Implementation of Color Filling 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 to draw various 2D shapes using lines, circles and ellipses. Provide an option to user, which lets user to fill a closed figure with some color. FreeGlut library is used in development of this application.**

I. Color Filling Algorithms

The objective is to fill a closed figure with some desired color. 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 Color Filling Algorithms which have been used are:

- Boundary Fill Algorithm
- Flood Fill Algorithm

A. Boundary Fill Algorithm

Start at a point inside the figure and paint with a particular color. Filling continues until a boundary color is encountered.

There are 2 ways to achieve it:

- 4-connected
- 8-connected

Algorithm 4-connected

- Step 1: Get the value of seed pixel (X, Y), fillColor and borderColor.
- Step 2: Get the color of current pixel (X, Y).
- Step 3: Check if the current pixel color is equal to borderColor. If not then return.
- Step 4: Set current pixel with fillColor

$$setPixel(X, Y, fillColor)$$

- Step 5: Recursively call the function with 4-neighbor (adjacent) pixels

```
boundaryfill(x+1,y,borderColor,fillColor)
boundaryfill(x-1,y,borderColor,fillColor)
boundaryfill(x,y+1,borderColor,fillColor)
boundaryfill(x,y-1,borderColor,fillColor)
```

Algorithm 8-connected

- Step 1: Get the value of seed pixel (X, Y), fillColor and borderColor.
- Step 2: Get the color of current pixel (X, Y).
- Step 3: Check if the current pixel color is equal to borderColor. If not then return.
- Step 4: Set current pixel with fillColor

$$setPixel(X, Y, fillColor)$$

- Step 5: Recursively call the function with 8-neighbor pixels

```
Bfill(x+1,y,borderColor,fillColor)
Bfill(x-1,y,borderColor,fillColor)
Bfill(x,y+1,borderColor,fillColor)
Bfill(x,y-1,borderColor,fillColor)
Bfill(x+1,y+1,borderColor,fillColor)
Bfill(x-1,y-1,borderColor,fillColor)
Bfill(x-1,y+1,borderColor,fillColor)
Bfill(x+1,y-1,borderColor,fillColor)
```

B. Flood Fill Algorithm

When filling an area that is not defined within a single color boundary we use Flood fill. It will keep coloring the adjacent pixel with some color value until it encounter the same interior color value of the seed pixel.

Algorithm 4-connected

- Step 1: Get the value of seed pixel (X, Y) and fillColor.
- Step 2: Get the color of current pixel (X, Y).
- Step 3: Check if the current pixel color is equal to interior color. If not then return.
- Step 4: Set current pixel with fillColor

$$setPixel(X, Y, fillColor)$$

- Step 5: Recursively call the function with 4-neighbor (adjacent) pixels

```
floodfill(x+1,y,fillColor)
floodfill(x-1,y,fillColor)
floodfill(x,y+1,fillColor)
floodfill(x,y-1,fillColor)
```

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.
  ```
  mouse(int btn, int state, int x, int y)
    if(btn == GLUT_LEFT_BUTTON
  && state == GLUT_DOWN)
  //Do some event
  ```
- **OpenGL Color Functions** To set the color display mode glut initialize display mode is used :
  ```
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB)
  ```
  Setting the color of a pixel can be done by 3 methods: Floating point values
  ```
  glColor3f(0.0,1.0,0.0)
  ```
  An array specification
  ```
  glColor3fv(colorArray)
  ```
  Integer Values
  ```
  glColor3i(0,255,0)
  ```
- **Get properties of certain pixel** To read the seed pixel and extract the properties out of it is can be done by using $glReadPixels()$ function. Syntax of $glReadPixels()$
  ```
  glReadPixels(X,Y,W,H,format,type,data);
  ```
  Here, X, Y specifies the window coordinates of the first pixel that is read from the frame buffer. W, H specifies the dimensions of the pixel rectangle, width and height of one correspond to a single pixel. $format$ specifies the format of the pixel data. $type$ specifies the data type of the pixel data. $data$ returns the pixel data(can be used to return the color of the seed pixel).

OUTPUTS :

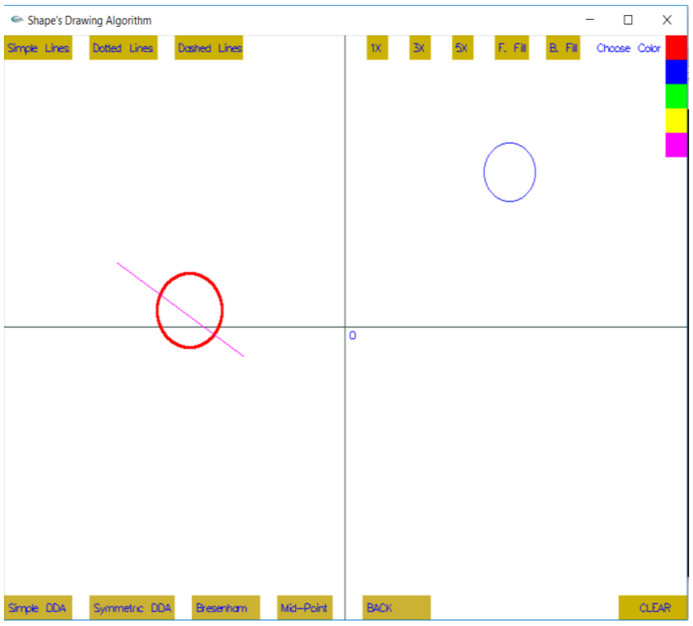

Fig 1: Sample Figure

Used for color filling demonstration in figure 2.

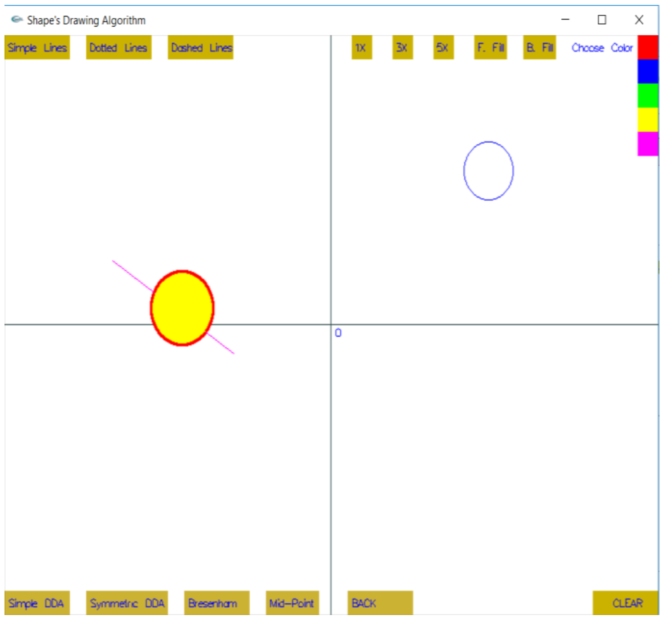

Fig 2: Boundary Filling
Color Filling properties
Color: Yellow
Algorithm: Boundary Fill
Figure 1 used as source.

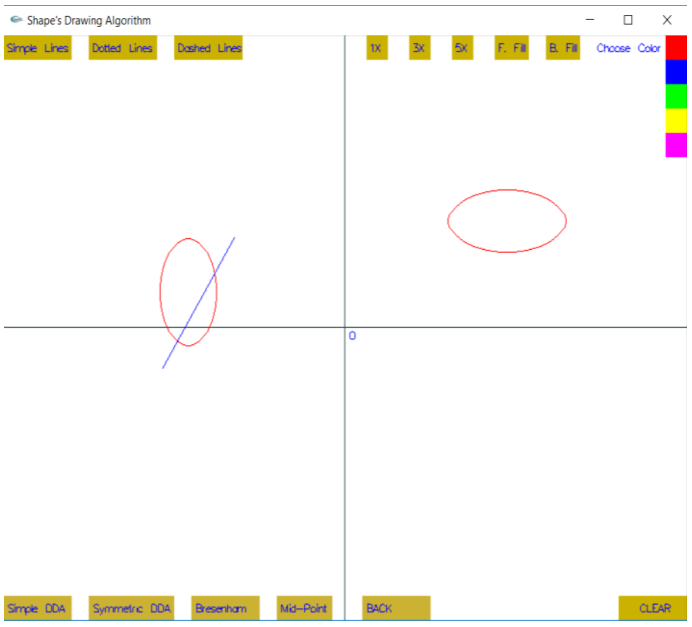

Fig 3: Sample Figure

Used for color filling demonstration in figure 4.

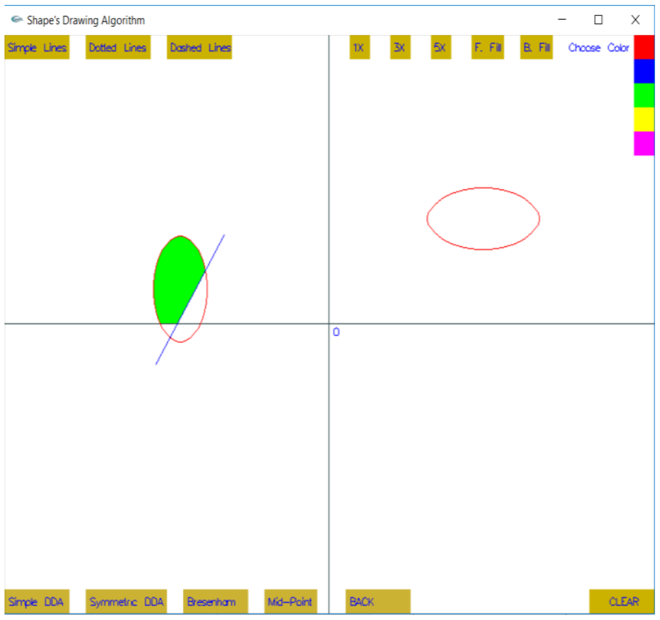

Fig 2: Flood Filling
Color Filling properties
Color: Green
Algorithm: Flood Fill
Figure 3 used as source.