<u>**University of Florida**</u>

Computer Graphics - CAP 5705

Implementation of Graphics Editor using OpenGL

PROJECT REPORT

Submitted by

PRIYANSHU PANDEY, UFID**:** 21081358
SOURAV PARMAR, UFID**:** 86511933

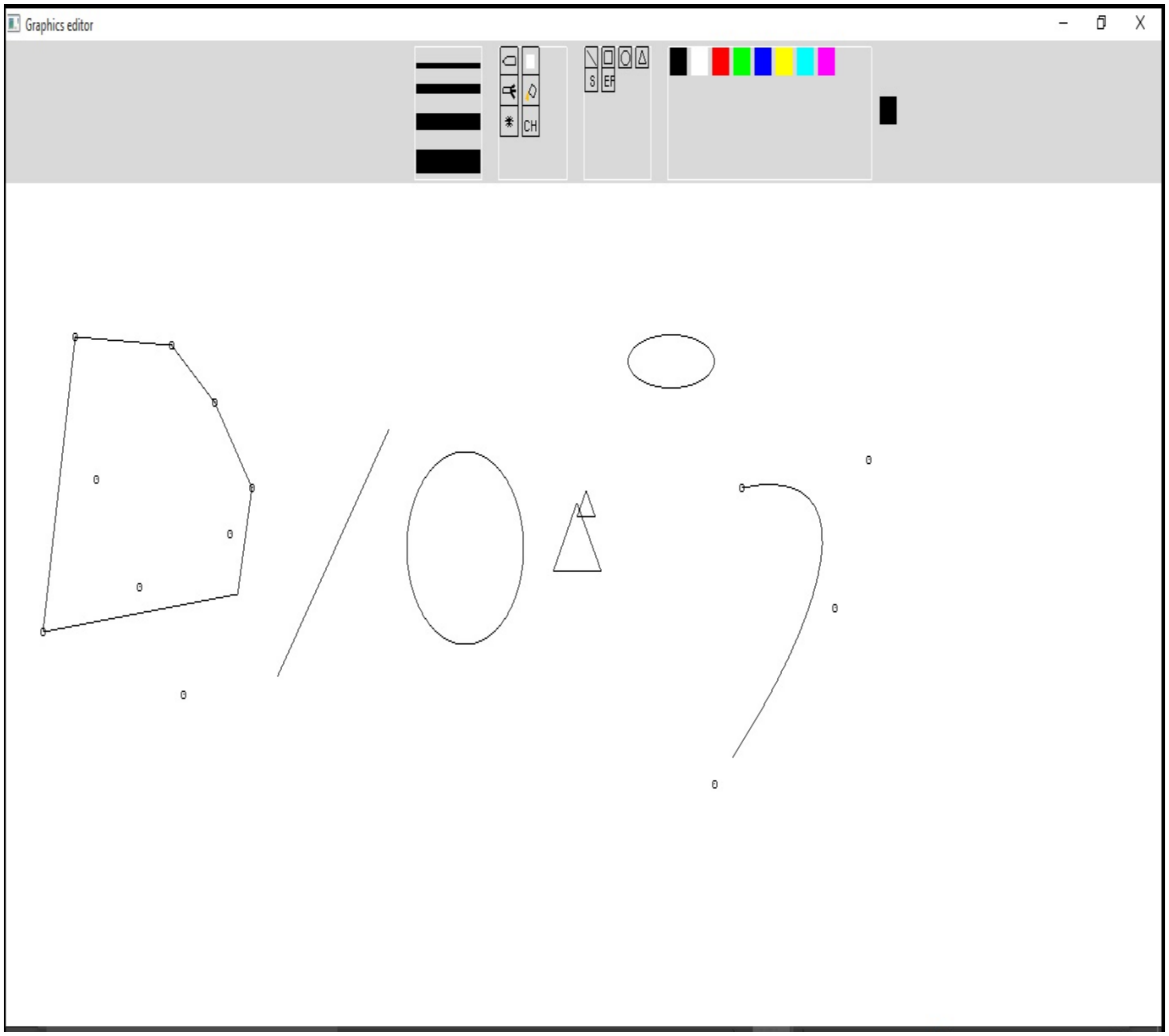


fig1: Graphics Editor

1. Motivation:

"**Graphics Editor**" implemented in this project is a paint application which gives various functionalities to the user to draw, color and to choose from multiple predefined shapes to draw on canvas. Majorly the following functionalities are implemented in this project.
Pencil, Brush, Draw lines, Draw polygons (line, rectangle, triangles etc), Draw circle and ellipse. Draw curve, Spray, Color fill, Convex Hull, Erase.

2. Tool Used:

- GLUT is the OpenGL Utility Toolkit used for writing OpenGL programs.
    - It implements a simple windowing application programming interface (API) for OpenGL.
    - GLUT provides a portable API so you can write a single OpenGL program that works across all PC.
- The code is written in C++.

3. Implementation:

3.1 User Interface:

The working window is designed to have icons on the top from which the user can choose the action he wants to perform.

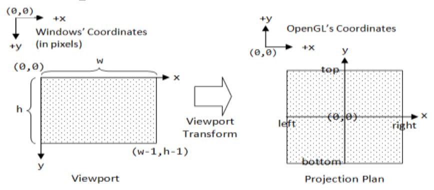


Fig 2 : OpenGL Coordinates [2]

- **Toolbox :** It allows the user to choose between one of the following options for drawing: Pencil, Brush, Spray, Eraser, Fill Color, Convex-Hull.

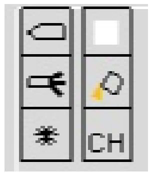


Fig3. Toolbox

- **Shapebox :** This option allows the user to choose between one of the many built-in shapes. Line, Rectangle, Circle, Triangle, Curve, Ellipse.

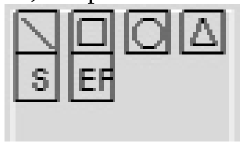


Fig4. Shapebox

- **Colorpalette :** This option enables the user to choose one of the following color for drawing. The icon chosen is displayed on the right side to keep a track of chosen color.

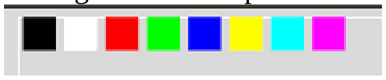


Fig5. Colorpalette

- **SizeBox :** Allows to change the thickness of pencil or the built-in shapes.

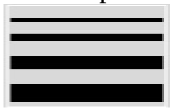


Fig6. Sizebox

3.2 Functionalities:

3.2.1Pencil:

(X_{new}, Y_{new}) and (X_{old}, Y_{old}) are all initialized to -1 when Pencil tool is selected from the menu options both these variables are assigned the current mouse position when Mouse_pressed_CB is called. With every Mouse_Motion_CB (X_{new}, Y_{new}) are assigned the new (x,y) coordinates. GL_LINE is drawn between (X_{new}, Y_{new}) and (X_{old}, Y_{old}) and then (X_{new}, Y_{new}) are copied to (X_{old}, Y_{old})

3.2.2 Brush:

Brush works on the same principle as the Pencil. It is like a thick Pencil i.e. default pixel width large.

3.2.3Eraser:

Eraser is implemented by GL_POLYGON of size of 4x4 pixel of white color.

3.2.4 Spray:

For Spray, 50 randomly generated pixels are colored in the vicinity of the current mouse position.

3.2.5 Circle:

Circle is implemented using the Circle Mid-point algorithm.

Step1: A point is known to lie inside a circle, on the circle and outside it when the equation of the circle F(x,y) evaluates to <0, =0 and >0 respectively. A circle is symmetric in its 8 octants. So if a point is drawn in one octant, 7 more similar points can be drawn on the circle. [3][4][5]

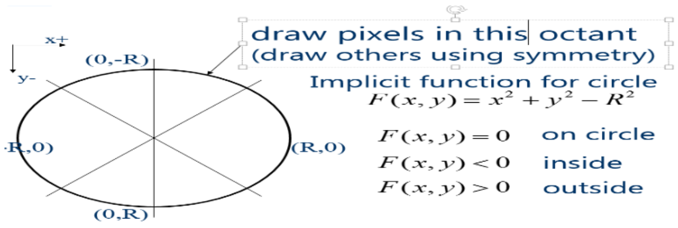

x+
y-
(0,-R)
(-R,0)
(R,0)
(0,R)
draw pixels in this octant
(draw others using symmetry)
Implicit function for circle
$F(x,y) = x^2 + y^2 - R^2$
$F(x,y) = 0$ on circle
$F(x,y) < 0$ inside
$F(x,y) > 0$ outside

fig7: 8 symmetric Octants

At every point when there is an option to choose where to draw the next pixel, it can either be drawn in the East or towards the South-East considering that we are implementing the algorithm for the first octant. This decision can be taken by the decision variable as shown in fig 8.

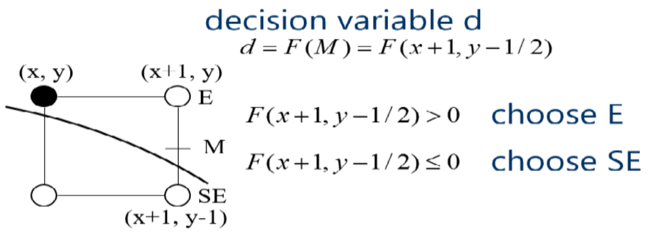
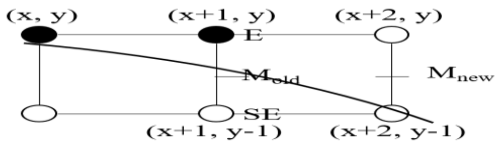

decision variable d
$d = F(M) = F(x+1, y-1/2)$
(x, y)
(x+1, y)
E
M
SE
(x+1, y-1)
$F(x+1, y-1/2) > 0$ choose E
$F(x+1, y-1/2) \le 0$ choose SE

Fig8: Choosing the next pixel.

Depending where the current pixel is drawn new decision points are to be calculated. If a pixel is drawn then the new decision variable is chosen depending where the current pixel is being drawn as described in fig 9 and 10.



$$d_{new} = (x+2)^2 + (y-1/2)^2 - R^2$$
$$d_{old} = (x+1)^2 + (y-1/2)^2 - R^2$$
$$\Delta d = d_{new} - d_{old} = 2x+3$$

Fig9: Change in decision parameter when point chosen is towards east

$$d_{new} = (x+2)^2 + (y-3/2)^2 - R^2$$
$$d_{old} = (x+1)^2 + (y-1/2)^2 - R^2$$
$$\Delta d = d_{new} - d_{old} = 2x - 2y + 5$$

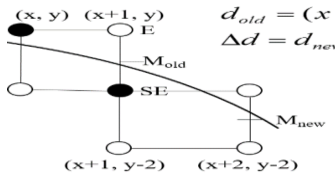
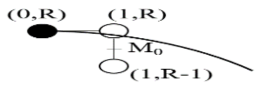


Fig10: Change in decision parameter when point chosen is towards south-east

Initial point chosen is (0,R).



$$d_0 = F(M_0)$$
$$d_0 = F(1, R-1/2)$$
$$d_0 = (1)^2 + (R-1/2)^2 - R^2$$
$$d_0 = 5/4 - R$$

Fig11: Choosing initial decision parameter

In order to avoid any floating point computations new decision variable h h=d-1/4 and hence hstart = 1-R is chosen. Note h > -1/4 can be replaced with h > 0 since h will always have an integer value. The overall algorithm can be concluded in the following pseudocode where one pixel calculated is interpolated to 8 points on the circle.

```
draw circle{
x = 0;
y = R;
h = 1 – R;
setPixel(x,y);
while (y > x) {
        if (h < 0) {/* E chosen */h += 2*x + 3; x++; }
          else {/* SE chosen */h += 2*(x-y) + 5; x++ ; y--;}
        drawpixel(x0, y0, x, y);
}
drawpixel(int x0, int y0, int x, int y){
        renderpixel(x + x0, y + y0);
        renderpixel(-x + x0, y + y0);
        renderpixel(x + x0, -y + y0);
        renderpixel(-x + x0, -y + y0);
        renderpixel(y + x0, x + y0);
        renderpixel(-y + x0, x + y0);
        renderpixel(y + x0, -x + y0);
        renderpixel(-y + x0, -x + y0);
}
```

3.2.6 Ellipse:

Ellipses are symmetrical between quadrants. There four symmetric points from a single (x,y) point. These are not symmetric between the two octants of a quadrant [5][6]. Thus, we must calculate pixel positions along the elliptical arc through one quadrant and then we obtain positions in the remaining 3 quadrants by symmetry. This single point is computed which is a variation generalizing circle Midpoint algorithm. Similar to the circle a decision variable can be used to decide which pixel to be chosen. Since a quadrant is not entirely symmetrical so the demarcation point where decision algorithms is to be changes is done by changing the algorithm after the point where the slope of tangent at the circle is -1. This point can be calculation by equation the slope at that point to -1.

$$Slope = \frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$$

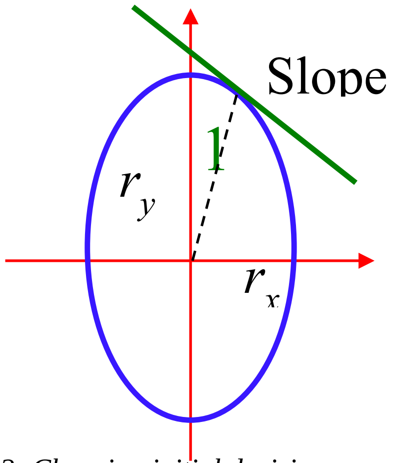


Fig12: Choosing initial decision parameter

3.2.7 Line and Rectangle:

Similar to Pencil, the old and new coordinates are stored in (X_{new}, Y_{new}) and (X_{old}, Y_{old}). GL_LINE_LOOP is completed by connecting the following points : (X_{old}, Y_{old}) , (X_{old}, Y_{new}) , (X_{new}, Y_{old}) , (X_{new}, Y_{new}) to make the rectangle. Similarly GL_LINE_STRIP is used to draw a line between (X_{new}, Y_{new}) and (X_{old}, Y_{old}).

3.2.8 Fill Color:

Color filling works on flood fill algorithm. Starting at a point inside a region, all the surrounding colors are replaced by a specified color by filling the 4-connected pixels until all interior points being replaced. It is a 4 way method which is implemented with std::stack. Algorithm works in the following way:

```
floodFill(GLint x, GLint y, float r, float g, float b){
 Push (pair(x,y)) on stack
 Until stack is not empty
 Get top element of stack tos
 If pixelcolor tos !=  newcolor && tos is not marked visited
 setPixelColor(tos.x, tos.y, newpixelcolor)
 stack.push(tos.x+1 ,tos.y)
 stack.push(tos.x ,tos.y+1)
 stack.push(tos.x ,tos.y-1)
 stack.push(tos.x-1 ,tos.y)
}
```

Fig13: Flood Fill in Action

3.2.9 Drawing Curve:

Bezier curve are implemented in this project. 4 set of control points are chosen and can Interpolate end-points curve is drawn which lies within the convex hull of the control points Cubic Bezier curves gives reasonable design flexibility while avoiding the increased calculations needed with higher order polynomials. Find x and y using Bernstein cubic polynomials

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \ , 0 \le t \le 1.$$

Draw GL_LINES with old vertex and new vertex (found using above equation);.

Fig14: Bezier Curves

3.2.10 Convex Hull:

The convex hull of a set of points is the smallest convex polygon containing the points It is a convex polygon which is nonintersecting and has all internal less than 180 degrees. In a convex polygon, a segment joining any two lies entirely inside the polygon. Graham Scan algorithm has been used to implement Convex Hulls of 10 chosen set of points.

```
GRAHAM_SCAN (Q){
        Find $p_0$ in Q with minimum y-coordinate (and minimum x-coordinate if there are ties).
        Sort the remaining points of Q by polar angle in counterclockwise order with respect to $p_0$.
        TOP [S] = 0
        PUSH($p_0$,S)
        PUSH($p_1$,S)
        PUSH($p_2$,S)
        for $i$ =3 to $m$  -Perform test for each pointp$_3$,...,p$_m$.
        do while the angle between NEXT_TO_TOP[S], TOP[S], and $p_i$ makes a nonleft turn ,
        remove if not a vertex{
            do POP(S)
            PUSH (S, $p_i$)
         return S
        }
}
```

Connect the points present in the stack in order by GL_LINES.

The orientation of three points (a, b, c) in the plane is clockwise, counterclockwise, or collinear based on which a right, left or no turn has to be taken in that order. Orientation can be calculated based on the sign of the determinant Δ(a, b, c).

$$\Delta(a,b,c) = \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix}$$

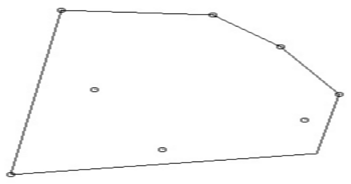


Fig15: Convex Hull

6. Results:

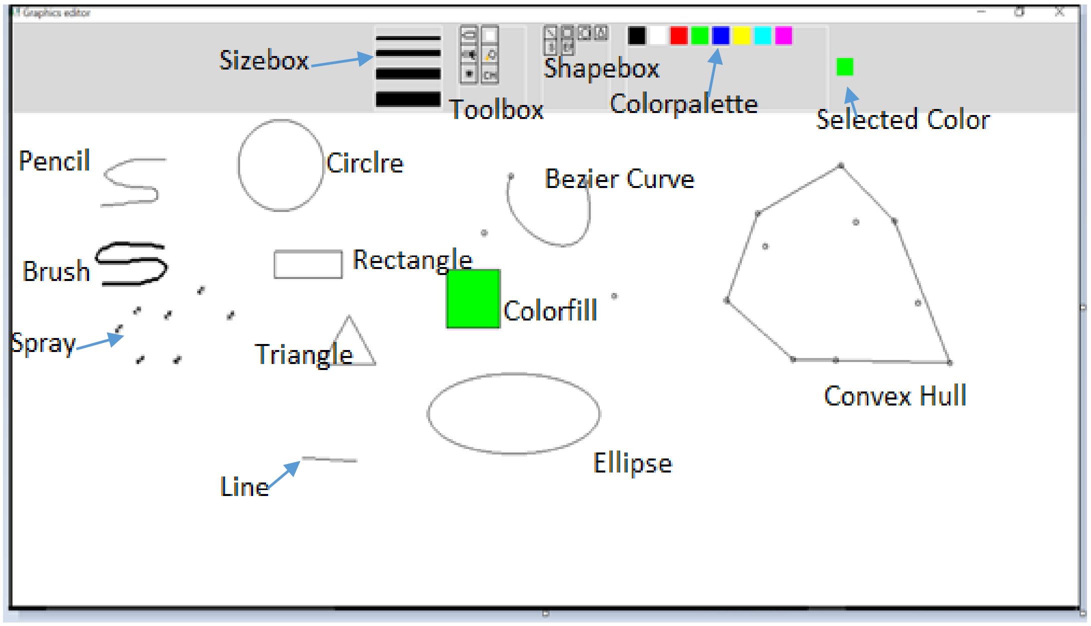


Fig16: Graphics Editor in action.

7. Conclusion:

We had a great experience in the course of designing this editor, which made us discover and learn many things as pertaining to the OpenGL programming. We have tried, to our best potential, to incorporate all the basic level requirements of a normal graphics editor. It supports various objects like circle, rectangle, and many kinds of polygon, convex hull and curves. It also provides color selection, icon based user friendly interface. This project meets all the basic requirements of a user, successfully. The project has been implemented using optimized algorithms.

8. Future Work:

- 3D Figure.
- Selecting Polygons on Canvas.
- Translation.
- Rotation.
- Crop Image.
- File Handling.
- Options to undo/redo the things.

9. References:

1. http://csclab.murraystate.edu/bob.pilgrim/515/lectures_02.html
2. https://www.google.com/search?q=opengl+window+coordinates&safe=off&espv=2&biw=1422&bih=687&tbm=isch&tbo=u&source=univ&sa=X&ved=0ahUKEwjVgO2Zls7JAhUQ9GMKHZbACrMQsAQINg#imgrc=ez6gLdoB9Kf0tM%3A
3. http://www.tutorialspoint.com/computer_graphics/circle_generation_algorithm.htm
4. https://www.youtube.com/watch?v=QeOvZTYlmAI
5. https://www.youtube.com/watch?v=frrBDwmkQ0E
6. http://geofhagopian.net/sablog/Slog-october/slog-10-25-05.htm
7. https://en.wikipedia.org/wiki/B%C3%A9zier_curve
8. http://www.dcs.gla.ac.uk/~pat/52233/slides/Hull1x1.pdf
9. Fundamentals of Computer Graphics 3rd ed. - P. Shirley, S. Marschner (CRC, 2009) WW (1)
10. http://people.wku.edu/qi.li/teaching/446/cg1_OpenGL.pdf
11. http://www.glprogramming.com/red/chapter01.html
12. http://www.mcs.csueastbay.edu/~grewe/CS4840/OpenGL/menu.htm
13. https://www.opengl.org/resources/libraries/glut/
14. http://https://www.opengl.org/archives/resources/faq/technical/transformations.htm
15. http://rosettacode.org/wiki/Bitmap/Midpoint_circle_algorithm
16. ftp://ftp.cis.upenn.edu/pub/.../Lecture5.ppt
17. www.cse.iitd.ernet.in/~amitk/cs201/ConvexHulls.ppt
18. people.csail.mit.edu/thies/6.046-web/graham.ppt
19. cs.brown.edu/courses/.../15%20Convex%20Hull.pptx
20. https://en.wikipedia.org/wiki/B%C3%A9zier_curve