Implementation of Line 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 line drawing algorithms such as : Bresenham, Simple DDA, Symmetric DDA and Mid-point algorithm. Also, providing option to choose among different properties of line e.g patterns, line width and color of line. FreeGlut library is used in development of this application.**

I. Line Generation Algorithms

The basic objective of Line generation on a digital display (fixed number of pixels) is to plot a line between two points which is nearly same as the actual straight line. In order to achieve the goal, some set of pixel in the screen are set to ON based on Line generating algorithms. In these algorithms, some decisions are made i.e. to glow the pixel which is near to the original line. The resultant line generated is similar to that of original line. As the pixel density of the screen increases the line tends to move towards perfect line (though perfect line is impossible to generate).

The three basic Line Drawing Algorithms which have been used to generate lines are:

- Digital Differential Analyser (DDA) Algorithm
- Bresenhams Line Generation Algorithm
- Mid-Point Algorithm

A. Digital Differential Analyzer (DDA) Algorithm

Algorithm

- Step 1: Get the input of two end points (x0,y0) and (x1,y1) from the user.
- Step 2: Calculate the difference between two end points as:

```
dx = x1 - x0
dy = y1 - y0
```

- Step 3: Based on the calculated difference in Step 2, the total number of decisions to put pixels is determined. If $dx > dy$, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (dx > dy)
Steps = absolute(dx);
else
Steps = absolute(dy);
```

- Step 4: The increment in x coordinate and y coordinate is calculated as:

```
Xinc = dx / (float) steps;
Yinc = dy / (float) steps;
```

- Step 5 The pixel is put by successfully incrementing x and y coordinates accordingly and then completing the drawing of the line.

```
for(int i=0; i < Steps; i++)
{
   x = x + Xinc ;
   y = y + Yinc ;
   putPixel(x, y);
}
```

B. Bresenhams Line Generation Algorithm

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.

Algorithm

- Step 1: Input the two end-points of line in $(x0, y0)$ and $(x1, y1)$.
- Step 2: Plot the point $(x0, y0)$.
- Step 3 Calculate the constants $dx, dy, 2dy$, and $(2dy2dx)$ and get the first value for the decision parameter as

$$P0 = 2dydx$$

- Step 4: At each Xk along the line, starting at $k = 0$, perform the following test
  If $Pk < 0$, the next point to plot is $(xk + 1, yk)$ and

$$Pk + 1 = Pk + 2dy$$

  Otherwise,

$$Pk + 1 = Pk + 2dy2dx$$

- Step 5: Repeat step 4 $(dx1)$ times.

C. Mid-Point Algorithm

Mid-Point Algorithm is a modified version of Bresenhams Algorithm. For a line with slope $m < 1$, in this algorithm, mid-point M of $(x+1, y)$ and $(x+1, y+1)$ is calculated which can be designated as $(x+1, y+1/2)$. If the line to be drawn passes above M then point $(x+1, y+1)$ is chosen as the next point otherwise $(x+1, y)$ is chosen as the next point.

Mathematical explanation for this algorithm being quite lengthy is not included here either.

II. 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.

III. 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. Here, btn will return the action performed by mouse, it can be mouse movement, position, left click, right click or scroll action. State will give the current mouse state of the mouse also x and y will have the current mouse location or co-ordinates.

```
mouse(int btn, int state, int x, int y)
  if(btn == GLUT_LEFT_BUTTON
&& state == GLUT_DOWN)
//Do some event as per specified
```

- **Creating Patterned Lines**
  Create some buttons allowing user to select among the options provided. As per the user selection select different algorithm in order to print desired line.
  - *Dotted Lines:* Set the pixel to ON firstly and after that keep altering the next pixel to either ON or OFF based on previous pixel decision. [**Note:** Generate the dotted line from both ends till mid point. This will eliminate the possibility of leaving the end pixel blank]
  - *Dashed Lines:* Set the first 3 pixel ON and next 2 OFF and keep on repeating this until end. Draw the line from both end to mid-point. To generate a variable dashed lines alter the number of ON and OFF pixel.
  - *Thick Lines:* Draw a simple line first and then draw another line above and below it as per specified by user. It will increase the thickness of the line.

OUTPUTS :

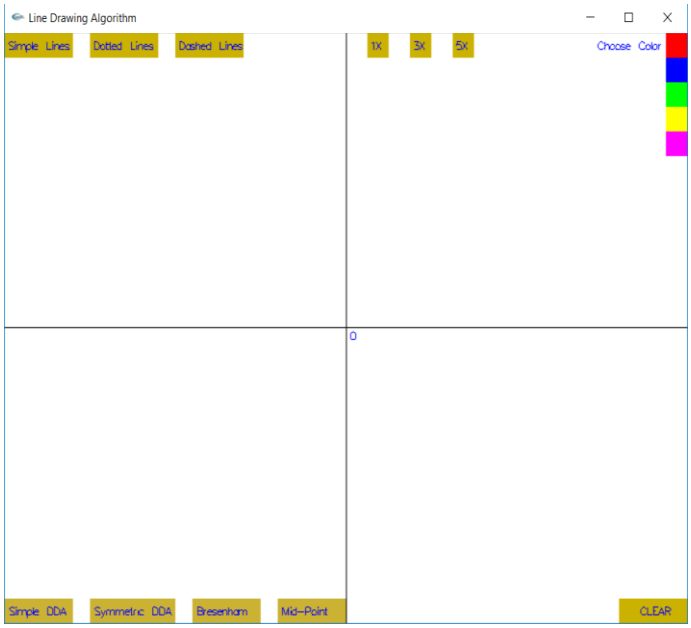

Fig 1: Opening Window
Default line properties
Color: Red
Type: Simple Line
Algorithm: Bresenham
Thickness: 1X

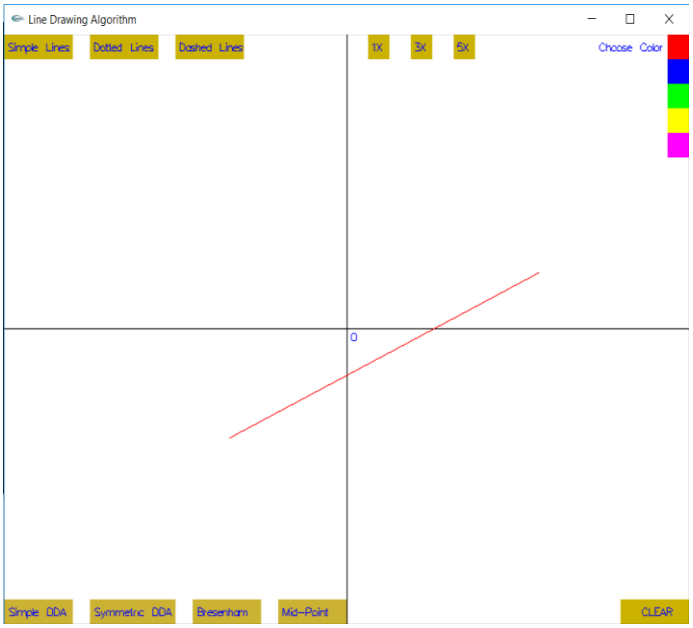

Fig 2: A Simple Line
Line properties
Color: Red
Type: Simple Line
Algorithm: Bresenham
Thickness: 1X

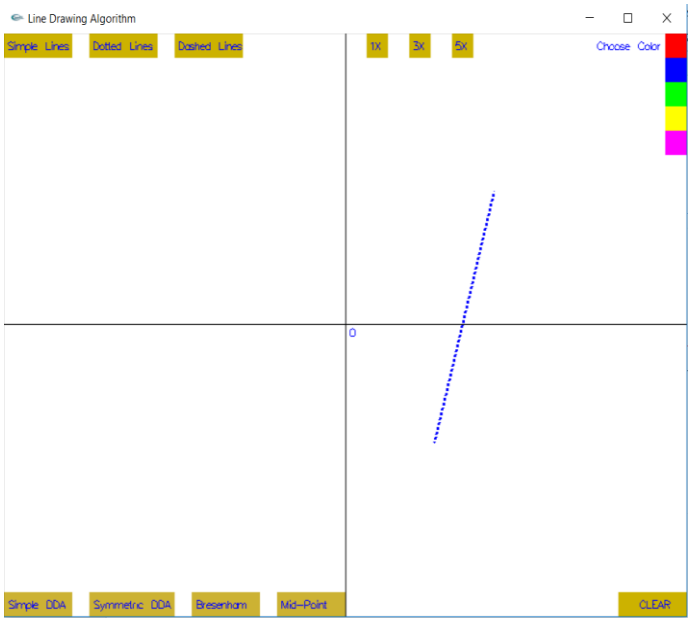

Fig 3: Dashed Line
Line properties
Color: Blue
Type: Dashed
Algorithm: Simple DDA
Thickness: 3X

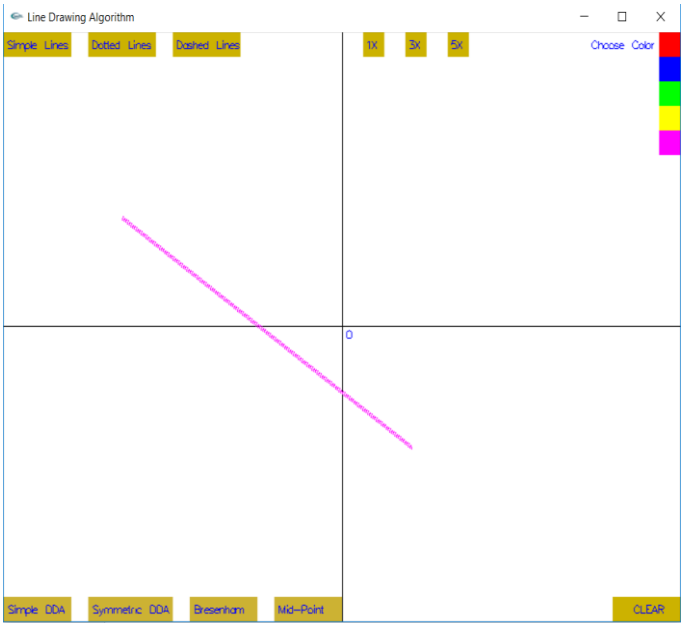

Fig 4: Dotted Line
Line properties
Color: Pink
Type: Dotted
Algorithm: Simple DDA
Thickness: 5X