Implementation of Circle and Ellipse Drawing Algorithms Using OpenGL

Tarun Kumar
UE143103
Computer Science Engineering
University Institute of Engineering and Technology
Panjab University
Chandigarh, 160030
Email: tarunk.1996@gmail.com

***Abstract*—Building an OpenGL application by implementing various Circle drawing and Ellipse drawing algorithms such as : Bresenham and Mid-point algorithm. Also, providing option to choose among different properties of curve e.g patterns, width and color of circle or ellipse. FreeGlut library is used in development of this application.**

I. Circle Drawing Algorithms

The objective is to draw a desired figure on a digital display machine using some algorithm (fixed number of pixels). In order to achieve the goal, some set of pixel in the screen are set to ON based on the Algorithm which is being implemented. In these algorithms, some decisions are made i.e. to glow a set of pixels such that the resultant figure obtained is nearly similar to original figure. As the pixel density of the screen increases perfection in the figure increases (deformity of the curve decreases).

The two basic Circle Drawing Algorithms which have been used to generate circles are:

- Bresenhams Algorithm
- Mid-Point Algorithm

A. Bresenhams Circle Drawing Algorithm

Algorithm

- Step 1: Get the first input from mouse as center of circle (x_1, y_1) and second input (x_2, y_2) as the distance from center.
- Step 2: Calculate the radius of the circle:

$$Radius = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Step 3: Initialize the variables

$$X = 0, Y = Radius$$

$$D = 3 - 2 * Radius$$

- Step 4: If $(D >= 0)$
  - $Y = Y - 1$
  - $D = D + 10 + 4 * (X - Y)$
- Step 5: Else
  - $D = D + (4 * X) + 6$
- Step 6: Repeat step 4 until $X <= Y$ and Increment X by 1 each time.
- Plot the pixel to display the circle using put pixel function. (Remember to print the pixel in all the 8 direction from the circle).

B. Mid-Point Circle Drawing Algorithm

Algorithm

- Step 1: Get the first input from mouse as center of circle (x_1, y_1) and second input (x_2, y_2) as the distance from center.
- Step 2: Calculate the radius of the circle:

$$Radius = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Step 3: Initialize the variables

$$X = 0, Y = Radius$$

$$P = 1 - Radius$$

- Step 4: If $(P < 0)$ then
  - The next point along the circle is $(X + 1, Y)$.
  - $P = P + 2 * X + 1$
- Step 5: Else
  - The next point along the circle is $(X + 1, Y - 1)$.
  - $P = P + 2 * (X - Y) + 1$
- Step 6: Repeat step 4 until $X < Y$.
- Plot the pixel to display the circle using putPixel function. (Remember to print the pixel in all the 8 direction from the circle).

II. Ellipse Drawing Algorithm

A. Mid-Point Ellipse Drawing Algorithm

Algorithm

- Step 1: Get Input R_x, R_y and Ellipse center (X_c, Y_c) from the user, Obtain the first point on an ellipse centered at origin as $(X_0, Y_0) = (0, R_y)$.
- Step 2: Calculate the initial value of the decision parameter in region 1 as

$$P1_0 = R_y^2 - R_x^2 * R_y + (1/4)R_x^2$$

- Step 3: At each X_k position in region 1, Starting at k=0, perform the following test:
  - If $P1_k < 0$, then the next point along the ellipse centered on (0,0) is $(X_k + 1, Y_k)$ and
    $$P1_{k+1} = P1_k + 2 * R_y^2 * X_{k+1} + R_y^2$$
  - Otherwise, the next point along the ellipse is $(X_k + 1, Y_k - 1)$ and
    $$P1_{k+1} = P1_k + 2 * R_y^2 * X_k + 1 - 2 * R_x^2 * Y_{k+1} + R_y^2$$
- Step 4: Repeat the steps for region 1 until
  $$2 * R_y^2 * x >= 2 * R_x^2 * y$$
  .
- Step 5: Calculate the initial value of the decision parameter in region 2 using the last point (X_0, Y_0) calculated in region 1 as
  $$P2_0 = R_y^2(X_0 + (1/2))^2 + R_x^2(Y_0 - 1)^2 - R_x^2 * R_y^2$$
- Step 6: At each Y_k position in region 2, starting at k=0,perform the following test:
  - If $P2_k > 0$, then the next point along the ellipse centered on (0,0) is $(X_k, Y_k - 1)$ and
    $$P2_{k+1} = P2_k - 2 * R_x^2 * Y_{k+1} + R_x^2$$
  - Otherwise, the next point along the ellipse is $(X_k + 1, Y_k - 1)$ and
    $$P2_{k+1} = P2_k + 2 * R_y^2 * X_k + 1 - 2 * R_x^2 * Y_{k+1} + R_y^2$$
- Step 7: Repeat step 6 until $Y > 0$.
- Plot the pixel to display the circle using putPixel function. (Remember to print the pixel in all the 4 direction from the Ellipse).

III. About FreeGlut

FreeGLUT is a free-software/open-source alternative to the OpenGL Utility Toolkit (GLUT) library. GLUT was originally written by Mark Kilgard to support the sample programs in the second edition OpenGL 'RedBook'. Since then, GLUT has been used in a wide variety of practical applications because it is simple, widely available and highly portable.

GLUT (and hence FreeGLUT) takes care of all the system-specific chores required for creating windows, initializing OpenGL contexts, and handling input events, to allow for trully portable OpenGL programs.

FreeGLUT is released under the X-Consortium license.

IV. Some FreeGlut Functions

Following are functions used in developing the application with brief intro about it.

- **Creating A Window**
  The glutInitWindowPosition and glutInitWindowSize functions specify a desired position and size for windows that freeglut will create in the system. This glutCreateWindow("Line Drawing Algorithm") will create a Glut Window with a Title Line Drawing Algorithm on it.
- **Setting Background Color**
  This will set the background color of the screen to white.
```
glClearColor(1, 1, 1, 0);
glClear(GL_COLOR_BUFFER_BIT);
glFlush();
```
- Plotting X and Y axis in the window. Set all the pixel along X-axis to black color and similar to that in Y-axis. (Remember according to FreeGlut co-ordinate system (0,0) will lie at the top-left corner of the window, So it should be shifted to the center of the screen first). In order to create X and Y - axis, you can either use inbuilt line drawing function or can set each pixel along that line to color black. Both will gives the same result.
- **Showing Text**
  To display some text at any co-ordinate in the OpenGL window. It can be done by using the following code:
```
drawStrokeText(*string, x, y, z)
{
 char *c;
 glPushMatrix();
 glTranslatef(x, y+8,z);
 glScalef(0.09f,0.08f,z);
 for (c=string; *c != '\0'; c++)
 {
  glutStrokeCharacter(GLUT_STROKE_ROMAN,*c);
 }
 glPopMatrix();
}
```
  In the above function drawStrokeText accepts the string, x, y and z co-ordinates. It will print the string as in character by character format. $GLUT_STROKE_ROMAN$ is the text format that will be displayed on the screen.
- **Creating a Button**
  FreeGlut library as such doesnt include any provision for creating a clickable buttons and associating an action when a button is clicked. So in order to create a button, Create a button look alike box which will perform some action whenever there will a left mouse click on it. Following code will create a box in the glut screen of 80x25 size with a background color of red. GL_QUADS is used to create a quadrilateral in the OpenGL window.
```
glBegin(GL_QUADS);
glColor3f(1,0, 0);
glVertex2i(0,0);
glVertex2i(80,0);
glVertex2i(0,25);
glVertex2i(80,25);
glEnd();
drawStrokeText("Button",5,0,0);
```
  Now, to make this box a fully functional button. Mouse Left click event must be introduced in it.
```
mouse(int btn, int state, int x, int y)
  if(btn == GLUT_LEFT_BUTTON
&& state == GLUT_DOWN)
//Do some event
```

OUTPUTS :

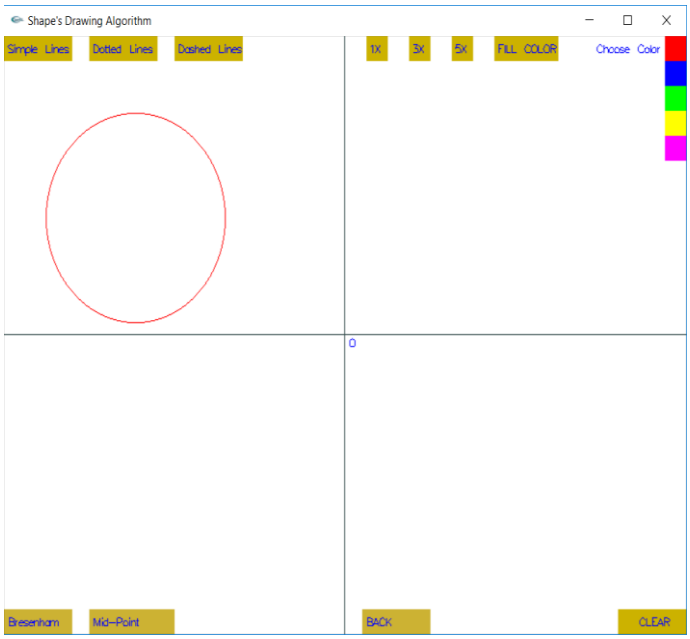

Fig 1: Circle
Default Circle properties
Color: Red
Type: Simple
Algorithm: Bresenham
Thickness: 1X

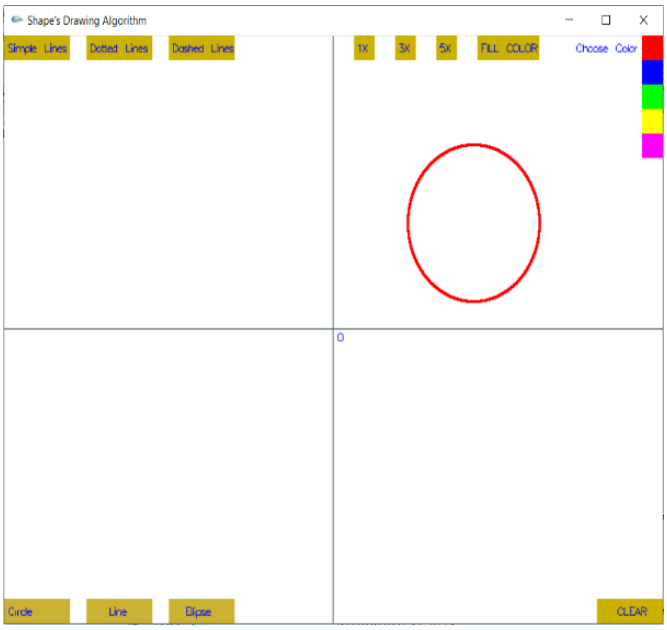

Fig 2: Circle
Circle properties
Color: Red
Type: Simple
Algorithm: Mid-Point
Thickness: 3X

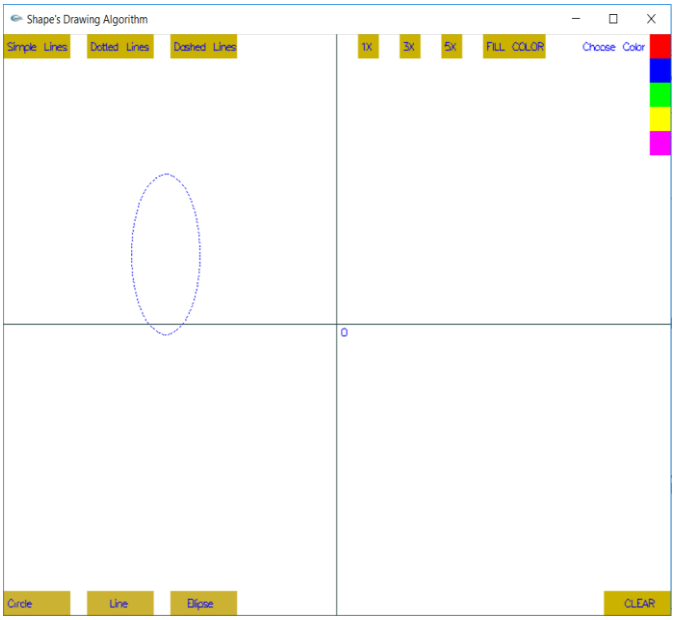

Fig 3: Dashed Ellipse
Ellipse properties
Color: Blue
Type: Dashed
Algorithm: Mid-Point
Thickness: 1X

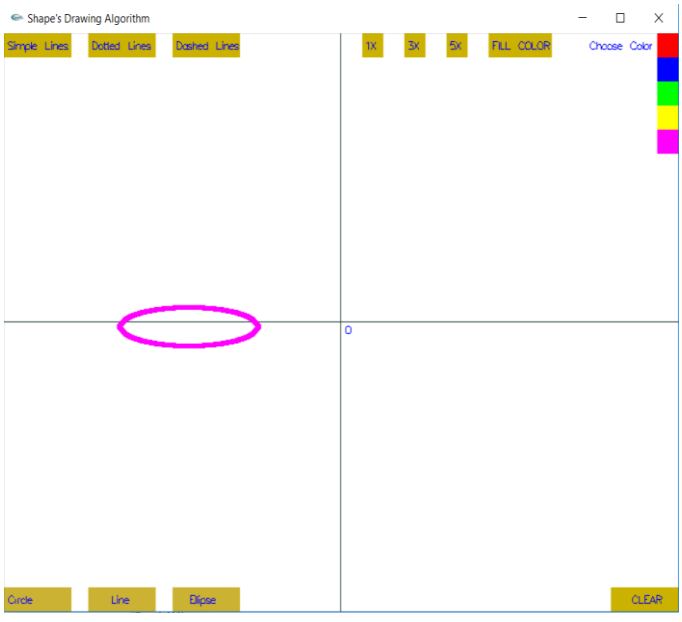

Fig 4: Ellipse
Ellipse properties
Color: Pink
Type: Simple
Algorithm: Mid-Point
Thickness: 5X