; circCtr.y = 400; radius = 49;
   circleMidpoint (circCtr, radius);

        glFlush ( );}

void winReshapeFcn (int newWidth, int newHeight)
{
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ( );
  gluOrtho2D (0.0, (GLdouble) newWidth, 0.0,
(GLdouble) newHeight);

  glClear (GL_COLOR_BUFFER_BIT);}

void main (int argc, char** argv)
{
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (100, 100);
  glutInitWindowSize (winWidth, winHeight);
  glutCreateWindow ("CircleMidpoint");

  init ( );

  glutDisplayFunc (displayFcn); glutReshapeFunc
(winReshapeFcn); glutMainLoop ( );}
```







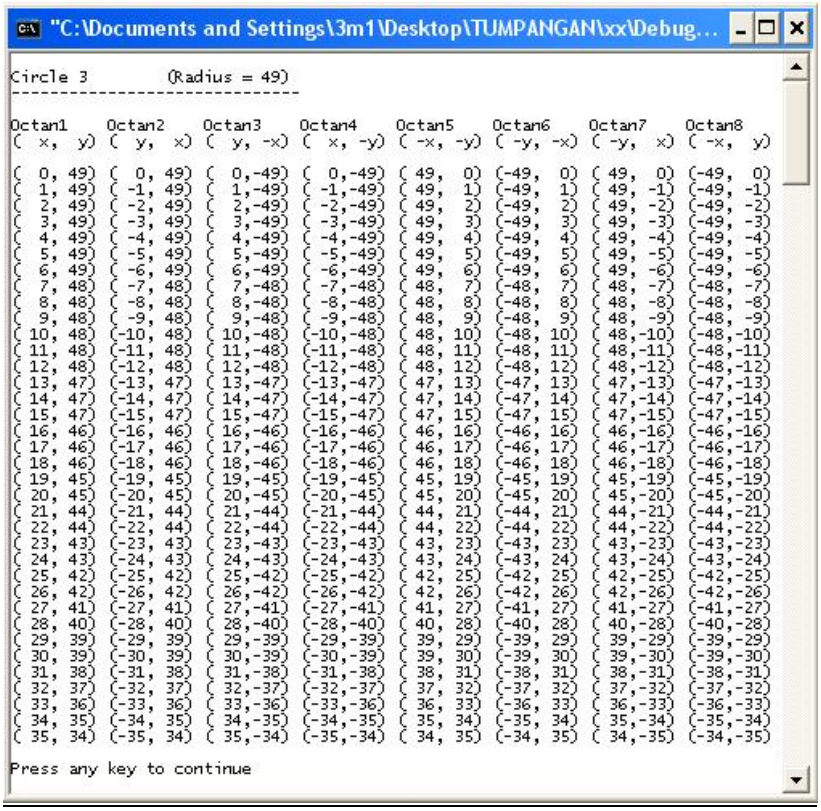


Figure 2. Bresenham Implementation for Cirlce.

C. Ellipse

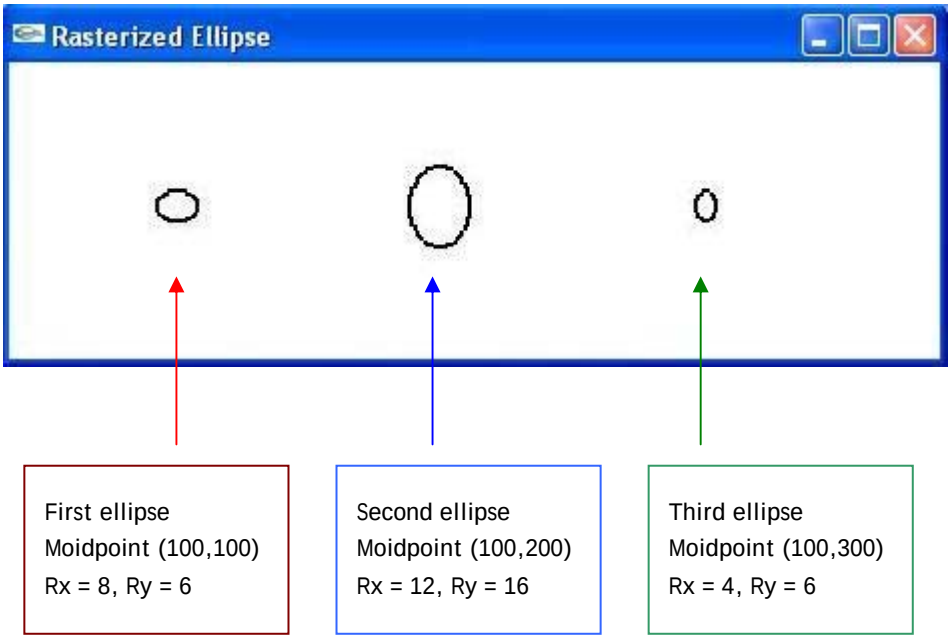
Ellipse with parameters: ($r_x = 8$ and $r_y = 6$), ($r_x = 12$ and $r_y = 16$) & ($r_x = 4$ and $r_y = 6$).

Program Modification :

```
#include <windows.h>
#include <gl/Gl.h>
#include <gl/glut.h>

//<<<<<<<<<<<<<<<<<<<<<<<<
myInit >>>>>>>>>>>>>>>>>>>>>>
 void myInit(void)
 {
    glClearColor(1.0,1.0,1.0,0.0);
    glColor3f(0.0f, 0.0f, 0.0f);
   glPointSize(2.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void ellipseMidpoint (int xCenter, int yCenter,
int Rx, int Ry);

//<<<<<<<<<<<<<<<<<<<<<<<<<<
myDisplay >>>>>>>>>>>>>>>>>
void myDisplay(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POINTS);
      ellipseMidpoint (200, 200, 8, 6);   //
   first ellipse
      ellipseMidpoint (300, 300, 12, 16); //
   second ellipse
      ellipseMidpoint (400, 400, 4, 6);   //
   third ellipse
   glEnd();
   glFlush();
}

//<<<<<<<<<<<<<<<<<<<<<<<<<<
main >>>>>>>>>>>>>>>>>>>>>>>
void main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640,580);
   glutInitWindowPosition(100, 50);
   glutCreateWindow("Raster Ellipse");
   glutDisplayFunc(myDisplay);
   myInit();
   glutMainLoop();
}

inline int round (const float a)  { return int
(a + 0.5); }

void ellipseMidpoint (int xCenter, int yCenter,
int Rx, int Ry)
{
  int Rx2 = Rx * Rx;    int Ry2 = Ry * Ry;
  int twoRx2 = 2 * Rx2; int twoRy2 = 2 * Ry2;
  int p;       int x = 0;
  int y = Ry;          int px = 0;
  int py = twoRx2 * y;
  void ellipsePlotPoints (int, int, int, int);

   ellipsePlotPoints (xCenter, yCenter, x, y);

  /* Region 1 */
  p = round (Ry2 - (Rx2 * Ry) + (0.25 * Rx2));
  while (px < py) {
     x++;
     px += twoRy2;
     if (p < 0)
        p += Ry2 + px;
     else {
        y--;
        py -= twoRx2;
        p += Ry2 + px - py;
     }
     ellipsePlotPoints (xCenter, yCenter, x, y);
  }
```

```
  /* Region 2 */
  p = round (Ry2 * (x+0.5) * (x+0.5) + Rx2 * (y-
1) * (y-1) - Rx2 * Ry2);
  while (y > 0) {
     y--;   py -= twoRx2;  if (p > 0)  {p += Rx2
- py;}
     else {
        x++;   px += twoRy2;  p += Rx2 - py +
px;}

     ellipsePlotPoints (xCenter, yCenter, x, y);
}
```

```
  }
}

void ellipsePlotPoints (int xCenter, int yCenter,
int x, int y)
{
  glVertex2i (xCenter + x, yCenter + y);
  glVertex2i (xCenter - x, yCenter + y);
  glVertex2i (xCenter + x, yCenter - y);
  glVertex2i (xCenter - x, yCenter - y);
```



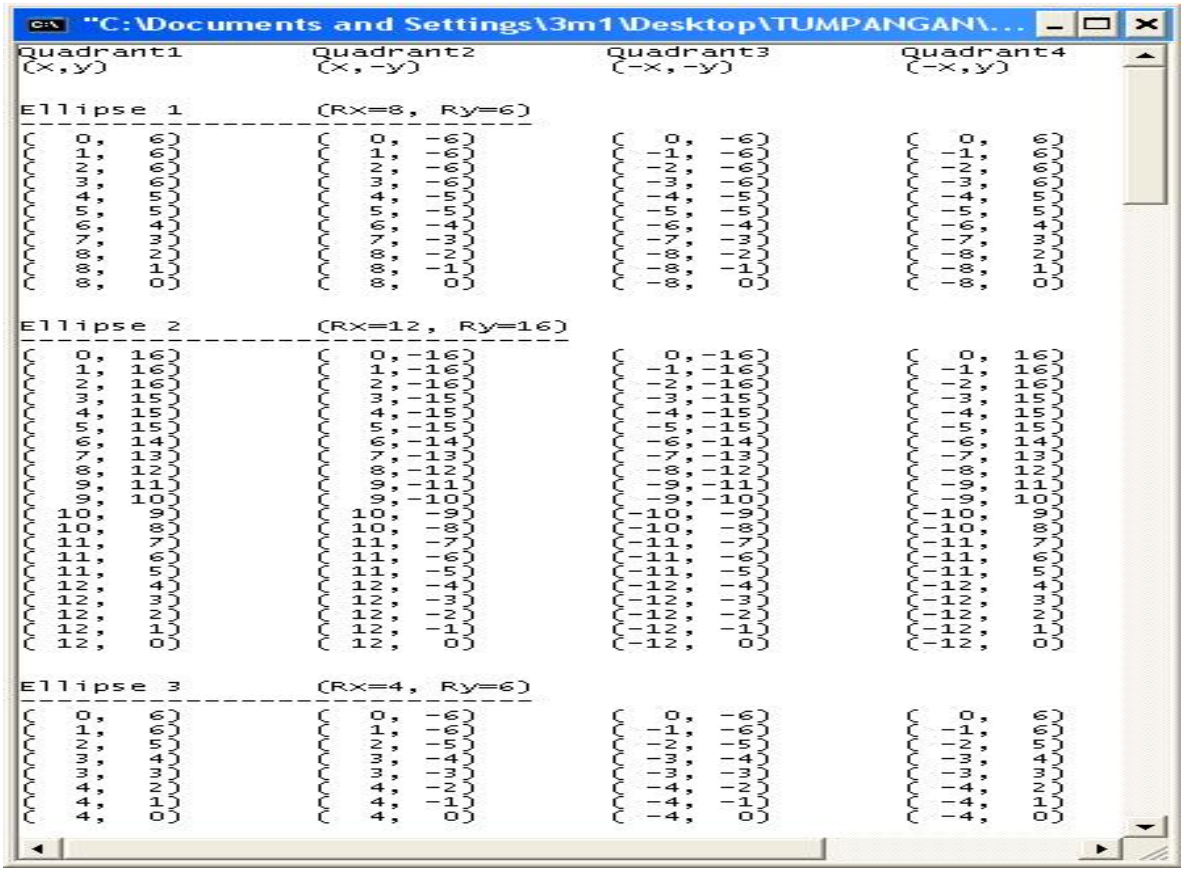


Figure 3. Bresenham Implementation for Ellipse

IV. Analysis of the Algorithm Performance

To test the efficiency of Bresenham Algorihm, different lines, circles and ellipse in [0,1] and lengths are designed. The testing programs are optimally designed using Visual Studio 2008 for comparison and analysis for Bresenham algorithm. To get higher precision of comparison, the two algorithms have run 10 thousand times separately, part of the experimental results are shown in Chart below

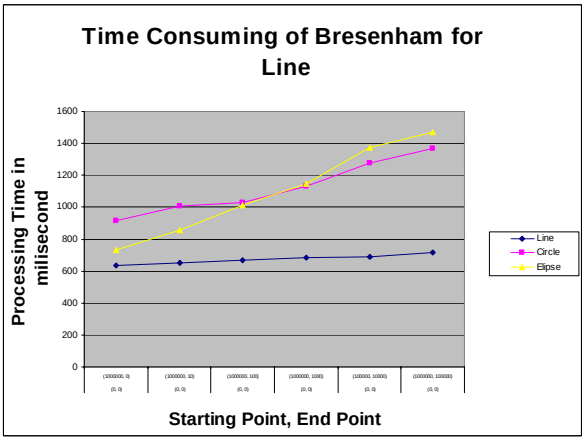


Chart 1. The speed comparison of algorithm of pixel line and Bresenham algorithm (millisecond)

V. Conclusion

In terms of accuracy the Bresenham algorithm produce a fine result in raster environment. The only variable that should be count only at the data type that used. This solution can easily be implemented in micro controllers and other processors in which there is no floating point unit. The graphs that produced are fine: example:

References

[1] Ammeraal, L., 1987. Computer Graphics for IBM PC. John Wiley & Sons, New York.
[2] Bresenham. J,. 1971. Algorithm for computer control of a digital plotter. IBM Systems Journal, 4(1):25–30, 1965.
[3] Bresenham. J,. 1977. A linear algorithm for incremental digital display of circular arcs. *Communications of ACM*, 20(2):100-106.
[4] Donald, E., 1986. METAFONT the Program. Addison-Wesley, Reading, Massachusetts.
[5] Foley, J., Dam, A., Feiner, S., Hughes, J., 1990.Computer Graphics: Principles and Practice. Addison-Wesley, Reading, Massachusetts.
[6] Foley, J., Dam, A., Feiner, S., Hughes, J., 1993. Introduction to: Computer Graphics. Addison-Wesley, Reading, Massachusetts.
[7] Klassen, R., 1991. Integer forward differencing of cubic polynomials: Analysis and algorithms. *ACM Transactions on Graphics*, **10**(2):152-181.
[8] Pitteway, M., 1985. Algorithms of Conic Generation. Fundamental Algorithms for Computer Graphics, NATO ASI Series, p.219-237.
[9] Rappoport, A., 1991. Rendering curves and surfaces with hybrid subdivision and forward differencing. *ACM Transactions on Graphcis*, 10(4):323-341.
[10] Andrea, F., G. Andrea and M. Giuseppe., 2010., Rock Slopes Failure Susceptibility Analysis: From Remote Sensing Measurements to Geographic Information System Raster Modules., American Journa of Environmental Sciences 6(6) . Pp 489-494 DOI: 10.3844/ajessp.2010.489.494.
[11] Donald Hearn, and M. Pauline Baker, Computer graphics, C Version, Second edition, Prentice-Hall (2005).
[12] R.M.Mersereau, "The processing of hexagonally sampled two dimensional signals," Proc. IEEE vol. 67 no. 6 pp.930-949, June 1979.
[13] R.C.Staunton, "Hexagonal image sampling, a practical proposition," Proc. SPIE vol. 1008 pp. 23-27, 1989. Luszak.E and Rosenfeld.A, Distance on a hexagonal grid, IEEE Transactions on Computers, 25(3), 532-533(1968).
[14] Wuthrich, C.A. and Stucki, P, An algorithm comparison between square- and hexagonal based grids. Graphical Models and Image Processing, 53(4), 324–339. (1991).
[15] Bimal kumar Ray, An alternative approach to circle drawing, Journal.Indian Institude of Science, 86,617-623(2006).
[16] M.Prabukumar. Article: Line Drawing Algorithm on an Interleaved Grid. International Journal of Computer Applications, 2007., 19(4):1-7,
[17] LIU Shi-jun, DENG Bei-sheng, XUN Huai-gan. Four-step Scan-symmetrical IncrementalGeneration of Line. Journal of Image and Graphic.2002,7A(10):1054-1057[4]
[18] Lin Li, Rong CHE. Bresenham-based 4-point line drawing algorithms. Journal of JinanUniversity (Natural Science),2003,24 (5):19-21[5]
[19] Asiful Haque, Mohammad Saifur Rahman, Mehedi Bakht. Drawing lines by uniform packing.Computers & Graphics,2006,30(2):270-212[6]