Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

19 July 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 1: Study of Basic Output Primitives in C++ using OpenGL

Aim:

- To create an output window using OPENGL and to draw the following basic output primitives - POINTS, LINES, LINE STRIP, LINE LOOP, TRIANGLES, QUADS, QUAD STRIP, POLYGON.
- To create an output window and draw a checkerboard using OpenGL.
- To create an output window and draw a house using POINTS, LINES, TRIANGLES and QUADS/POLYGON.

Code: Basic Primitives:

```
//Basic output primitives using OpenGL
//Shapes: Points, Lines, Line Strips, Line Loops, Triangles, Quads, Quad Strips and Polygons

//Documentation: https://docs.microsoft.com/en-us/windows/win32/opengl/gl-functions

#include <windows.h>
#include <GL/glut.h>

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;

void initializeDisplay();
void displayShapes();

void displayPoints();
void displayLines();
void displayLineStrips();
void displayLineLoops();
void displayTriangles();
void displayQuads();
void displayQuadStrips();
void displayPolygons();

int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("Basic Shapes");
    glutDisplayFunc(displayShapes);
    initializeDisplay();
    glutMainLoop();

    return 1;
}

void initializeDisplay(){
    glClearColor(1.0, 1.0, 1.0, 0.0);   //The glClearColor function specifies clear values for the color buffers.
    glColor3f(255.0f, 0.0f, 127.0f);    //Sets the current color.
    glPointSize(5);                    //The glPointSize function specifies the diameter of rasterized points.
    glMatrixMode(GL_PROJECTION);        //The glMatrixMode function specifies which matrix is the current matrix.
    glLoadIdentity();                   //The glLoadIdentity function replaces the current matrix with the identity matrix.
```

```
42      gluOrtho2D(0.0, 800.0, 0.0, 600.0); //The gluOrtho2D function defines
    a 2-D orthographic projection matrix.
43  }
44
45  void displayShapes(){
46      glClear(GL_COLOR_BUFFER_BIT);
47
48      displayPoints();
49      displayLines();
50      displayLineStrips();
51      displayLineLoops();
52      displayTriangles();
53      displayQuads();
54      displayQuadStrips();
55      displayPolygons();
56
57      glFlush();
58  }
59
60  void displayPoints(){
61      //Treats each vertex as a single point.
62      //Vertex n defines point n. N points are drawn.
63
64      glBegin(GL_POINTS);
65
66      glVertex2d(10, 10);
67      glVertex2d(15, 15);
68      glVertex2d(20, 20);
69      glVertex2d(25, 25);
70      glVertex2d(30, 30);
71
72      glEnd();
73  }
74
75  void displayLines(){
76      //Treats each pair of vertices as an independent line segment.
77      //Vertices 2n - 1 and 2n define line n. N/2 lines are drawn.
78
79      glBegin(GL_LINES);
80
81      glVertex2d(0, 0);
82      glVertex2d(800, 600);
83
84      glEnd();
85  }
86
87  void displayLineStrips(){
88      //Draws a connected group of line segments from the first vertex to
    the last.
89      // Vertices n and n+1 define line n. N - 1 lines are drawn.
90
```

```
91      glBegin(GL_LINE_STRIP);
92
93      glVertex2d(100, 100);
94      glVertex2d(200, 200);
95
96      glVertex2d(200, 500);
97      glVertex2d(500, 600);
98
99      glEnd();
100 }
101
102 void displayLineLoops(){
103     //Draws a connected group of line segments from the first vertex to
        the last,
104     //then back to the first. Vertices n and n + 1 define line n.
105     //The last line, however, is defined by vertices N and 1. N lines are
        drawn.
106
107     glBegin(GL_LINE_LOOP);
108
109     glVertex2d(650, 250);
110     glVertex2d(750, 250);
111     glVertex2d(750, 350);
112     glVertex2d(650, 350);
113
114     glEnd();
115 }
116
117 void displayTriangles(){
118     //Treats each triplet of vertices as an independent triangle.
119     //Vertices 3n - 2, 3n - 1, and 3n define triangle n. N/3 triangles are
        drawn.
120
121     glBegin(GL_TRIANGLES);
122
123     glVertex2d(170, 170);
124     glVertex2d(170, 220);
125     glVertex2d(150, 200);
126
127     glEnd();
128 }
129
130 void displayQuads(){
131     //Treats each group of four vertices as an independent quadrilateral.
132     //Vertices 4n - 3, 4n - 2, 4n - 1, and 4n define quadrilateral n. N/4
        quadrilaterals are drawn.
133
134     glBegin(GL_QUADS);
135
136     glVertex2d(400, 400);
137     glVertex2d(450, 400);
```

```
138     glVertex2d(450, 500);
139     glVertex2d(400, 500);
140 
141     glEnd();
142 }
143 
144 void displayQuadStrips(){
145     //Draws a connected group of quadrilaterals.
146     //One quadrilateral is defined for each pair of vertices presented
    after the first pair.
147     //Vertices 2n - 1, 2n, 2n + 2, and 2n + 1 define quadrilateral n. N/2
    - 1 quadrilaterals are drawn.
148     //Note that the order in which vertices are used to construct a
    quadrilateral
149     //from strip data is different from that used with independent data.
150 
151     glBegin(GL_QUAD_STRIP);
152 
153     glVertex2d(320, 320);
154     glVertex2d(360, 320);
155 
156     glVertex2d(320, 360);
157     glVertex2d(360, 360);
158 
159     glVertex2d(360, 390);
160     glVertex2d(390, 390);
161 
162     glEnd();
163 
164 }
165 
166 void displayPolygons(){
167     //Draws a single, convex polygon.
168     //Vertices 1 through N define this polygon.
169 
170     glBegin(GL_POLYGON);
171 
172     glVertex2d(510, 0);
173     glVertex2d(500, 20);
174     glVertex2d(500, 40);
175     glVertex2d(510, 60);
176     glVertex2d(520, 40);
177     glVertex2d(520, 20);
178 
179     glEnd();
180 }
```

Code: Checker Board

```
//To draw a checkerboard using OpenGL

#include <windows.h>
#include <GL/glut.h>

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;

void initializeDisplay();
void displayCheckerboard();
void drawSquare(GLint x, GLint y, GLint x_step, GLint y_step);

int CURRENT_COLOR = 0;      //Global variable to keep track of current checker color

int main(int argc, char **argv){
    glutInit(&argc, argv);                          //Initialize glut
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);      //Set display mode
    glutInitWindowPosition(100, 100);               //Set Window position
    glutInitWindowSize(800, 600);                   //Set window size
    glutCreateWindow("OpenGL Checkerboard");        //Create display window with title

    initializeDisplay();                            //Initialization procedure
    glutDisplayFunc(displayCheckerboard);           //Send graphics to display window
    glutMainLoop();                                 //Display everything and wait

    return 1;
}

void initializeDisplay(){
    //Initialize the display parameters

    glClearColor(0, 1, 1, 0);       //Display window color
    glMatrixMode(GL_PROJECTION);    //Choose projection
    gluOrtho2D(0, 800, 0, 600);     //Set transformation
}

void displayCheckerboard(){
    //Displays an 8x8 checkerboard

    glClear(GL_COLOR_BUFFER_BIT);   //Clear display window
    GLint x, y;
    GLint x_step = 100, y_step = 75;
```

```
43      //For 8x8 board in an 800x600 window, x_step = 800/8 = 100, y_step =
    600/8 = 75
44
45      for(x = 0; x <= 800; x += x_step){
46          for(y = 0; y <= 600; y += y_step){
47              drawSquare(x, y, x_step, y_step);
48          }
49      }
50
51      glFlush();  //Forces execution of OpenGL functions in finite time.
52  }
53
54  void drawSquare(GLint x, GLint y, GLint x_step, GLint y_step){
55      //Draws a square, given a pair of coordinates and step sizes
56
57      GLint x1, y1, x2, y2, x3, y3, x4, y4;
58
59      //Vertex 1
60      x1 = x;
61      y1= y + y_step;
62
63      //Vertex 2
64      x2 = x + x_step;
65      y2 = y + y_step;
66
67      //Vertex 3
68      x3 = x + x_step;
69      y3 = y;
70
71      //Vertex 4
72      x4 = x;
73      y4 = y;
74
75      if(CURRENT_COLOR == 0){
76          glColor3f(1, 1, 1);      //White color
77          CURRENT_COLOR = 1;
78      }
79      else{
80          glColor3f(0, 0, 0);      //Black color
81          CURRENT_COLOR = 0;
82      }
83
84      glBegin(GL_POLYGON);
85
86      glVertex2i(x1, y1);
87      glVertex2i(x2, y2);
88      glVertex2i(x3, y3);
89      glVertex2i(x4, y4);
90
91      glEnd();
92  }
```

Code: House

```
//To draw a house using OpenGL

#include <windows.h>
#include <GL/glut.h>

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;

void initializeDisplay();
void drawHouse();


int main(int argc, char **argv){
    glutInit(&argc, argv);                          //Initialize glut
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);  //Set display mode
    glutInitWindowPosition(100, 100);           //Set Window position
    glutInitWindowSize(800, 600);               //Set window size
    glutCreateWindow("OpenGL - House");         //Create display window
    with title

    initializeDisplay();                            //Initialization procedure
    glutDisplayFunc(drawHouse);                 //Send graphics to display
    window
    glutMainLoop();                                 //Display everything and
    wait

    return 1;
}

void initializeDisplay(){
    glClearColor(0.5, 0.1, 1, 0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 800, 0, 600);
}

void drawHouse(){
    glClear(GL_COLOR_BUFFER_BIT);       //Clear display window

    //Ground
    glColor3f(0.5, 0.3, 0);
    glBegin(GL_POLYGON);

    glVertex2i(0, 100);
    glVertex2i(800, 100);
    glVertex2i(800, 0);
    glVertex2i(0, 0);

```

```
45      glEnd();
46
47      //Side Roof
48      glColor3f(0.3, 0.5, 0.8);
49      glBegin(GL_POLYGON);
50
51      glVertex2i(200, 500);
52      glVertex2i(600, 500);
53      glVertex2i(700, 350);
54      glVertex2i(300, 350);
55
56      glEnd();
57
58      //Front Roof
59      glColor3f(0.1, 0.5, 0.0);
60      glBegin(GL_TRIANGLES);
61
62      glVertex2i(200, 500);
63      glVertex2i(100, 350);
64      glVertex2i(300, 350);
65
66      glEnd();
67
68      //Front Wall
69      glColor3f(0.7, 0.2, 0.3);
70      glBegin(GL_POLYGON);
71
72      glVertex2i(100, 350);
73      glVertex2i(300, 350);
74      glVertex2i(300, 100);
75      glVertex2i(100, 100);
76
77      glEnd();
78
79      //Side Wall
80      glColor3f(0.1, 0.2, 0.3);
81      glBegin(GL_POLYGON);
82
83      glVertex2i(300, 350);
84      glVertex2i(700, 350);
85      glVertex2i(700, 100);
86      glVertex2i(300, 100);
87
88      glEnd();
89
90      //Front Door
91      glColor3f(0.7, 0.2, 0.9);
92      glBegin(GL_POLYGON);
93
94      glVertex2i(150, 250);
95      glVertex2i(250, 250);
```

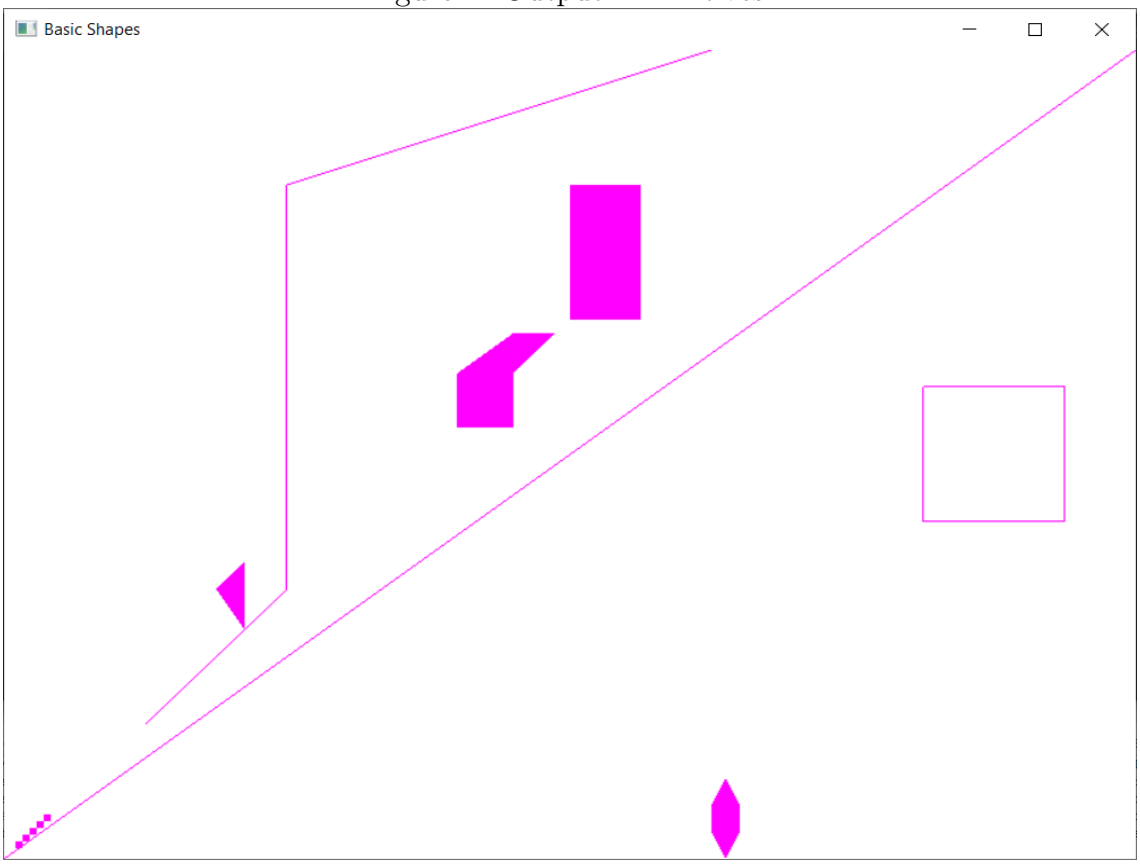
```
96      glVertex2i(250, 100);
97      glVertex2i(150, 100);
98
99      glEnd();
100
101     //Front Door Lock
102     glColor3f(0.3, 0.7, 0.9);
103     glPointSize(15);
104     glBegin(GL_POINTS);
105
106     glVertex2i(170, 170);
107
108     glEnd();
109
110     //Front Door Frame
111     glColor3f(1, 1, 1);
112     glLineWidth(2.5);
113     glBegin(GL_LINES);
114
115     glVertex2i(150, 250);
116     glVertex2i(250, 250);
117
118     glVertex2i(150, 100);
119     glVertex2i(150, 250);
120
121     glVertex2i(250, 100);
122     glVertex2i(250, 250);
123
124
125     glVertex2i(150, 100);
126     glVertex2i(250, 100);
127
128
129     glEnd();
130
131
132     //Pathway
133     glColor3f(0.3, 0.5, 0.7);
134     glLineWidth(5);
135     glBegin(GL_POLYGON);
136
137     glVertex2i(150, 100);
138     glVertex2i(250, 100);
139     glVertex2i(210, 0);
140     glVertex2i(40, 0);
141
142     glEnd();
143
144     //Windows
145
146     //Window - 1
```

```
147     glColor3f(0.2, 0.4, 0.3);
148     glBegin(GL_POLYGON);
149 
150     glVertex2i(330, 320);
151     glVertex2i(450, 320);
152     glVertex2i(450, 230);
153     glVertex2i(330, 230);
154 
155     glEnd();
156 
157     //Window - 2
158     glColor3f(0.2, 0.4, 0.3);
159     glBegin(GL_POLYGON);
160 
161     glVertex2i(530, 320);
162     glVertex2i(650, 320);
163     glVertex2i(650, 230);
164     glVertex2i(530, 230);
165 
166     glEnd();
167 
168     //Window Borders
169 
170     //Window - 1
171     glColor3f(0.1, 0.7, 0.5);
172     glLineWidth(5);
173     glBegin(GL_LINES);
174 
175     glVertex2i(390, 320);
176     glVertex2i(390, 230);
177     glVertex2i(330, 273);
178     glVertex2i(450, 273);
179 
180     glEnd();
181 
182     //Window -2
183     glColor3f(0.1, 0.7, 0.5);
184     glLineWidth(5);
185     glBegin(GL_LINES);
186 
187     glVertex2i(590, 320);
188     glVertex2i(590, 230);
189     glVertex2i(530, 273);
190     glVertex2i(650, 273);
191 
192     glEnd();
193 
194     //Decoration
195     glColor3f(0.2, 0.4, 0.2);
196     glPointSize(5);
197     glBegin(GL_POINTS);
```

```
198
199     GLint x = 310;
200     for(x; x <= 690; x += 10){
201         glVertex2i(x, 120);
202     }
203
204     glEnd();
205
206     //Hexagonal Sun
207     glColor3f(0.8, 1, 0);
208     glBegin(GL_POLYGON);
209
210     glVertex2i(50, 500);
211     glVertex2i(75, 550);
212     glVertex2i(125, 550);
213     glVertex2i(150, 500);
214     glVertex2i(125, 450);
215     glVertex2i(75, 450);
216
217     glEnd();
218
219
220     glFlush();      //Flush the output to the display
221 }
```

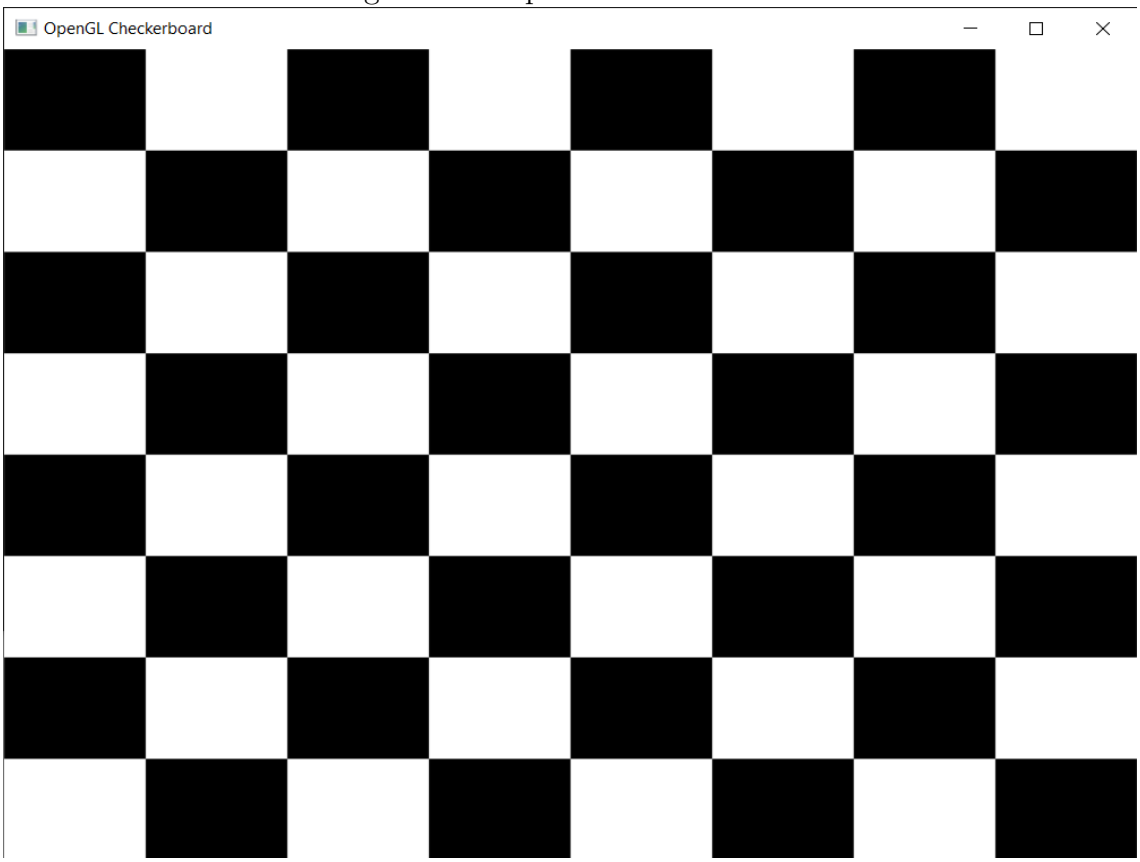
Output: Primitives:

Figure 1: Output: Primitives.



Output: Checker Board:

Figure 2: Output: Checker Board.



Output: House:

Figure 3: Output: House.

Learning Outcome:

- I configured **OpenGL** and **GLUT Framework** on my system using the CodeBlocks IDE.
- I learnt about OpenGL and its usage in the high-performance graphics industry - like creating animations and games.
- I learnt to draw some **primitive output shapes** like points, lines, line strips, line loops, triangles, quads, quad strips and polygons using GLUT's inbuilt functions.
- I understood how to **initialize a new GLUT output display** with colors, matrix mode, window title, window size etc.
- I learnt how these shapes are plotted using the **glVertex2d()** function.
- I was able to construct a basic **8x8 checkerboard** using the inbuilt primitive functions and was able to color the checkerboard appropriately.
- I was able to draw a **simple house** with shapes like triangles, quads and polygons. I was also able to color the house with different shades.
- I understood that the OpenGL uses a **coordinate system** to map the output shapes onto the display window.

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

31 July 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 2: Line Drawing Using Digital Differential Analyzer Algorithm

Aim:

To plot points that make up the line with endpoints (x0, y0) and (xn, yn) using DDA line drawing algorithm.

- Case 1: +ve slope Left to Right line
- Case 2: +ve slope Right to Left line
- Case 3: -ve slope Left to Right line
- Case 4: -ve slope Right to Left line

Each case has two subdivisions (i) $|m| <= 1$ (ii) $|m| > 1$

Note that all four cases of line drawing must be given as test cases.

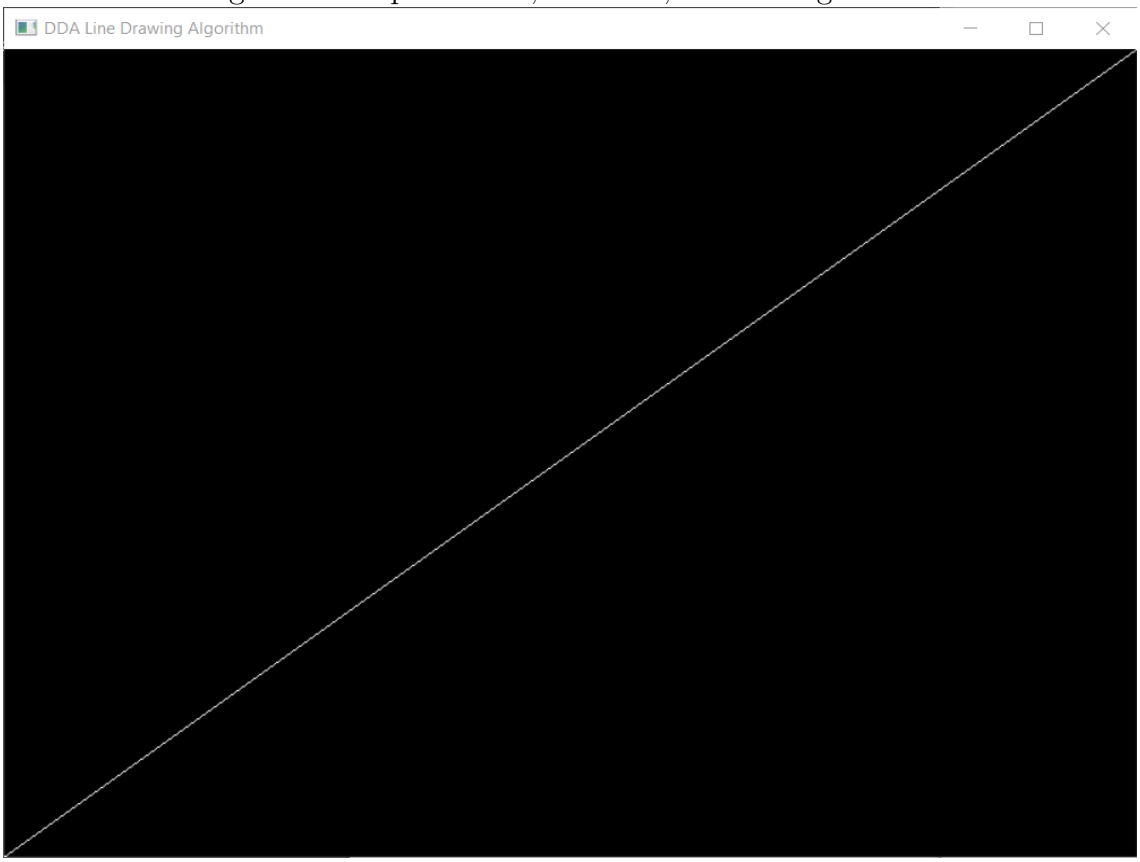
Code: DDA:

```
//To implement the DDA Line Drawing Algorithm
// DDA: Digital Differential Algorithm

#include <windows.h>
#include <stdio.h>
#include <GL/glut.h>

GLfloat x1, y1, x2, y2;

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;

void initializeDisplay();
void drawLine();
GLint round(GLfloat num);

int main(int argc, char **argv){


    printf("\nEnter the value of X1: ");
    scanf("%f", &x1);

    printf("\nEnter the value of Y1: ");
    scanf("%f", &y1);

    printf("\nEnter the value of X2: ");
    scanf("%f", &x2);

    printf("\nEnter the value of Y2: ");
    scanf("%f", &y2);

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("DDA Line Drawing Algorithm");

    initializeDisplay();
    glutDisplayFunc(drawLine);
    glutMainLoop();

    return 1;
}

void initializeDisplay(){
    //Initialize the display parameters

```

```
48      glClearColor(0, 1, 1, 0);       //Display window color
49      glMatrixMode(GL_PROJECTION);    //Choose projection
50      gluOrtho2D(0, 800, 0, 600);     //Set transformation
51  }
52
53  void drawLine(){
54      GLfloat dx, dy, k, x_inc, y_inc, x, y, slope_max;
55
56      dx = x2 - x1;
57      dy = y2 - y1;
58
59      if(abs(dx) > abs(dy)){
60          // |slope| < 0
61          slope_max = abs(dx);
62      } else {
63          // |slope| >= 0
64          slope_max = abs(dy);
65      }
66
67      // if dx/dy >= slope_max, then x_inc/y_inc will be >= 1
68      // respectively, and the other will be < 1.
69      // increments will be calculated on this basis
70      // according to the DDA Algorithm
71      x_inc = dx/slope_max;
72      y_inc = dy/slope_max;
73
74      //Initial point
75      x = x1;
76      y = y1;
77
78      glBegin(GL_POINTS);
79      glVertex2i(x, y);
80
81      //Plot for all points from 1 to slope_max
82      for(k = 1; k <= slope_max; k++){
83          x += x_inc;
84          y += y_inc;
85
86          glVertex2i(x, y);
87      }
88
89      glEnd();
90      glFlush();
91  }
```

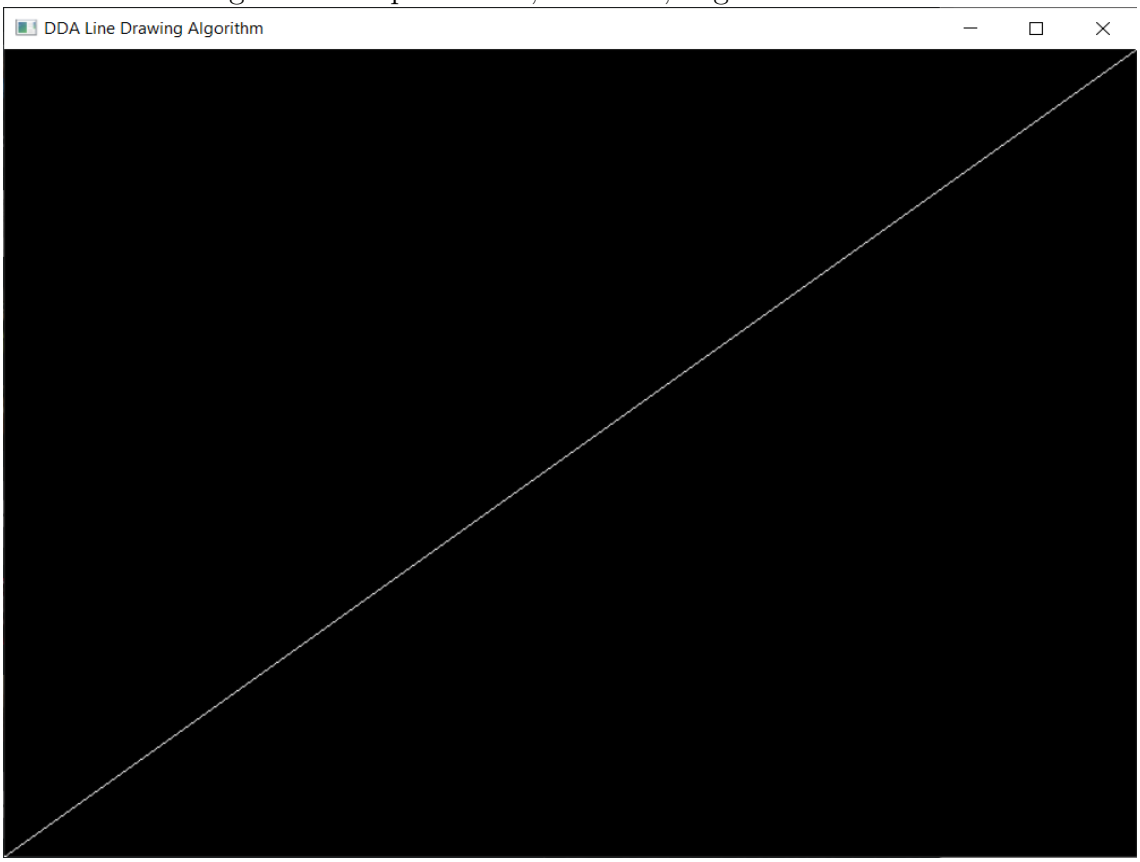
Output: DDA Case 1 - (0, 0) to (800, 600):

Figure 1: Output: DDA, M <= 1, Left to Right Line.



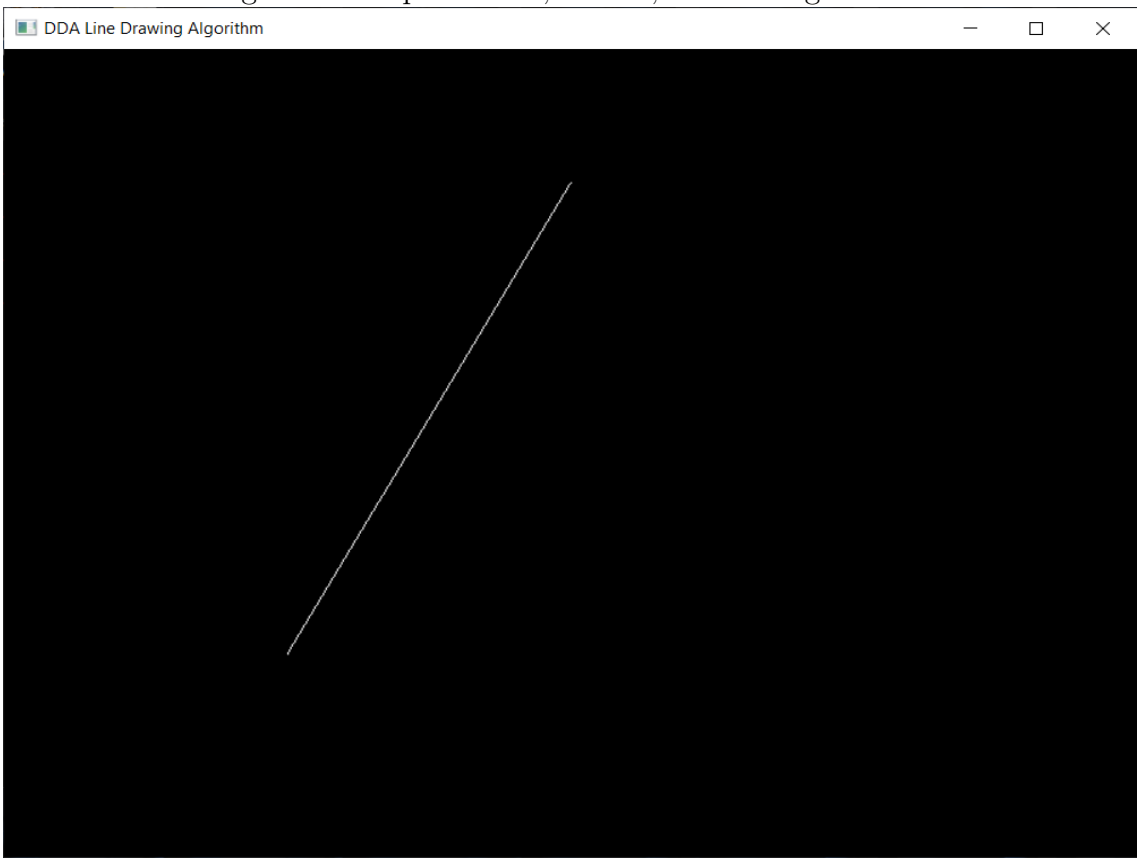
Output: DDA Case 2 - (800, 600) to (0, 0):

Figure 2: Output: DDA, M <= 1, Right to Left Line.



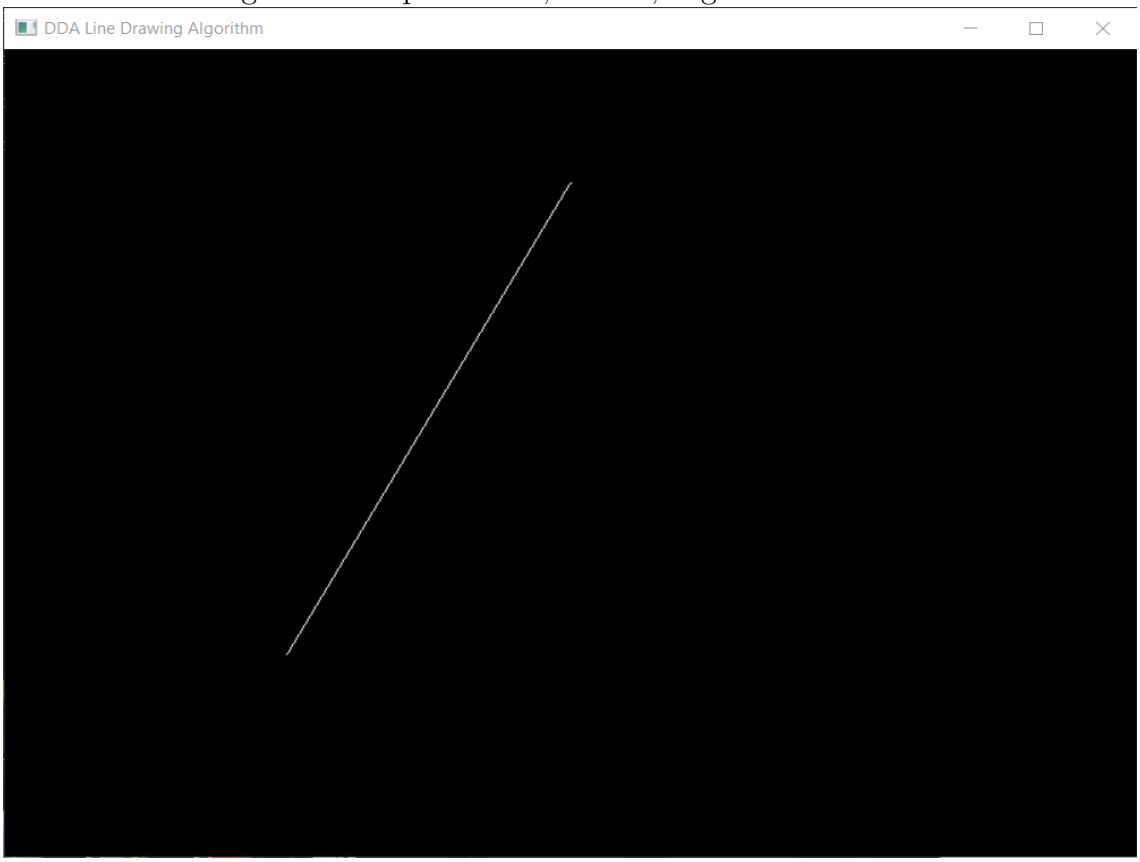
Output: DDA Case 3 - (200, 150) to (400, 500):

Figure 3: Output: DDA, M > 1, Left to Right Line.



Output: DDA Case 4 - (400, 500) to (200, 150):

Figure 4: Output: DDA, M > 1, Right to Left Line.



Learning Outcome:

- I understood the **Digital Differential Analyzer Algorithm**'s working.
- I implemented the DDA Algorithm using an OpenGL program.
- I understood how points are plotted and how increments are calculated based on slope & Δx *and* Δy values in DDA Algorithm.
- I understood that DDA Algorithm makes fine approximations and rounding off's which might be a pitfall when accurate diagrams are required.
- I understood that DDA Algorithm is less efficient and less precise than the Bresenham's Line Algorithm.
- I was able to output all different test cases appropriately to verify the correctness of my program to implement DDA Algorithm.

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

31 July 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 3: Line Drawing Using Bresenham's Algorithm

Aim:

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using Bresenham's Line Drawing Algorithm.

- Case 1: +ve slope Left to Right line
- Case 2: +ve slope Right to Left line
- Case 3: -ve slope Left to Right line
- Case 4: -ve slope Right to Left line

Each case has two subdivisions (i) $|m| <= 1$ (ii) $|m| > 1$

Note that all four cases of line drawing must be given as test cases.

Code: Bresenham's Line Drawing Algorithm:

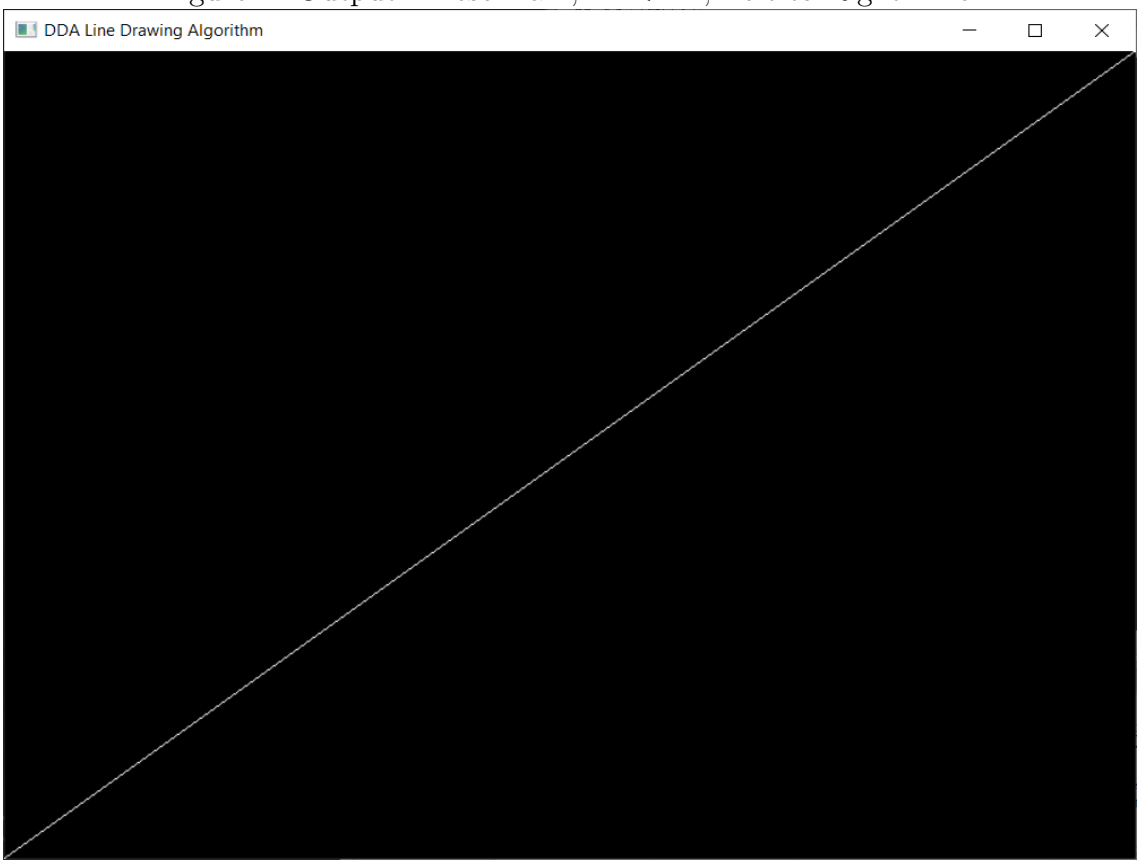
```
//To implement the Bresenham Line Drawing Algorithm

#include <windows.h>
#include <stdio.h>
#include <GL/glut.h>

GLfloat x1, y1, x2, y2;

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 600;

void initializeDisplay();
void drawLine();
GLint round(GLfloat num);

int main(int argc, char **argv){

    printf("\nEnter the value of X1: ");
    scanf("%f", &x1);

    printf("\nEnter the value of Y1: ");
    scanf("%f", &y1);

    printf("\nEnter the value of X2: ");
    scanf("%f", &x2);

    printf("\nEnter the value of Y2: ");
    scanf("%f", &y2);

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("DDA Line Drawing Algorithm");

    initializeDisplay();
    glutDisplayFunc(drawLine);
    glutMainLoop();

    return 1;
}

void initializeDisplay(){
    //Initialize the display parameters

    glClearColor(0, 1, 1, 0);       //Display window color
    glMatrixMode(GL_PROJECTION);    //Choose projection
```

```
48      gluOrtho2D(0, 800, 0, 600);      //Set transformation
49  }
50
51  void drawLine(){
52      GLint dx, dy, x, y, p, inc_x1, inc_y1, inc_p1, inc_p2;
53      GLint x_sign, y_sign;
54
55      dx = x2 - x1;
56      dy = y2 - y1;
57
58      //note the sign for directionality
59      x_sign = dx/abs(dx);
60      y_sign = dy/abs(dy);
61
62      //increment is by 1, and in the direction of +/-
63      inc_x1 = 1 * x_sign;
64      inc_y1 = 1 * y_sign;
65
66      //change the differences to absolute values (crucial step)
67      dx = abs(dx);
68      dy = abs(dy);
69
70      //initial coordinates
71      x = x1;
72      y = y1;
73
74      glBegin(GL_POINTS);
75
76      if(abs(dx) > abs(dy)){
77          //X difference > Y difference
78          glVertex2i(x, y);
79
80          p = (2 * dy) - dx;
81          inc_p1 = 2 * (dy - dx);
82          inc_p2 = 2 * dy;
83
84          //plot for dx number of points
85          for(GLint i = 0; i < dx; i++){
86              if(p >= 0){
87                  y += inc_y1;
88                  p += inc_p1;
89              }
90              else{
91                  p += inc_p2;
92              }
93
94              x += inc_x1;
95
96
97              glVertex2i(x, y);
98          }
```

```
99      }
100     else{
101         //X difference <= Y difference
102         glVertex2i(x, y);
103 
104         p = (2 * dx) - dy;
105         inc_p1 = 2 * (dx - dy);
106         inc_p2 = 2 * dx;
107 
108         //plot for dy number of points
109         for(GLint i = 0; i < dy; i++){
110             if(p >= 0){
111                 x += inc_x1;
112                 p += inc_p1;
113             }
114             else{
115                 p += inc_p2;
116             }
117 
118             y += inc_y1;
119 
120             glVertex2i(x, y);
121         }
122     }
123 
124     glEnd();
125     glFlush();
126 }
```

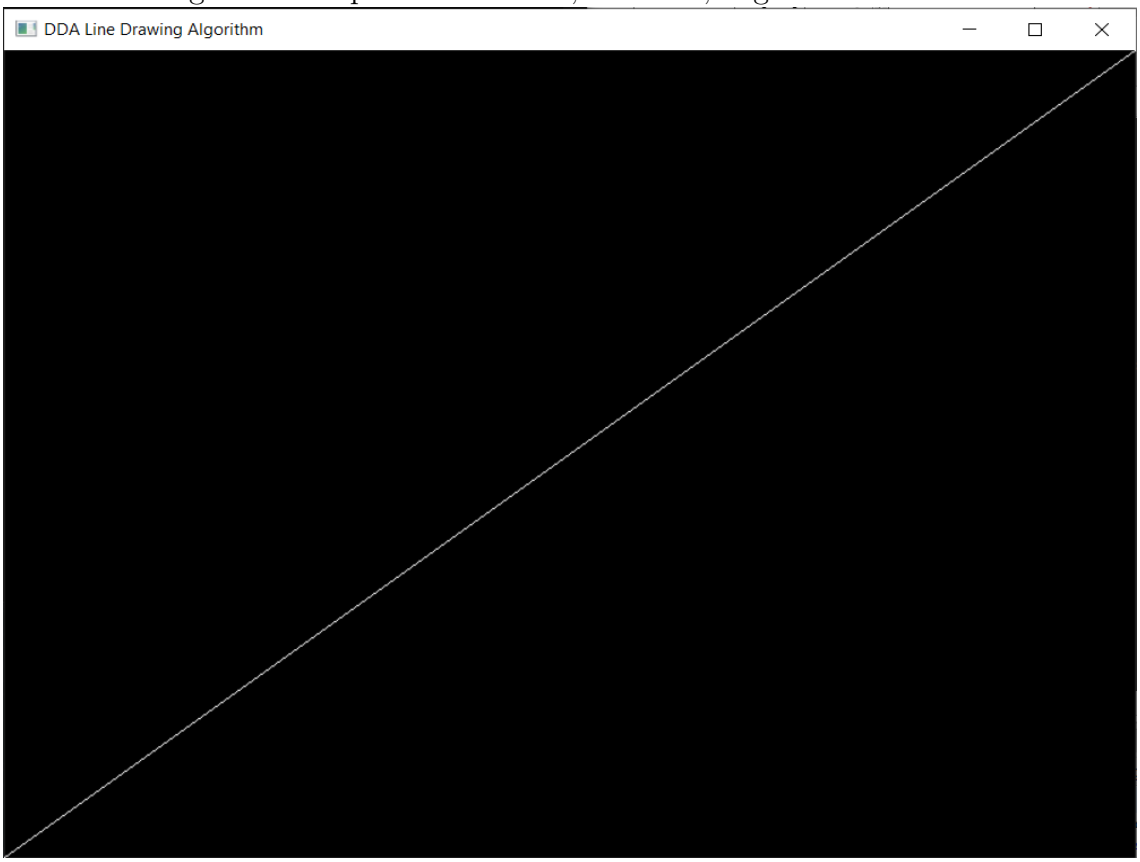
Output: Bresenham Case 1 - (0, 0) to (800, 600):

Figure 1: Output: Bresenham, M <= 1, Left to Right Line.



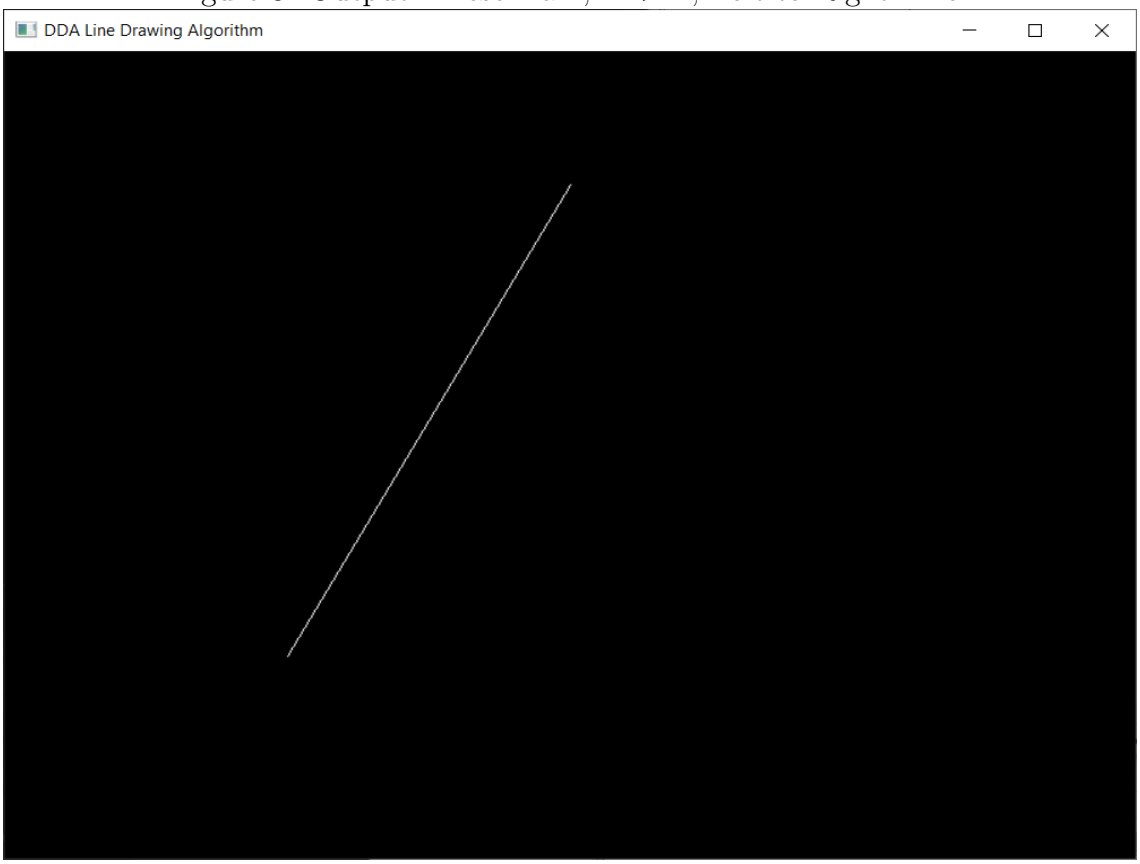
Output: Bresenham Case 2 - (800, 600) to (0, 0):

Figure 2: Output: Bresenham, M <= 1, Right to Left Line.



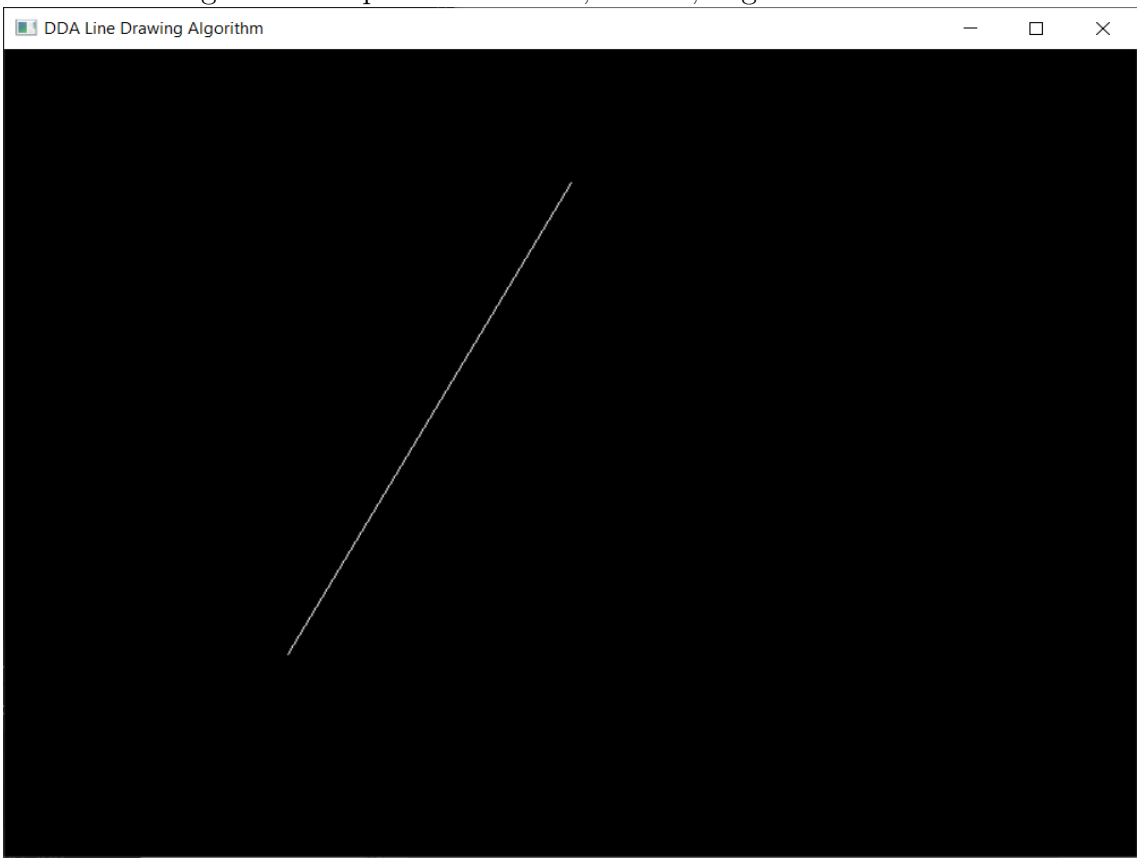
Output: Bresenham Case 3 - (200, 150) to (400, 500):

Figure 3: Output: Bresenham, M > 1, Left to Right Line.



Output: Bresenham Case 4 - (400, 500) to (200, 150):

Figure 4: Output: Bresenham, M > 1, Right to Left Line.



Learning Outcome:

- I understood the **Bresenham's Line Drawing Algorithm**'s working.
- I implemented the Bresenham Line algorithm using an OpenGL program.
- I understood how points are plotted and how increments are calculated based on the Δx *and* Δy values.
- I understood that there are two different calculations to be followed in the Bresenham's algorithm, based on the difference between the Δx *and* Δy values.
- I understood how each iteration increments x and y based on a parameter value p and how p changes it's value in each iteration.
- I understood that Bresenham's Algorithm is faster than DDA Algorithm due to purely integral calculations, avoiding the necessity for rounding off.
- I understood that Bresenham's Algorithm is also more optimal and precise, compared to DDA Algorithm.
- I was able to output all different test cases appropriately to verify the correctness of my program to implement Bresenham's Line Drawing Algorithm.

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

09 August 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 4: Midpoint Circle Drawing Algorithm in C++ using OpenGL

Aim:

a) To plot points that make up the circle with center (xc,yc) and radius r using Midpoint circle drawing algorithm. Give atleast 2 test cases.

- Case 1: With center (0,0)
- Case 2: With center (xc,yc)

b) To draw any object using line and circle drawing algorithms.

Code: Midpoint Circle Drawing Algorithm:

```
/* To draw a circle using the midpoint circle drawing algorithm with
   OpenGL */

#include <windows.h>
#include <stdio.h>
#include <GL/glut.h>

GLint x, y, radius; //Variables for circle

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 800;

class screenPoint{
    /* Class for a coordinate point */

private:
    GLint x, y;

public:
    screenPoint(){
        //Default constructor, initialize to (0, 0)
        x = y = 0;
    }

    screenPoint(GLint xCoord, GLint yCoord){
        //Constructor with preset points
        x = xCoord;
        y = yCoord;
    }

    void setCoords(GLint xCoord, GLint yCoord){
        //Function to set coordinates
        x = xCoord;
        y = yCoord;
    }

    GLint getX() const{
        //Return x coordinate
        return x;
    }

    GLint getY() const{
        //Return y coordinate
        return y;
    }

    void incX(){
```

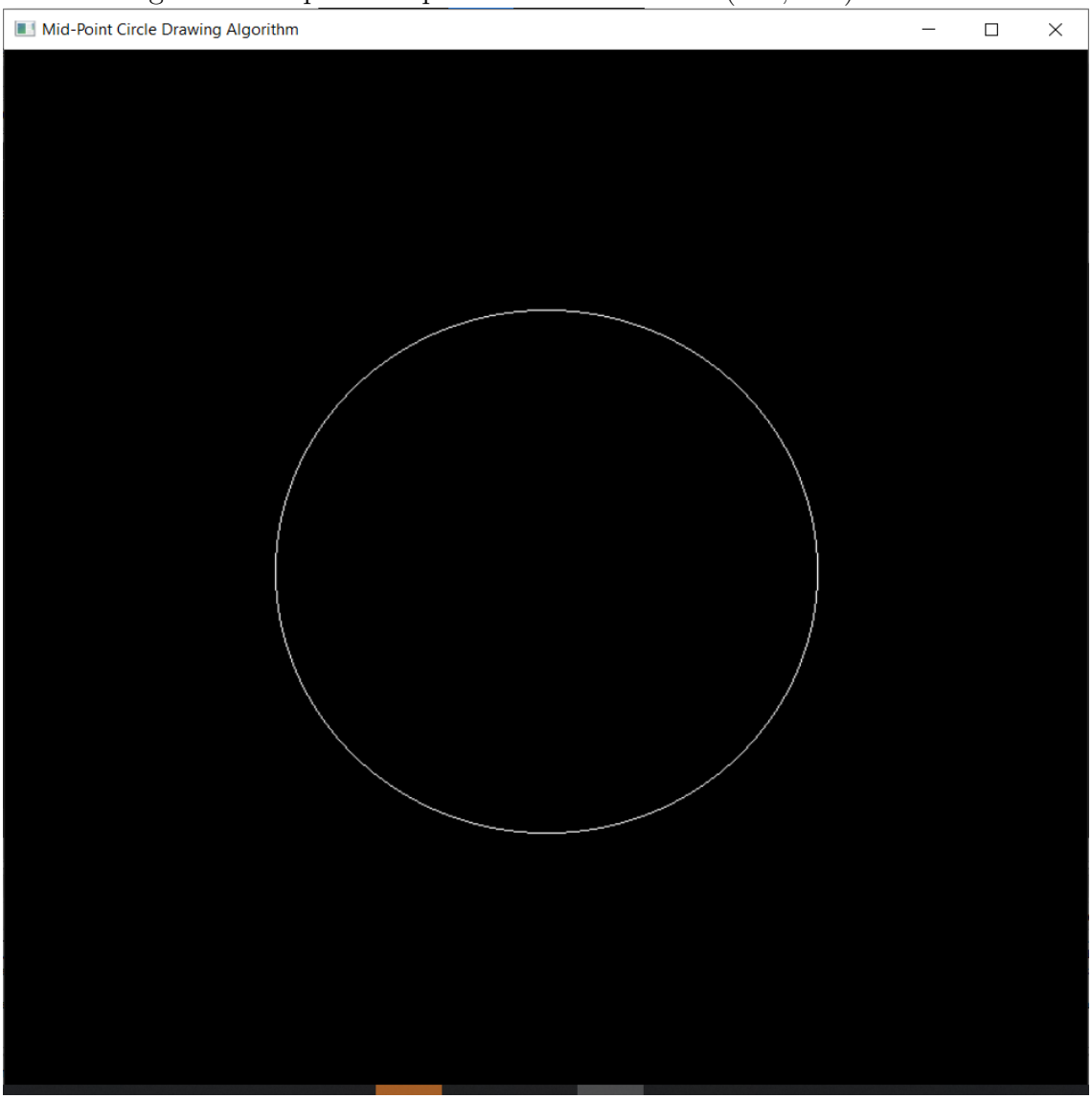
```
47          //Increment x coordinate
48          x++;
49      }
50
51      void decY(){
52          //Decrement y coordinate
53          y--;
54      }
55
56      void printCoords(){
57          //Print current coordinates
58          printf("\nX: %d\tY: %d", x, y);
59      }
60  };
61
62  void initializeDisplay();
63  void drawCircle();
64  void setPixel(GLint xCoord, GLint yCoord);
65  void circleMidPoint(GLint xc, GLint yc, GLint radius);
66  void circlePlotPoints(GLint xc, GLint yc, screenPoint circlePoint);
67
68
69  int main(int argc, char **argv){
70
71      printf("\n\t\tEnter the center coordinates of the circle\n");
72
73      printf("\nEnter the X coordinate: ");
74      scanf("%d", &x);
75
76      printf("\nEnter the Y coordinate: ");
77      scanf("%d", &y);
78
79      printf("\nEnter the radius of the circle: ");
80      scanf("%d", &radius);
81
82      glutInit(&argc, argv);
83      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
84      glutInitWindowPosition(100, 100);
85      glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
86      glutCreateWindow("Mid-Point Circle Drawing Algorithm");
87
88      initializeDisplay();
89      glutDisplayFunc(drawCircle);
90      glutMainLoop();
91
92      return 1;
93  }
94
95  void initializeDisplay(){
96      //Initialize the display parameters
97
```

```
98      glClearColor(0, 1, 1, 0);       //Display window color
99      glMatrixMode(GL_PROJECTION);    //Choose projection
100     gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HEIGHT);     //Set
        transformation
101 }
102
103 void setPixel(GLint xCoord, GLint yCoord){
104     //Draw a pixel at the given point
105
106     glBegin(GL_POINTS);
107     glVertex2i(xCoord, yCoord);
108     glEnd();
109 }
110
111 void circleMidPoint(GLint xc, GLint yc, GLint radius){
112     //Implementation of the midpoint circle drawing algorithm
113
114     screenPoint circlePoint;
115
116     GLint p = 1 - radius;   //Initial value for midpoint parameter
117
118     circlePoint.setCoords(0, radius);   //Set coordinates at top of circle
119
120     circlePlotPoints(xc, yc, circlePoint);  //Plot initial point
121     //circlePoint.printCoords();
122
123     //Calculate the next points while X < Y
124     while(circlePoint.getX() < circlePoint.getY()){
125         circlePoint.incX();
126
127         if(p < 0){
128             p += 2 * circlePoint.getX() + 1;
129         }
130
131         else{
132             circlePoint.decY();
133             p += 2 * (circlePoint.getX() - circlePoint.getY()) + 1;
134         }
135
136         //circlePoint.printCoords();
137         circlePlotPoints(xc, yc, circlePoint);
138     }
139
140     glFlush();  //Flush the completed circle output to display
141 }
142
143 void circlePlotPoints(GLint xc, GLint yc, screenPoint circlePoint){
144     //Plot points for all 8 octants of the circle
145
146     setPixel(xc + circlePoint.getX(), yc + circlePoint.getY());
147     setPixel(xc - circlePoint.getX(), yc + circlePoint.getY());
```

```
148     setPixel(xc + circlePoint.getX(), yc - circlePoint.getY());
149     setPixel(xc - circlePoint.getX(), yc - circlePoint.getY());
150     setPixel(xc + circlePoint.getY(), yc + circlePoint.getX());
151     setPixel(xc - circlePoint.getY(), yc + circlePoint.getX());
152     setPixel(xc + circlePoint.getY(), yc - circlePoint.getX());
153     setPixel(xc - circlePoint.getY(), yc - circlePoint.getX());
154 }
155
156 void drawCircle(){
157     //Driver function to call the circle drawing function
158
159     circleMidPoint(x, y, radius);
160 }
```

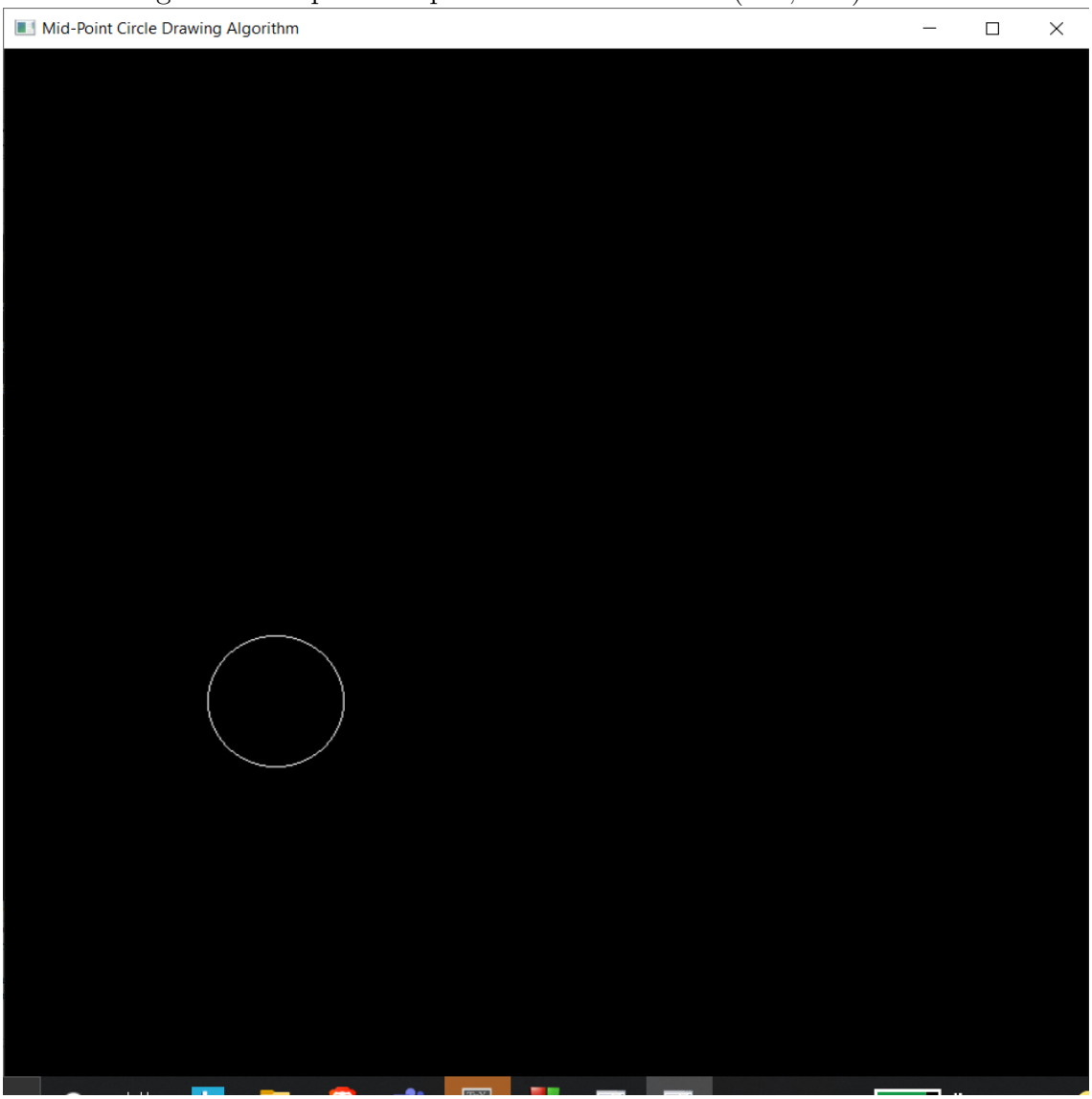
Output: Midpoint Circle Case 1 - C(400, 400) R - 200:

Figure 1: Output: Midpoint Circle Case 1 - C(400, 400) R - 200.



Output: Midpoint Circle Case 2 - C(200, 300) R - 50:

Figure 2: Output: Midpoint Circle Case 2 - C(200, 300) R - 50.



Code: Object Using Circles and Lines:

```
/* To draw an object with circles and lines - Stick man */

#include <windows.h>
#include <stdio.h>
#include <GL/glut.h>

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 800;

class screenPoint{
    /* Class for a coordinate point */

private:
    GLint x, y;

public:
    screenPoint(){
        //Default constructor, initialize to (0, 0)
        x = y = 0;
    }

    screenPoint(GLint xCoord, GLint yCoord){
        //Constructor with preset points
        x = xCoord;
        y = yCoord;
    }

    void setCoords(GLint xCoord, GLint yCoord){
        //Function to set coordinates
        x = xCoord;
        y = yCoord;
    }

    GLint getX() const{
        //Return x coordinate
        return x;
    }

    GLint getY() const{
        //Return y coordinate
        return y;
    }

    void incX(){
        //Increment x coordinate
        x++;
    }
```

```
48
49      void decY(){
50          //Decrement y coordinate
51          y--;
52      }
53
54      void printCoords(){
55          //Print current coordinates
56          printf("\nX: %d\tY: %d", x, y);
57      }
58  };
59
60  void drawStickman();
61  void initializeDisplay();
62  void setPixel(GLint xCoord, GLint yCoord);
63  void drawCircle(GLint xc, GLint yc, GLint radius);
64  void drawLine(GLint x1, GLint y1, GLint x2, GLint y2);
65  void circlePlotPoints(GLint xc, GLint yc, screenPoint circlePoint);
66
67
68  int main(int argc, char **argv){
69
70      glutInit(&argc, argv);
71      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
72      glutInitWindowPosition(100, 100);
73      glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
74      glutCreateWindow("Stickman using Circles & Lines");
75
76      initializeDisplay();
77      glutDisplayFunc(drawStickman);
78      glutMainLoop();
79
80      return 1;
81  }
82
83  void initializeDisplay(){
84      //Initialize the display parameters
85
86      glClearColor(0, 1, 1, 0);       //Display window color
87      glMatrixMode(GL_PROJECTION);    //Choose projection
88      gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HEIGHT);     //Set
        transformation
89  }
90
91  void setPixel(GLint xCoord, GLint yCoord){
92      //Draw a pixel at the given point
93
94      glBegin(GL_POINTS);
95      glVertex2i(xCoord, yCoord);
96      glEnd();
97  }
```

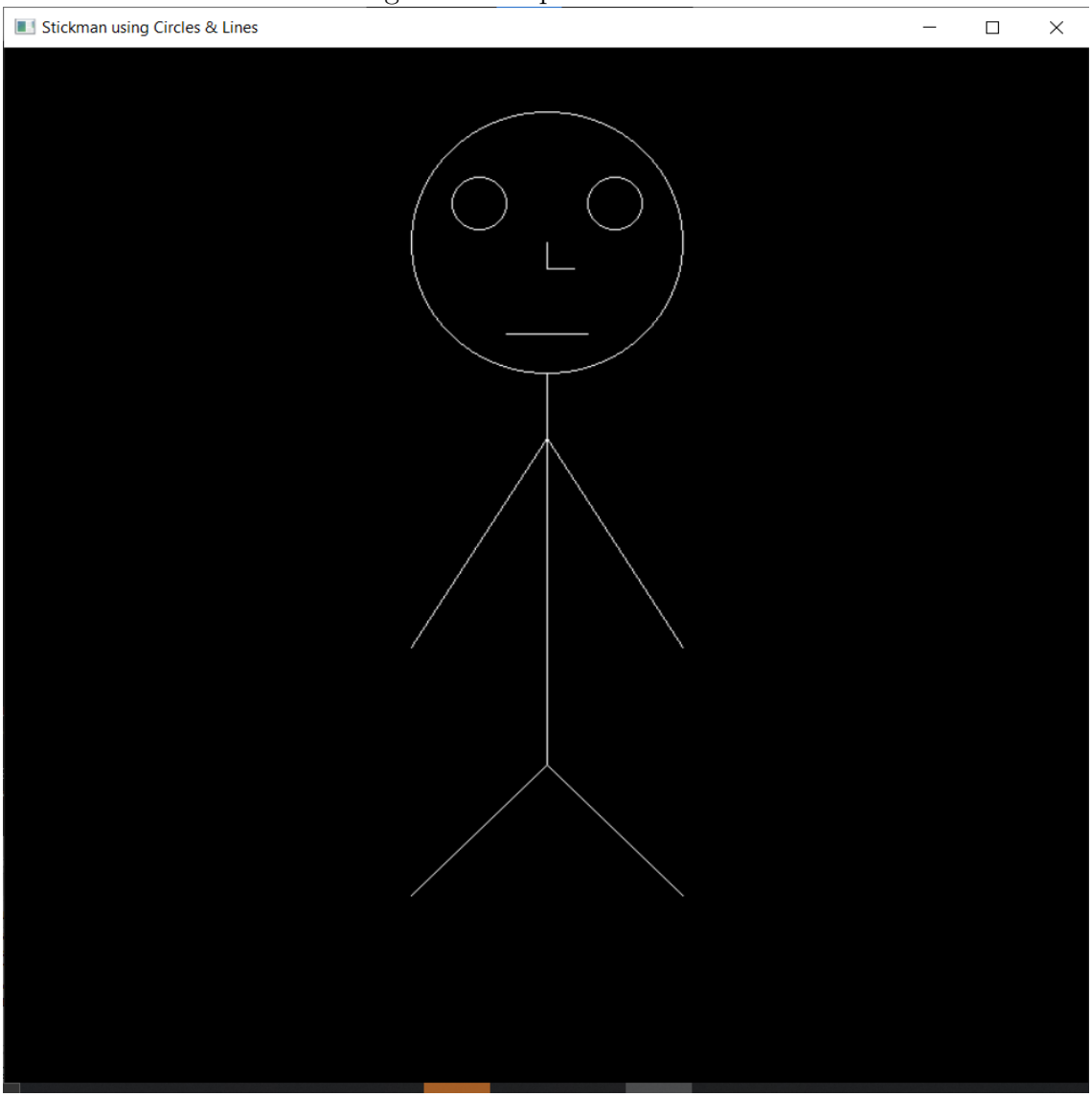
```
98
99  void drawCircle(GLint xc, GLint yc, GLint radius){
100     //Implementation of the midpoint circle drawing algorithm
101
102     screenPoint circlePoint;
103
104     GLint p = 1 - radius;   //Initial value for midpoint parameter
105
106     circlePoint.setCoords(0, radius);   //Set coordinates at top of circle
107
108     circlePlotPoints(xc, yc, circlePoint);  //Plot initial point
109     //circlePoint.printCoords();
110
111     //Calculate the next points while X < Y
112     while(circlePoint.getX() < circlePoint.getY()){
113         circlePoint.incX();
114
115         if(p < 0){
116             p += 2 * circlePoint.getX() + 1;
117         }
118
119         else{
120             circlePoint.decY();
121             p += 2 * (circlePoint.getX() - circlePoint.getY()) + 1;
122         }
123
124         //circlePoint.printCoords();
125         circlePlotPoints(xc, yc, circlePoint);
126     }
127
128     glFlush();  //Flush the completed circle output to display
129 }
130
131 void circlePlotPoints(GLint xc, GLint yc, screenPoint circlePoint){
132     //Plot points for all 8 octants of the circle
133
134     setPixel(xc + circlePoint.getX(), yc + circlePoint.getY());
135     setPixel(xc - circlePoint.getX(), yc + circlePoint.getY());
136     setPixel(xc + circlePoint.getX(), yc - circlePoint.getY());
137     setPixel(xc - circlePoint.getX(), yc - circlePoint.getY());
138     setPixel(xc + circlePoint.getY(), yc + circlePoint.getX());
139     setPixel(xc - circlePoint.getY(), yc + circlePoint.getX());
140     setPixel(xc + circlePoint.getY(), yc - circlePoint.getX());
141     setPixel(xc - circlePoint.getY(), yc - circlePoint.getX());
142 }
143
144 void drawLine(GLint x1, GLint y1, GLint x2, GLint y2){
145     //Bresenham's Line Drawing Algorithm Implementation
146
147     GLint dx, dy, x, y, p, inc_x1, inc_y1, inc_p1, inc_p2;
148     GLint x_sign, y_sign;
```

```
149
150     dx = x2 - x1;
151     dy = y2 - y1;
152
153     //note the sign for directionality
154     x_sign = dx/abs(dx);
155     y_sign = dy/abs(dy);
156
157     //increment is by 1, and in the direction of +/-
158     inc_x1 = 1 * x_sign;
159     inc_y1 = 1 * y_sign;
160
161     //change the differences to absolute values (crucial step)
162     dx = abs(dx);
163     dy = abs(dy);
164
165     //initial coordinates
166     x = x1;
167     y = y1;
168
169     glBegin(GL_POINTS);
170
171     if(abs(dx) > abs(dy)){
172         //X difference > Y difference
173         glVertex2i(x, y);
174
175         p = (2 * dy) - dx;
176         inc_p1 = 2 * (dy - dx);
177         inc_p2 = 2 * dy;
178
179         //plot for dx number of points
180         for(GLint i = 0; i < dx; i++){
181             if(p >= 0){
182                 y += inc_y1;
183                 p += inc_p1;
184             }
185             else{
186                 p += inc_p2;
187             }
188
189             x += inc_x1;
190
191             glVertex2i(x, y);
192         }
193     }
194     else{
195         //X difference <= Y difference
196         glVertex2i(x, y);
197
198         p = (2 * dx) - dy;
199         inc_p1 = 2 * (dx - dy);
```

```
        inc_p2 = 2 * dx;

        //plot for dy number of points
        for(GLint i = 0; i < dy; i++){
            if(p >= 0){
                x += inc_x1;
                p += inc_p1;
            }
            else{
                p += inc_p2;
            }

            y += inc_y1;

            glVertex2i(x, y);
        }
    }

    glEnd();
    glFlush();
}


void drawStickman(){
    //Draw a stick man using the drawLine() and drawCircle() algorithms

    //Face
    drawCircle(400, 650, 100);

    //Eyes
    drawCircle(350, 680, 20);
    drawCircle(450, 680, 20);

    //Nose
    drawLine(400, 650, 400, 630);
    drawLine(400, 630, 420, 630);

    //Mouth
    drawLine(370, 580, 430, 580);

    //Body
    drawLine(400, 550, 400, 250);

    //Arms
    drawLine(400, 500, 300, 340);
    drawLine(400, 500, 500, 340);

    //Legs
    drawLine(400, 250, 300, 150);
    drawLine(400, 250, 500, 150);
}
```

Output: Stickman:

Figure 3: Output: Stickman.



Learning Outcome:

- I understood the **Midpoint Circle Drawing Algorithm**'s working.
- I implemented the Midpoint Circle Drawing algorithm using an OpenGL program.
- I understood how the algorithm makes use of **octant symmetry** and plots points on the second octant and mirrors it to other octants.
- I understood how the parameter **p** is calculated in each iteration to determine the decrement of Y coordinate.
- I learnt about **classes and member functions** in C++ to implement methods to facilitate the circle drawing algorithm process.
- I was able to output all different test cases appropriately to verify the correctness of my program to implement the Midpoint Circle Drawing Algorithm.
- I was able to draw and output a **Stickman** using the **Midpoint Circle Drawing Algorithm** and the **Bresenham's Line Drawing Algorithm**.

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

25 August 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 5: 2D Transformations in C++ using OpenGL

Aim:

To apply the following 2D transformations on objects and to render the final output along with the original object.

- Translation
- Rotation
  - About Origin
  - With Respect to a fixed point (x_r, y_r)
- Scaling with respect to
  - Origin - Uniform vs. Differential Scaling
  - Fixed Point (x_f, y_f)

- Reflection with respect to
  - X - Axis
  - Y - Axis
  - Origin
  - The Line $X = Y$
- Shearing
  - X - Direction Shear
  - Y - Direction Shear

Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants.
(Use LINES primitive to draw X & Y Axes)

Code: 2D Transformations:

```
//To perform 2D Transformations on objects and to render the final output along with the original object
//Translation, Rotation, Reflection, Scaling, Shearing

#include <windows.h>
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 800;
const int X_MIN = -400;
const int X_MAX = 400;
const int Y_MIN = -400;
const int Y_MAX = 400;

enum Axes {xAxis, yAxis};
enum Lines {XAxis, YAxis, Origin, XEqualsY};
enum Transformations {  Translation = 1, Rotation = 2, RotationAboutPivot = 3, ReflectAboutX = 4,
                        ReflectAboutY = 5, ReflectAbout0 = 6, ReflectAboutXEqY = 7, UniformScale = 8,
                        DifferentialScale = 9, ScaleAboutFixed = 10, ShearAboutX = 11, ShearAboutY = 12,
                        ShearAboutXRef = 13, ShearAboutYRef = 14, ClearTransforms = 15, ChangePolygon = 16, Refresh = 17};

class Point{
private:
    GLdouble x, y, h;

public:
    Point(){
        x = y = 0;
        h = 1;
    }

    Point(GLint xCoord, GLint yCoord){
        x = xCoord;
        y = yCoord;
        h = 1;
    }

    Point(GLint xCoord, GLint yCoord, GLint H){
        x = xCoord;
        y = yCoord;
        h = H;
```

```
43      }
44
45      void setCoords(GLint xCoord, GLint yCoord){
46          x = xCoord;
47          y = yCoord;
48      }
49
50      GLdouble getX() const{
51          return x;
52      }
53
54      GLdouble getY() const{
55          return y;
56      }
57
58      GLdouble getH() const{
59          return h;
60      }
61
62      GLdouble getHomogenousX() const{
63          return x * h;
64
65      }
66      GLdouble getHomogenousY() const{
67          return y * h;
68      }
69
70      Point getTranslatedPoint(Point translationVector){
71          //For 2D Translation about a given translation vector
72
73          double translationMatrix[3][3] = {  {1, 0, translationVector.
    getHomogenousX()},
74                                              {0, 1, translationVector.
    getHomogenousY()},
75                                              {0, 0, 1}};
76
77          double values[3];
78
79          for(int i = 0; i < 3; i++){
80              values[i] = translationMatrix[i][0] * getHomogenousX() +
81                          translationMatrix[i][1] * getHomogenousY() +
82                          translationMatrix[i][2] * getH();
83          }
84
85          return Point(values[0]/h, values[1]/h, values[2]);
86      }
87
88      Point getRotatedPoint(int rotationAngle, Point pivot = Point(0, 0, 1))
    {
89          //For 2D Rotation about a given pivot by a given rotation angle
90
```

```
91          double rotationAngleInRadians = rotationAngle * 3.14159/180;
92          double cosAngle = cos(rotationAngleInRadians);
93          double sinAngle = sin(rotationAngleInRadians);
94
95          double xPivotValue = (pivot.getX() * (1 - cosAngle)) + (pivot.getY
        () * sinAngle);
96          double yPivotValue = (pivot.getY() * (1 - cosAngle)) - (pivot.getX
        () * sinAngle);
97
98          double rotationMatrix[3][3] = { {cosAngle, -sinAngle, xPivotValue
        },
99                                          {sinAngle, cosAngle, yPivotValue},
100                                         {0, 0, 1}};
101
102         double values[3];
103
104         for(int i = 0; i < 3; i++){
105             values[i] = rotationMatrix[i][0] * getHomogenousX() +
106                         rotationMatrix[i][1] * getHomogenousY() +
107                         rotationMatrix[i][2] * getH();
108         }
109
110         return Point(values[0]/h, values[1]/h, values[2]);
111     }
112
113     Point getReflectionAboutXAxis(){
114         //For 2D Reflection about the X axis
115
116         double reflectionMatrix[3][3] = {   {1, 0, 0},
117                                             {0, -1, 0},
118                                             {0, 0, 1}};
119
120         double values[3];
121
122         for(int i = 0; i < 3; i++){
123             values[i] = reflectionMatrix[i][0] * getHomogenousX() +
124                         reflectionMatrix[i][1] * getHomogenousY() +
125                         reflectionMatrix[i][2] * getH();
126         }
127
128         return Point(values[0]/h, values[1]/h, values[2]);
129     }
130
131     Point getReflectionAboutYAxis(){
132         //For 2D Reflection about the Y axis
133
134         double reflectionMatrix[3][3] = {   {-1, 0, 0},
135                                             {0, 1, 0},
136                                             {0, 0, 1}};
137
138         double values[3];
```

```
139
140         for(int i = 0; i < 3; i++){
141             values[i] = reflectionMatrix[i][0] * getHomogenousX() +
142                         reflectionMatrix[i][1] * getHomogenousY() +
143                         reflectionMatrix[i][2] * getH();
144         }
145
146         return Point(values[0]/h, values[1]/h, values[2]);
147     }
148
149     Point getReflectionAboutOrigin(){
150         //For 2D Reflection about the Origin
151
152         double reflectionMatrix[3][3] = {   {-1, 0, 0},
153                                             {0, -1, 0},
154                                             {0, 0, 1}};
155
156         double values[3];
157
158         for(int i = 0; i < 3; i++){
159             values[i] = reflectionMatrix[i][0] * getHomogenousX() +
160                         reflectionMatrix[i][1] * getHomogenousY() +
161                         reflectionMatrix[i][2] * getH();
162         }
163
164         return Point(values[0]/h, values[1]/h, values[2]);
165     }
166
167     Point getReflectionAboutXEqualsY(){
168         //For 2D Reflection about the line X=Y
169
170         double reflectionMatrix[3][3] = {   {0, 1, 0},
171                                             {1, 0, 0},
172                                             {0, 0, 1}};
173
174         double values[3];
175
176         for(int i = 0; i < 3; i++){
177             values[i] = reflectionMatrix[i][0] * getHomogenousX() +
178                         reflectionMatrix[i][1] * getHomogenousY() +
179                         reflectionMatrix[i][2] * getH();
180         }
181
182         return Point(values[0]/h, values[1]/h, values[2]);
183     }
184
185     Point getScaledPoint(double ScaleX, double ScaleY, Point fixed){
186         //For 2D Scaling about a fixed point and scale factors for X & Y
    axes
187
188         double xFixedValue = fixed.getX() * (1 - ScaleX);
```

```
189         double yFixedValue = fixed.getY() * (1 - ScaleY);
190 
191         double scalingMatrix[3][3] = {  {ScaleX, 0, xFixedValue},
192                                         {0, ScaleY, yFixedValue},
193                                         {0, 0, 1}};
194 
195         double values[3];
196 
197         for(int i = 0; i < 3; i++){
198             values[i] = scalingMatrix[i][0] * getHomogenousX() +
199                         scalingMatrix[i][1] * getHomogenousY() +
200                         scalingMatrix[i][2] * getH();
201         }
202 
203         return Point(values[0]/h, values[1]/h, values[2]);
204     }
205 
206     Point getShearAboutXAxis(double shearParam, double yRefLine = 0){
207         //For shearing about X axis
208 
209         double shearMatrix[3][3] = {{1, shearParam, -shearParam * yRefLine
    },
210                                     {0, 1, 0},
211                                     {0, 0, 1}};
212 
213         double values[3];
214 
215         for(int i = 0; i < 3; i++){
216             values[i] = shearMatrix[i][0] * getHomogenousX() +
217                         shearMatrix[i][1] * getHomogenousY() +
218                         shearMatrix[i][2] * getH();
219         }
220 
221         return Point(values[0]/h, values[1]/h, values[2]);
222     }
223 
224     Point getShearAboutYAxis(double shearParam, double xRefLine = 0){
225         //For shearing about Y axis
226 
227         double shearMatrix[3][3] = {{1, 0, -shearParam * xRefLine},
228                                     {shearParam, 1, 0},
229                                     {0, 0, 1}};
230 
231         double values[3];
232 
233         for(int i = 0; i < 3; i++){
234             values[i] = shearMatrix[i][0] * getHomogenousX() +
235                         shearMatrix[i][1] * getHomogenousY() +
236                         shearMatrix[i][2] * getH();
237         }
238 
```

```
239         return Point(values[0]/h, values[1]/h, values[2]);
240     }
241 };
242 
243 
244 class PolygonShape{
245 private:
246     int numVertices;
247     Point *points;
248 
249 public:
250     PolygonShape(){
251         numVertices = 0;
252     }
253 
254     PolygonShape(int noVertices){
255         numVertices = noVertices;
256         points = new Point[numVertices];
257     }
258 
259     int getVertexCount() const{
260         return numVertices;
261     }
262 
263     Point getPoint(int i){
264         return points[i];
265     }
266 
267     void setVertices(int noVertices){
268         numVertices = noVertices;
269         points = new Point[numVertices];
270     }
271 
272     void setPoint(int i, GLint x, GLint y){
273         points[i].setCoords(x, y);
274     }
275 };
276 
277 
278 void initializeDisplay();
279 void plotComponents();
280 void dummyFunction();
281 void dummyKeyFunction(unsigned char key, int x, int y);
282 void transformationMenu(int option);
283 void plotTransformation();
284 void drawAxes();
285 void drawPolygon(PolygonShape polygon);
286 void translatePolygon(PolygonShape polygon, Point translationVector);
287 void reflectPolygon(PolygonShape polygon, Lines line);
288 void rotatePolygon(PolygonShape polygon, int rotationAngle, Point pivot =
      Point(0, 0, 1));
```

```
289 void scalePolygon(PolygonShape polygon, double scaleX, double scaleY,
      Point fixed = Point(0, 0, 1));
290 void shearPolygon(PolygonShape polygon, Axes axis, double shearParam,
      double refLine = 0);
291
292 PolygonShape polygon;           //Global PolygonShape object to be plotted
      on the graph
293 int chosenTransformation = 0;   //Global variable to keep track of chosen
      transformation
294
295 int main(int argc, char **argv){
296     glutInit(&argc, argv);
297     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
298     glutInitWindowPosition(0, 0);
299     glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
300     glutCreateWindow("2D Transformations - Examples");
301
302     printf("\n------[2D TRANSFORMATIONS]------\n");
303     printf("\nUsage:\tRight-Click the GLUT Window to select a
      transformation.");
304     printf("\n\tProvide input for the necessary parameters in this window.
      ");
305     printf("\n\tRefresh the output window if it becomes unresponsive
      during console I/O.");
306     printf("\n\n------[2D TRANSFORMATIONS]------\n\n");
307
308     //Set the initial default polygon for the graph
309     polygon.setVertices(4);
310     polygon.setPoint(0, 0, 0);
311     polygon.setPoint(1, 0, 50);
312     polygon.setPoint(2, 100, 50);
313     polygon.setPoint(3, 100, 0);
314
315     initializeDisplay();
316     glutDisplayFunc(dummyFunction);
317     plotComponents();
318
319     glutCreateMenu(transformationMenu);
320     glutAddMenuEntry("Translation", 1);
321     glutAddMenuEntry("Rotation", 2);
322     glutAddMenuEntry("Rotation About Pivot Point", 3);
323     glutAddMenuEntry("Reflection About X Axis", 4);
324     glutAddMenuEntry("Reflection About Y Axis", 5);
325     glutAddMenuEntry("Reflection About Origin", 6);
326     glutAddMenuEntry("Reflection About X = Y", 7);
327     glutAddMenuEntry("Uniform Scaling", 8);
328     glutAddMenuEntry("Differential Scaling", 9);
329     glutAddMenuEntry("Scaling About Fixed Point", 10);
330     glutAddMenuEntry("Shear About X Axis", 11);
331     glutAddMenuEntry("Shear About Y Axis", 12);
332     glutAddMenuEntry("Shear About X Axis About Y = y", 13);
```

```
333     glutAddMenuEntry("Shear About Y Axis About X = x", 14);
334     glutAddMenuEntry("Clear Transformations", 15);
335     glutAddMenuEntry("Change Polygon", 16);
336     glutAddMenuEntry("Refresh Screen", 17);
337     glutAttachMenu(GLUT_RIGHT_BUTTON);
338 
339     glutMainLoop();
340 
341     return 1;
342 }
343 
344 void transformationMenu(int option){
345     chosenTransformation = option;
346     plotTransformation();
347 }
348 
349 void initializeDisplay(){
350     //Initialize the display parameters
351 
352     glClearColor(1, 1, 1, 0);
353     glMatrixMode(GL_PROJECTION);
354     gluOrtho2D(X_MIN, X_MAX, Y_MIN, Y_MAX);
355     glClear(GL_COLOR_BUFFER_BIT);   //Clear the display window
356 
357     glEnable(GL_BLEND);     //enable blending (translucent colors)
358     glDepthMask(GL_FALSE);
359     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);  //set the blend
    function for translucency
360 }
361 
362 void plotComponents(){
363     //Plot the axes and the base polygon
364 
365     glClear(GL_COLOR_BUFFER_BIT);   //Clear the display window
366     drawAxes();
367     drawPolygon(polygon);
368     glFlush();
369 }
370 
371 void dummyFunction(){
372     //Placeholder function to be called in glutDisplayFunc
373 }
374 
375 void plotTransformation(){
376     //Plot the specified transformation
377 
378     switch(chosenTransformation){
379         case Translation:{
380             double x, y;
381             printf("\n\n------[TRANSLATION]------\n");
382             printf("\n\tEnter the Translation Vector Magnitudes: ");
```

```
383
384             printf("\n\t\tX Component: ");
385             scanf("%lf", &x);
386             printf("\n\t\tY Component: ");
387             scanf("%lf", &y);
388             Point translationVector(x, y, 1);
389             translatePolygon(polygon, translationVector);
390
391             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
392             break;
393         }
394
395         case Rotation:{
396             double rotationAngle;
397             printf("\n\n------[ROTATION]------\n");
398             printf("\n\tEnter the Rotation Angle: ");
399
400             scanf("%lf", &rotationAngle);
401             rotatePolygon(polygon, rotationAngle);
402
403             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
404             break;
405         }
406
407         case RotationAboutPivot:{
408             double rotationAngle, x, y;
409             printf("\n\n------[ROTATION ABOUT PIVOT]------\n");
410             printf("\n\tEnter the Rotation Angle: ");
411
412             scanf("%lf", &rotationAngle);
413             printf("\n\tEnter Pivot Point: ");
414             printf("\n\t\tEnter X Coordinate: ");
415             scanf("%lf", &x);
416             printf("\n\t\tEnter Y Coordinate: ");
417             scanf("%lf", &y);
418             rotatePolygon(polygon, rotationAngle, Point(x, y, 1));
419
420             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
421             break;
422         }
423
424         case ReflectAboutX:{
425             printf("\n\n------[REFLECTION ABOUT X AXIS]------\n");
426
427             reflectPolygon(polygon, XAxis);
428
429             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
430             break;
431         }
432
433         case ReflectAboutY:{
```

```
434             printf("\n\n------[REFLECTION ABOUT Y AXIS]------\n");
435
436             reflectPolygon(polygon, YAxis);
437
438             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
439             break;
440         }
441
442         case ReflectAboutO:{
443             printf("\n\n------[REFLECTION ABOUT ORIGIN]------\n");
444
445             reflectPolygon(polygon, Origin);
446
447             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
448             break;
449         }
450
451         case ReflectAboutXEqY:{
452             printf("\n\n------[REFLECTION ABOUT X = Y]------\n");
453
454             reflectPolygon(polygon, XEqualsY);
455
456             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
457             break;
458         }
459
460         case UniformScale:{
461             double scaleFactor;
462             printf("\n\n------[UNIFORM SCALING]------\n");
463
464             printf("\n\tEnter the Scaling Factors: ");
465             scanf("%lf", &scaleFactor);
466
467             scalePolygon(polygon, scaleFactor, scaleFactor);
468
469             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
470             break;
471         }
472
473         case DifferentialScale:{
474             double xScale, yScale;
475             printf("\n\n------[DIFFERENTIAL SCALING]------\n");
476
477             printf("\n\tEnter the Scaling Factors: ");
478
479             printf("\n\t\tX Scale Factor: ");
480             scanf("%lf", &xScale);
481             printf("\n\t\tY Scale Factor: ");
482             scanf("%lf", &yScale);
483
484             scalePolygon(polygon, xScale, yScale);
```

```
485
486             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
487             break;
488         }
489
490         case ScaleAboutFixed:{
491             double xScale, yScale, xFixed, yFixed;
492             printf("\n\n------[SCALING ABOUT FIXED POINT]------\n");
493
494             printf("\n\tEnter the Scaling Factors: ");
495
496             printf("\n\t\tX Scale Factor: ");
497             scanf("%lf", &xScale);
498             printf("\n\t\tY Scale Factor: ");
499             scanf("%lf", &yScale);
500
501             printf("\n\tEnter the Fixed Point: ");
502
503             printf("\n\t\tX Coordinate: ");
504             scanf("%lf", &xFixed);
505             printf("\n\t\tY Coordinate: ");
506             scanf("%lf", &yFixed);
507
508             scalePolygon(polygon, xScale, yScale, Point(xFixed, yFixed, 1)
    );
509
510             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
511             break;
512         }
513
514         case ShearAboutX:{
515             double shearParam;
516             printf("\n\n------[SHEARING ABOUT X AXIS]------\n");
517
518             printf("\n\tEnter the Shearing Parameter: ");
519             scanf("%lf", &shearParam);
520
521             shearPolygon(polygon, xAxis, shearParam);
522
523             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
524             break;
525         }
526
527         case ShearAboutY:{
528             double shearParam;
529             printf("\n\n------[SHEARING ABOUT Y AXIS]------\n");
530
531             printf("\n\tEnter the Shearing Parameter: ");
532             scanf("%lf", &shearParam);
533
534             shearPolygon(polygon, yAxis, shearParam);
```

```
535
536             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
537             break;
538         }
539
540         case ShearAboutXRef:{
541             double shearParam, yRef;
542             printf("\n\n------[SHEARING ABOUT X AXIS ABOUT REF. LINE Y = y
    ]------\n");
543
544             printf("\n\tEnter the Shearing Parameter: ");
545             scanf("%lf", &shearParam);
546
547             printf("\n\tEnter the Reference Line Constant y (Y = y): ");
548             scanf("%lf", &yRef);
549
550             shearPolygon(polygon, xAxis, shearParam, yRef);
551
552             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
553             break;
554         }
555
556         case ShearAboutYRef:{
557             double shearParam, xRef;
558             printf("\n\n------[SHEARING ABOUT Y AXIS ABOUT REF. LINE X = x
    ]------\n");
559
560             printf("\n\tEnter the Shearing Parameter: ");
561             scanf("%lf", &shearParam);
562
563             printf("\n\tEnter the Reference Line Constant x (X = x): ");
564             scanf("%lf", &xRef);
565
566             shearPolygon(polygon, yAxis, shearParam, xRef);
567
568             printf("\n\n------[TRANSFORMATION COMPLETE]------\n");
569             break;
570         }
571
572         case ClearTransforms:{
573             plotComponents();   //Re plot the base graph
574             break;
575         }
576
577         case ChangePolygon:{
578             int i = 0, vertices = 0;
579             double x, y;
580             printf("\n\n------[CHANGE POLYGON]------\n");
581
582             printf("\n\tEnter the number of vertices: ");
583             scanf("%d", &vertices);
```

```
584
585             polygon.setVertices(vertices);
586
587             while(i < vertices){
588                 printf("\n\tEnter Vertex %d Coordinates:", i+1);
589                 printf("\n\t\tX: ");
590                 scanf("%lf", &x);
591                 printf("\n\t\tY: ");
592                 scanf("%lf", &y);
593                 polygon.setPoint(i, x, y);
594                 i++;
595             }
596
597             plotComponents();   //Re plot the base graph
598
599             printf("\n\n------[POLYGON CHANGED]------\n");
600             break;
601         }
602
603         case Refresh:{
604             //Draw an object off screen to refresh the display buffer
605             glBegin(GL_LINES);
606             glVertex2f(2000, 2000);
607             glVertex2f(2001, 2001);
608
609             glEnd();
610             break;
611         }
612     }
613
614     glFlush();
615     glutPostRedisplay();    //IMPORTANT: To refresh the window with the
    new updated plots
616 }
617
618 void drawAxes(){
619     //To draw the X and Y axes
620
621     glColor3d(0, 0, 0); //Black color
622
623     glBegin(GL_LINES);
624
625     //X-axis
626     glVertex2f(X_MIN, 0);
627     glVertex2f(X_MAX, 0);
628
629     //Y-axis
630     glVertex2f(0, Y_MIN);
631     glVertex2f(0, Y_MAX);
632
633     glEnd();
```

```
}

void drawPolygon(PolygonShape polygon){
    //To draw a given polygon

    glColor3d(1, 0, 0); //Red color

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
        glVertex2f(p.getX(), p.getY());
    }

    glEnd();
}

void translatePolygon(PolygonShape polygon, Point translationVector){
    //To translate a given polygon using the translation vector

    glColor4f(0, 0, 1, 0.6); //Blue color

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
        Point pDash = p.getTranslatedPoint(translationVector);
        glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal
   coordinates
    }

    glEnd();
}

void rotatePolygon(PolygonShape polygon, int rotationAngle, Point pivot){
    //To rotate a given polygon using the rotation angle and pivot point

    //Plot the pivot point
    glColor3d(1, 0, 1); //Purple Color
    glPointSize(5);

    glBegin(GL_POINTS);
    glVertex2f(pivot.getX(), pivot.getY());
    glEnd();

    glColor4f(0, 0, 1, 0.6); //Blue Color

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
```

```
684             Point pDash = p.getRotatedPoint(rotationAngle, pivot);
685             glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal
        coordinates
686         }
687 
688         glEnd();
689     }
690 
691     void reflectPolygon(PolygonShape polygon, Lines line){
692         //To reflect a polygon about a given line
693 
694         //Plot the given line
695         glColor3f(1, 0, 1);
696 
697         glBegin(GL_LINES);
698 
699         switch(line){
700             case XAxis:
701                 glVertex2f(X_MIN, 0);
702                 glVertex2f(X_MAX, 0);
703                 break;
704             case YAxis:
705                 glVertex2f(0, Y_MIN);
706                 glVertex2f(0, Y_MAX);
707                 break;
708             case Origin:
709                 glVertex2f(X_MIN, 0);
710                 glVertex2f(X_MAX, 0);
711                 glVertex2f(0, Y_MIN);
712                 glVertex2f(0, Y_MAX);
713                 break;
714             case XEqualsY:
715                 glVertex2f(X_MIN, Y_MIN);
716                 glVertex2f(X_MAX, Y_MAX);
717                 break;
718             default:
719                 return;
720         }
721 
722         glEnd();
723 
724         glColor4f(0, 0, 1, 0.6); //Blue Color
725 
726         glBegin(GL_POLYGON);
727 
728         for(int i = 0; i < polygon.getVertexCount(); i++){
729             Point p = polygon.getPoint(i);
730             Point pDash;
731 
732             switch(line){
733                 case XAxis:
```

```
                pDash = p.getReflectionAboutXAxis();
                break;
            case YAxis:
                pDash = p.getReflectionAboutYAxis();
                break;
            case Origin:
                pDash = p.getReflectionAboutOrigin();
                break;
            case XEqualsY:
                pDash = p.getReflectionAboutXEqualsY();
                break;
            default:
                return;
        }

        glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal coordinates
    }

    glEnd();
}

void scalePolygon(PolygonShape polygon, double scaleX, double scaleY, Point fixed){
    //To translate a given polygon using the scale factors and fixed point

    //Plot the fixed point
    glColor3d(1, 0, 1); //Purple Color
    glPointSize(5);

    glBegin(GL_POINTS);
    glVertex2f(fixed.getX(), fixed.getY());
    glEnd();

    glColor4f(0, 0, 1, 0.6); //Blue Color

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
        Point pDash = p.getScaledPoint(scaleX, scaleY, fixed);
        glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal coordinates
    }

    glEnd();
}

void shearPolygon(PolygonShape polygon, Axes axis, double shearParam, double refLine){
    //To shear a polygon about axis and shear parameter
```

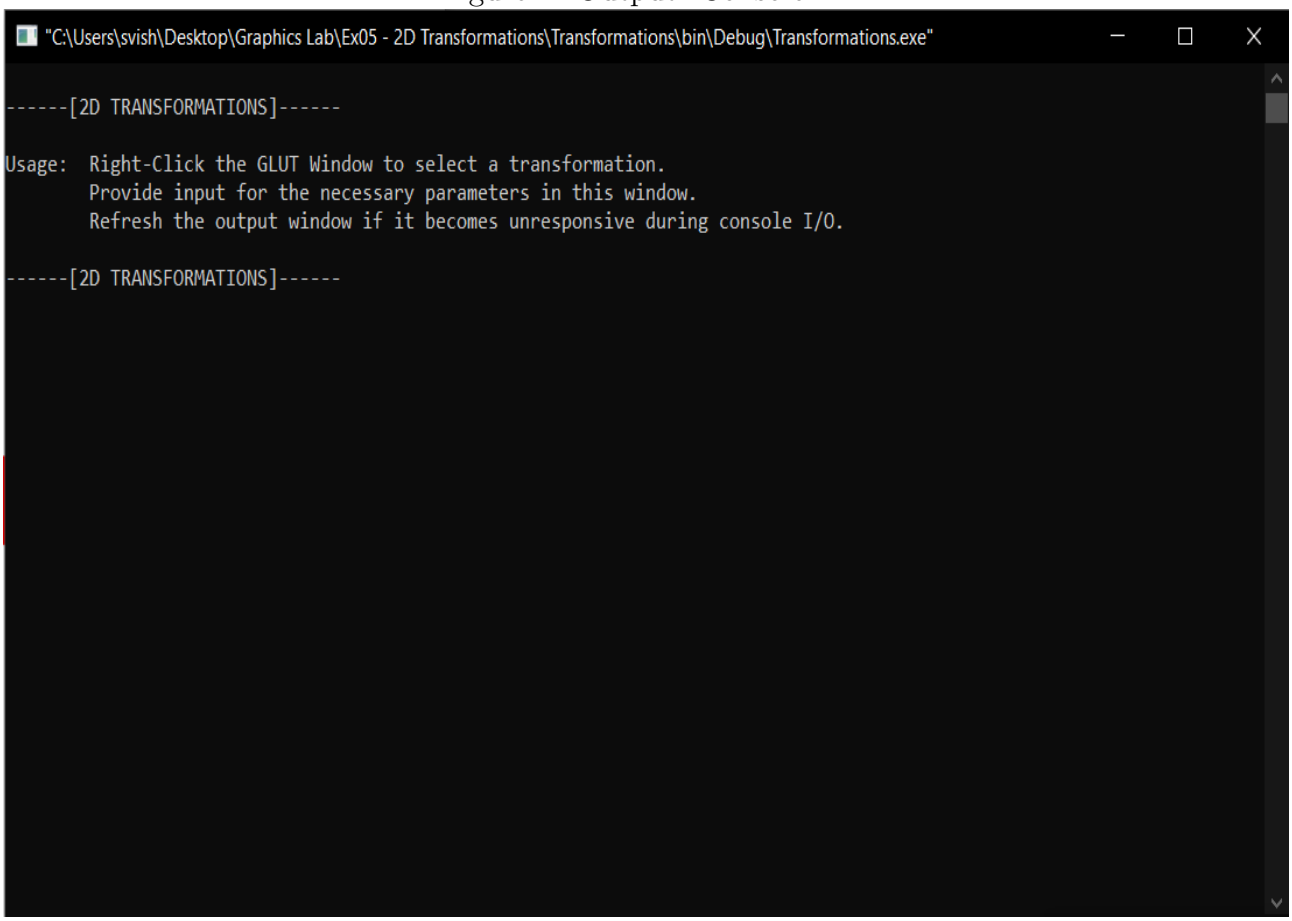
```
781
782     //Plot the given line
783     glColor3f(1, 0, 1);
784
785     glBegin(GL_LINES);
786
787     switch(axis){
788         case xAxis:
789             glVertex2f(X_MIN, 0);
790             glVertex2f(X_MAX, 0);
791             break;
792         case yAxis:
793             glVertex2f(0, Y_MIN);
794             glVertex2f(0, Y_MAX);
795             break;
796         default:
797             return;
798     }
799
800     glEnd();
801
802     glColor4f(0, 0, 1, 0.6); //Blue Color, with alpha (transparency)
    factor as 0.6
803
804     glBegin(GL_POLYGON);
805
806     for(int i = 0; i < polygon.getVertexCount(); i++){
807         Point p = polygon.getPoint(i);
808         Point pDash;
809
810         switch(axis){
811             case xAxis:
812                 pDash = p.getShearAboutXAxis(shearParam, refLine);
813                 break;
814             case yAxis:
815                 pDash = p.getShearAboutYAxis(shearParam, refLine);
816                 break;
817             default:
818                 return;
819         }
820
821         glVertex2f(pDash.getX(), pDash.getY()); //Plot the normal
    coordinates
822     }
823
824     glEnd();
825 }
```

Output: Initial Plot With Transformations Menu

Figure 1: Initial Plot With Transformations Menu.

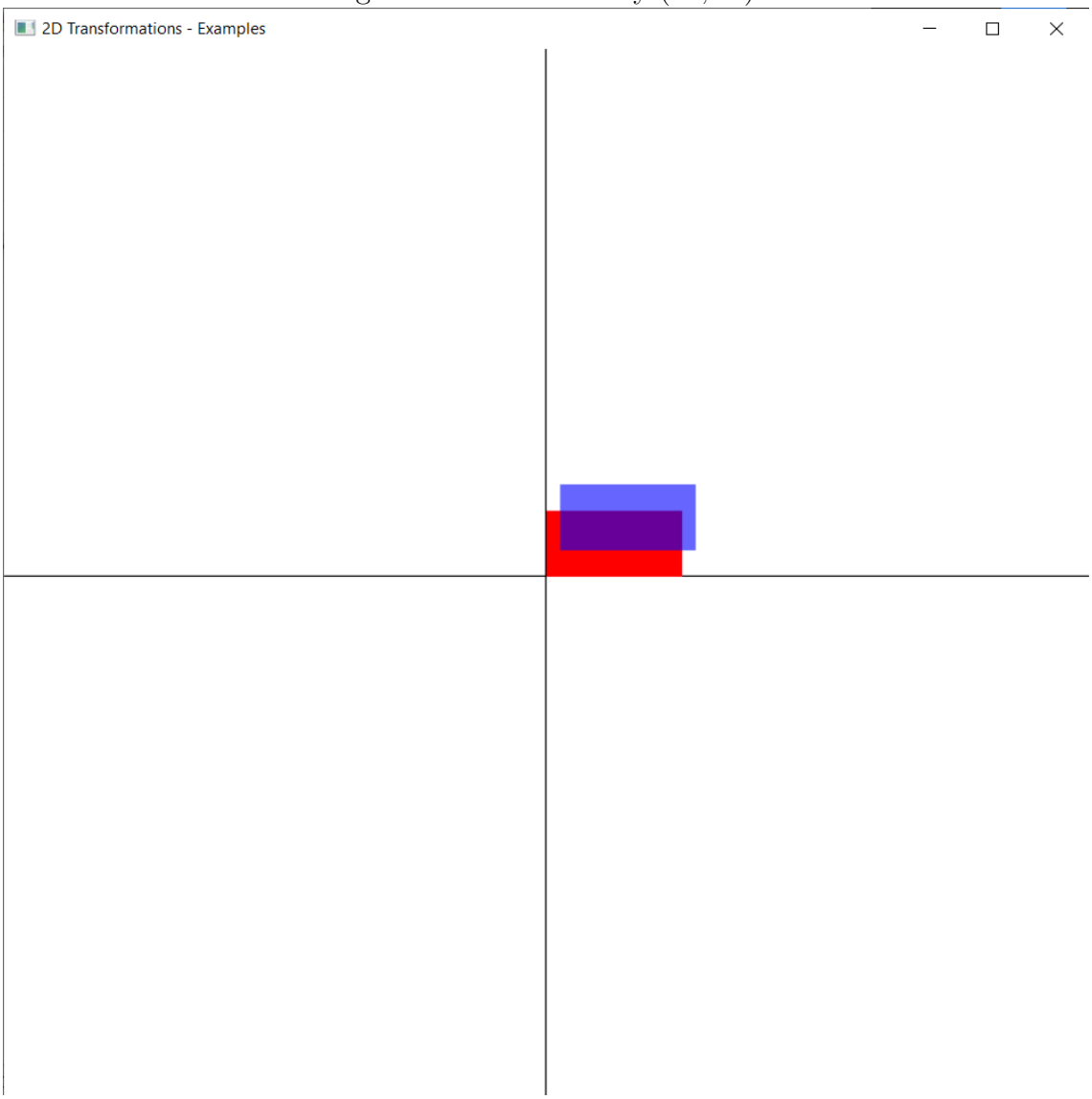
Output: Console

Figure 2: Output: Console.



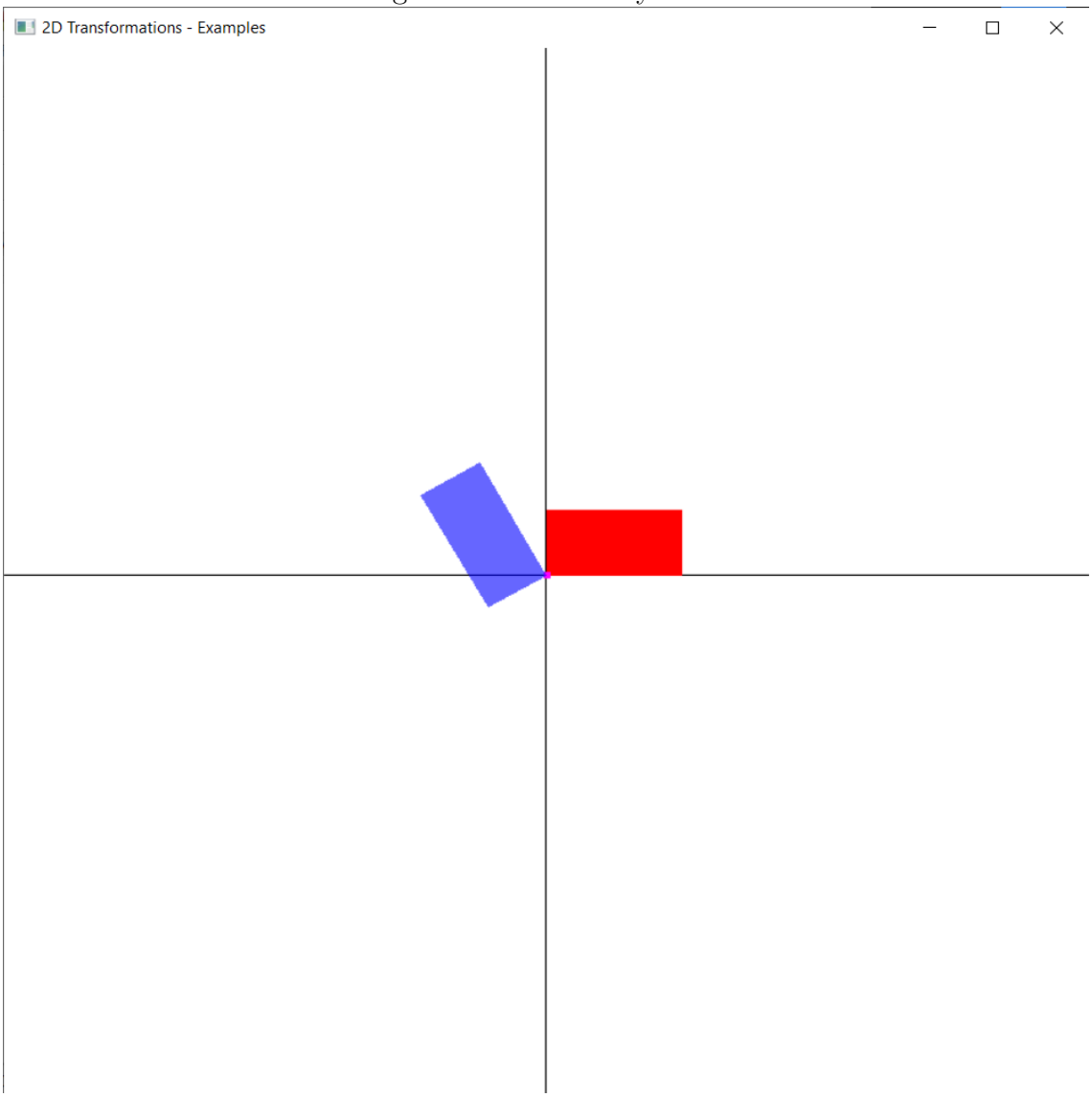
Output: Translation

Figure 3: Translation by $(10, 20)$.



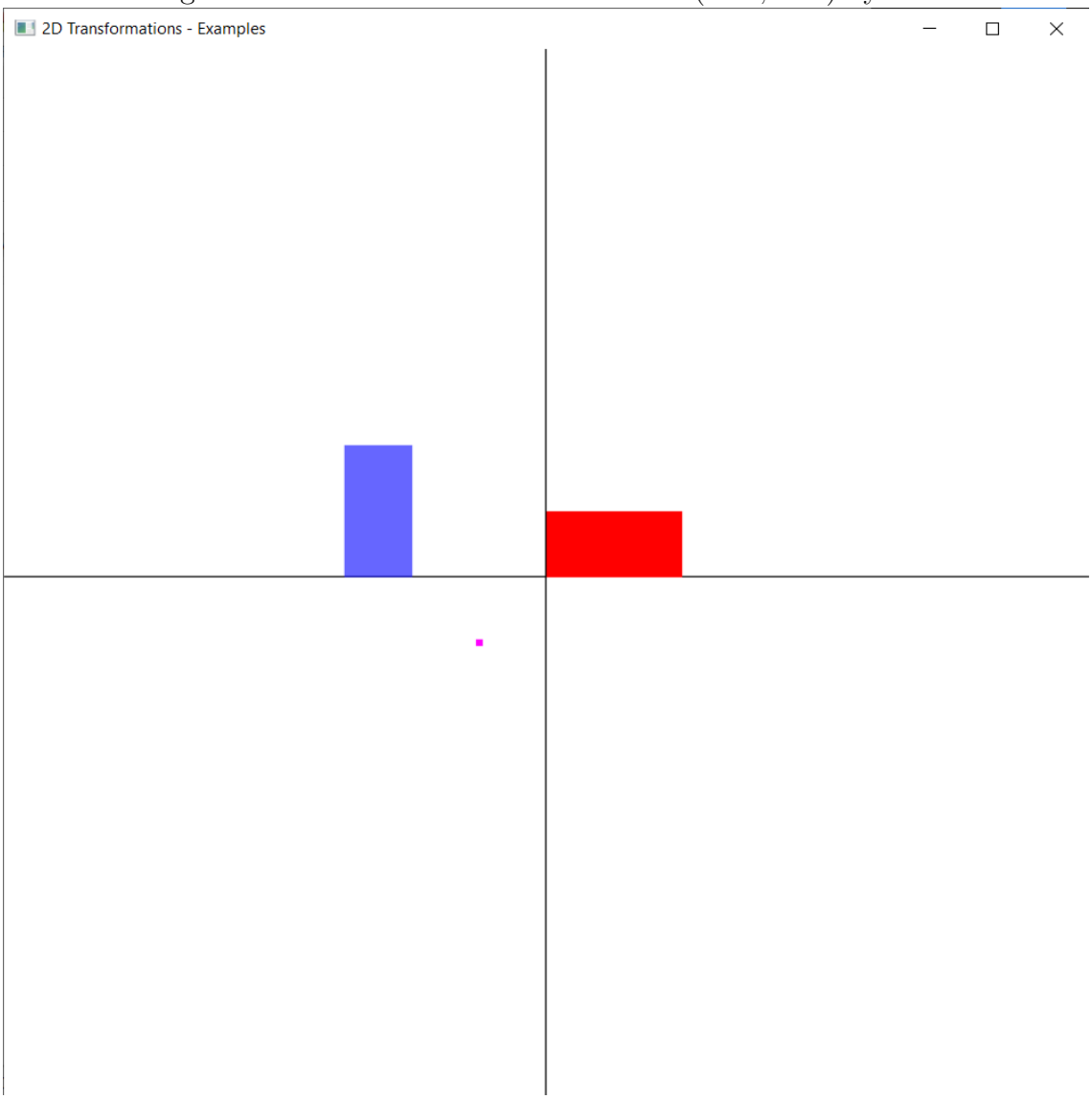
Output: Rotation

Figure 4: Rotation by 120°.



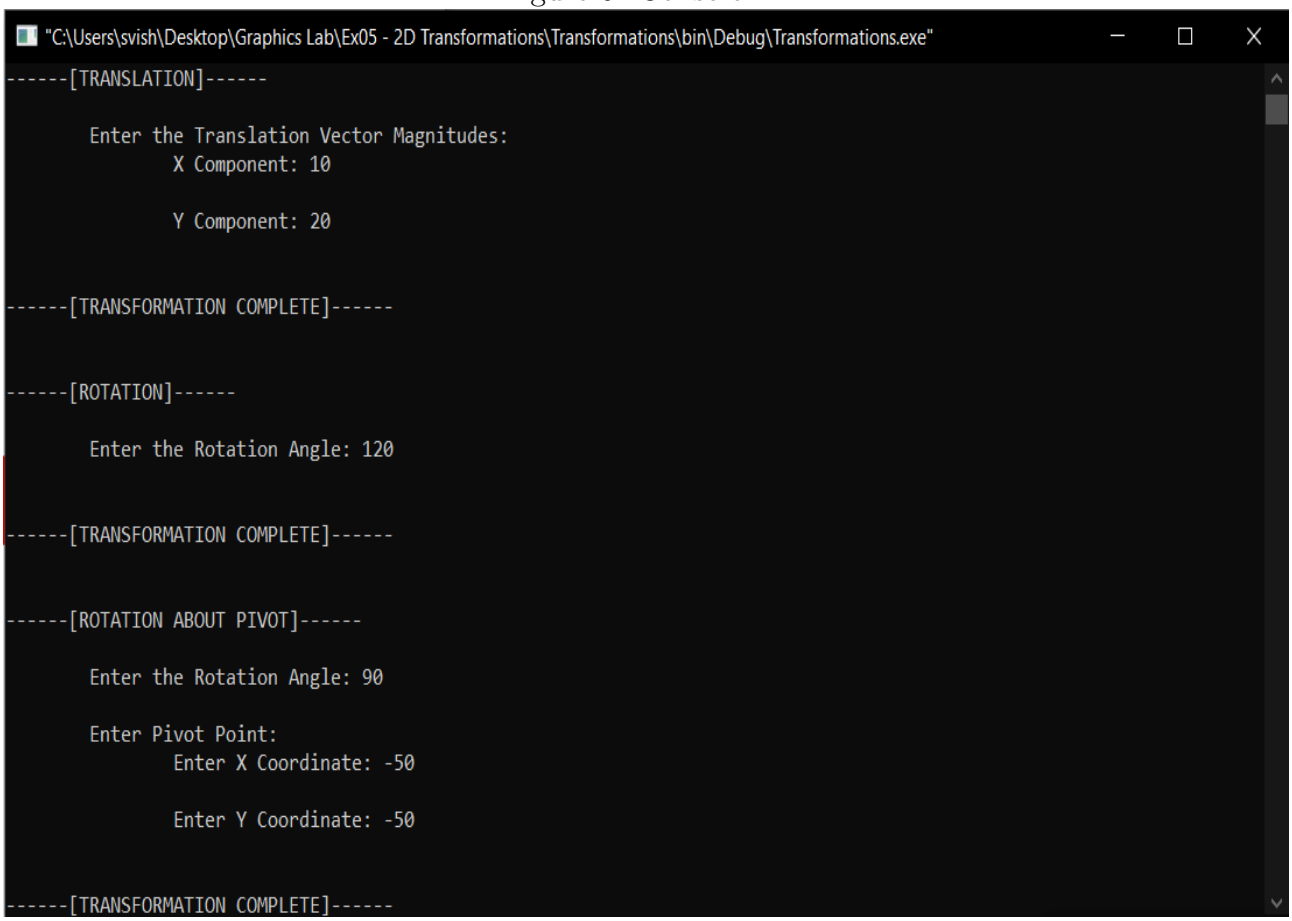
Output: Rotation About A Pivot Point

Figure 5: Rotation About A Pivot Point $(-50, -50)$ by $90°$.



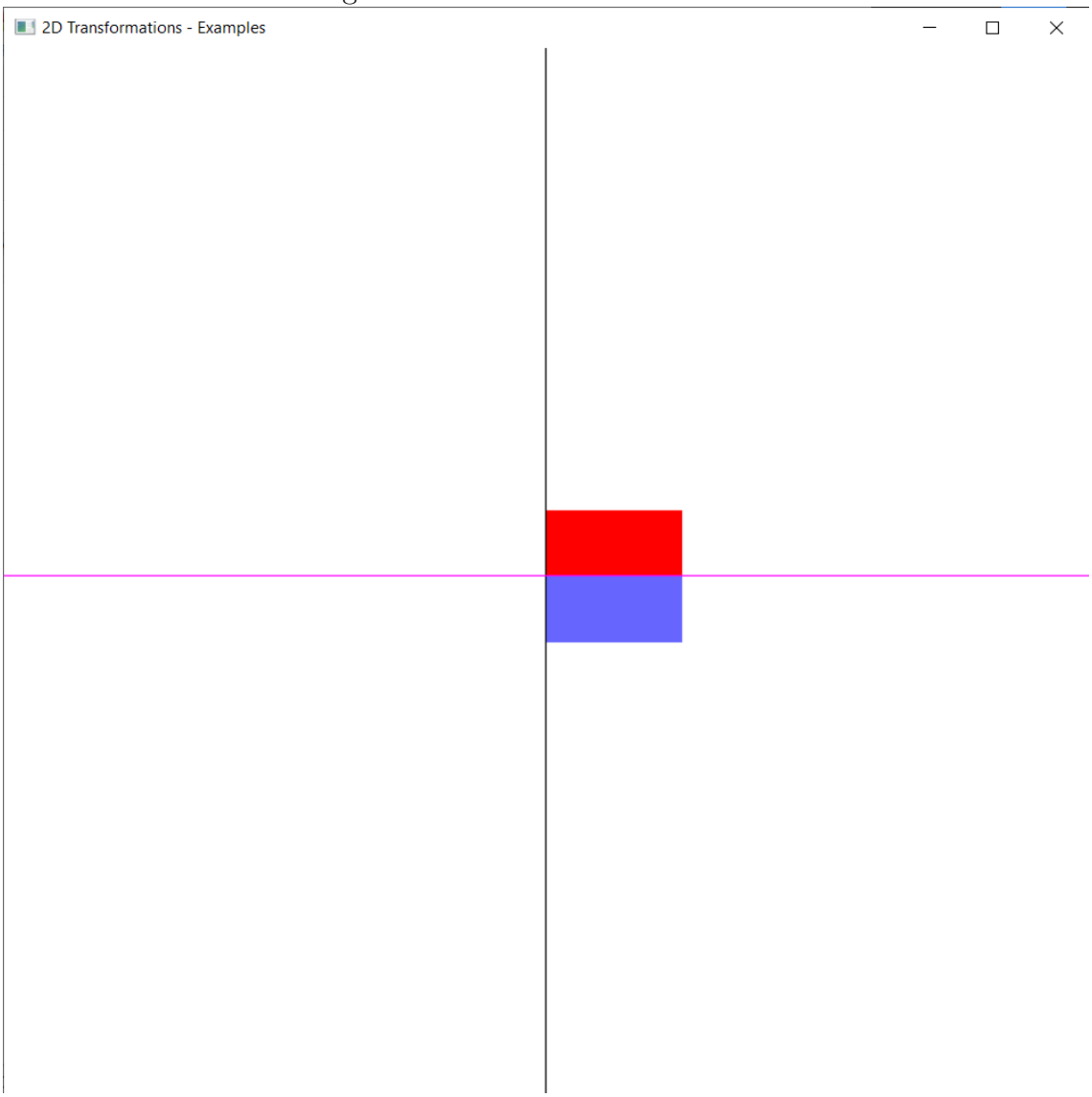
Output: Console

Figure 6: Console.



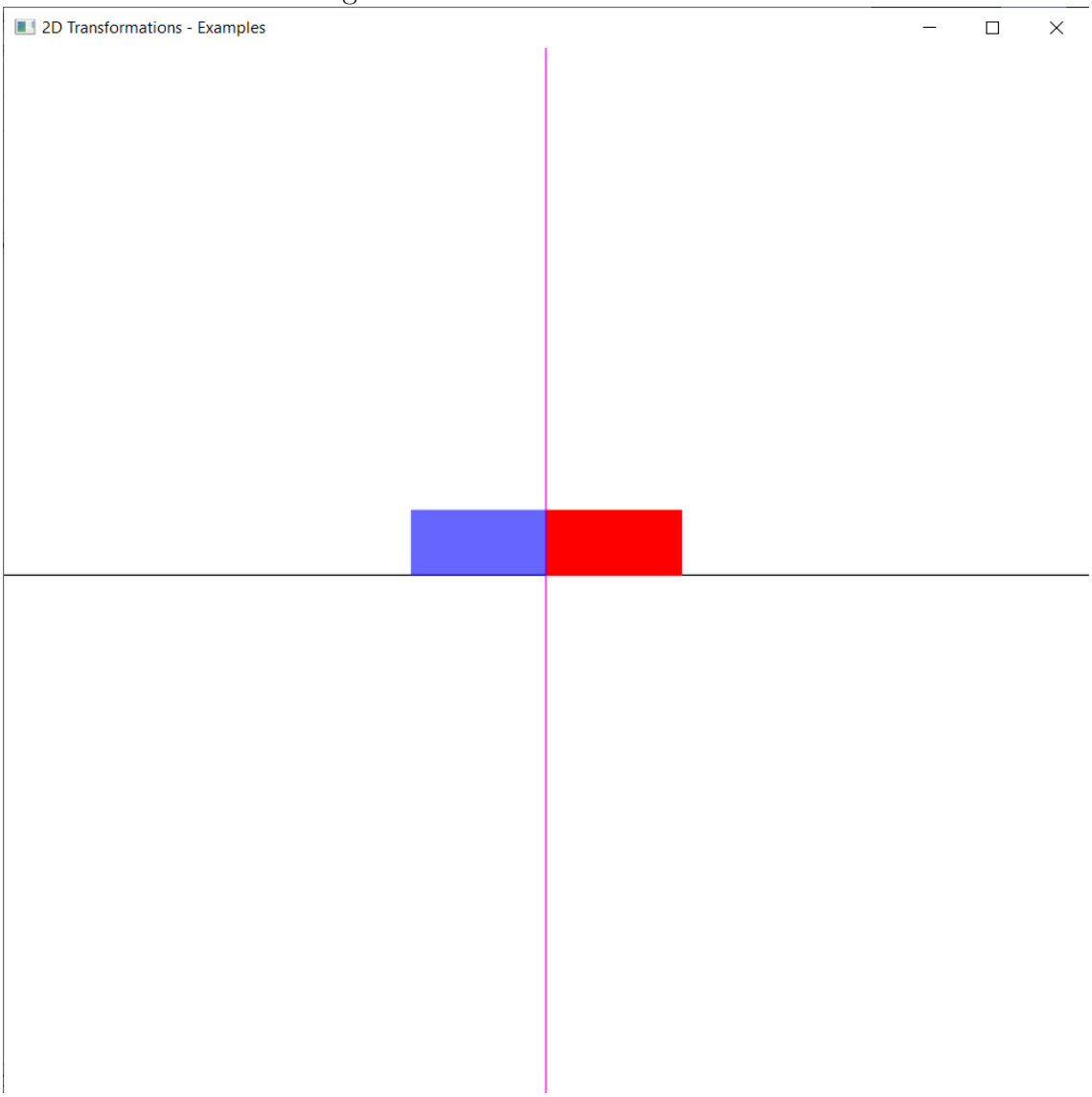
Output: Reflection About X Axis

Figure 7: Reflection About X Axis.



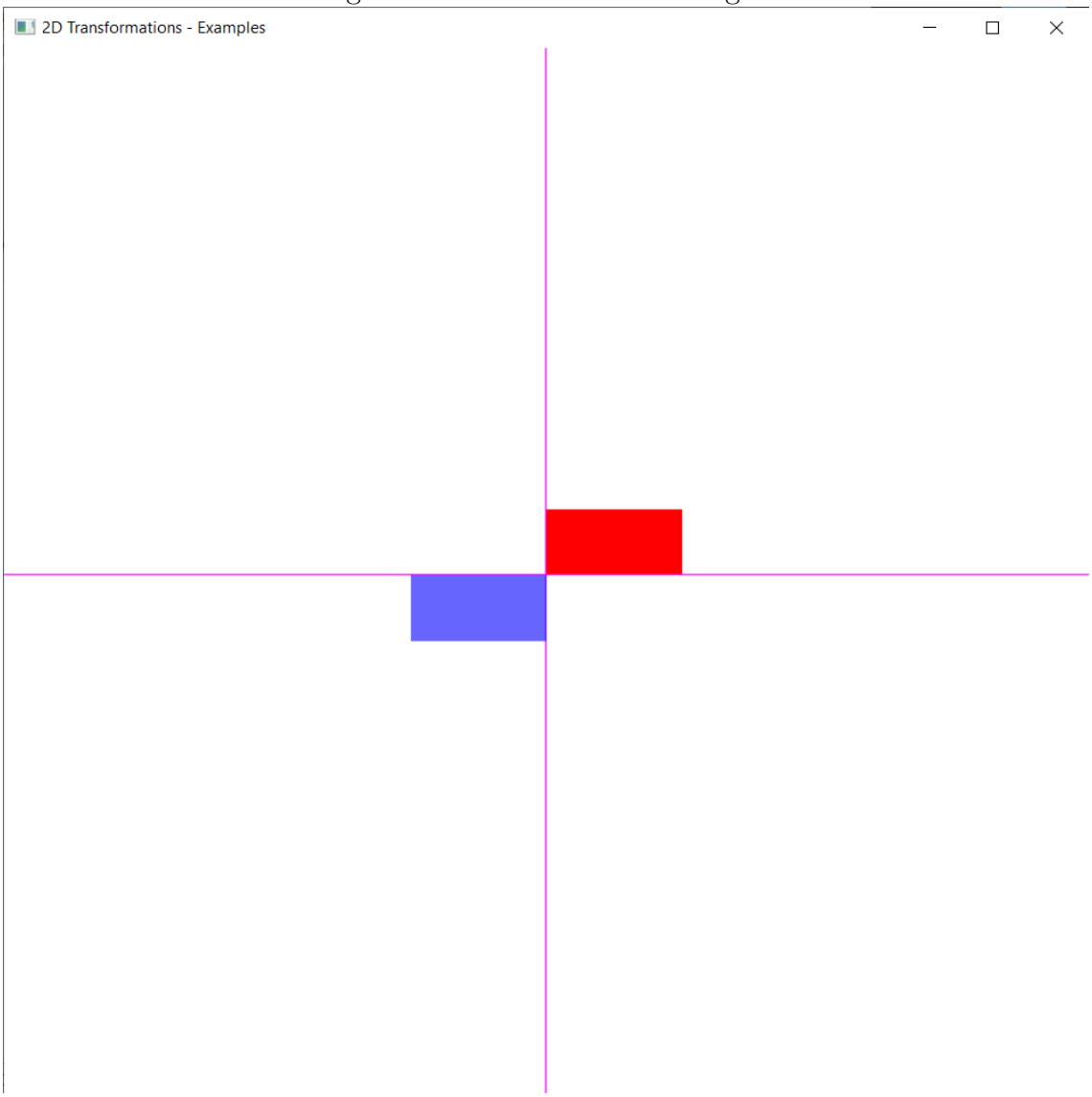
Output: Reflection About Y Axis

Figure 8: Reflection About Y Axis.



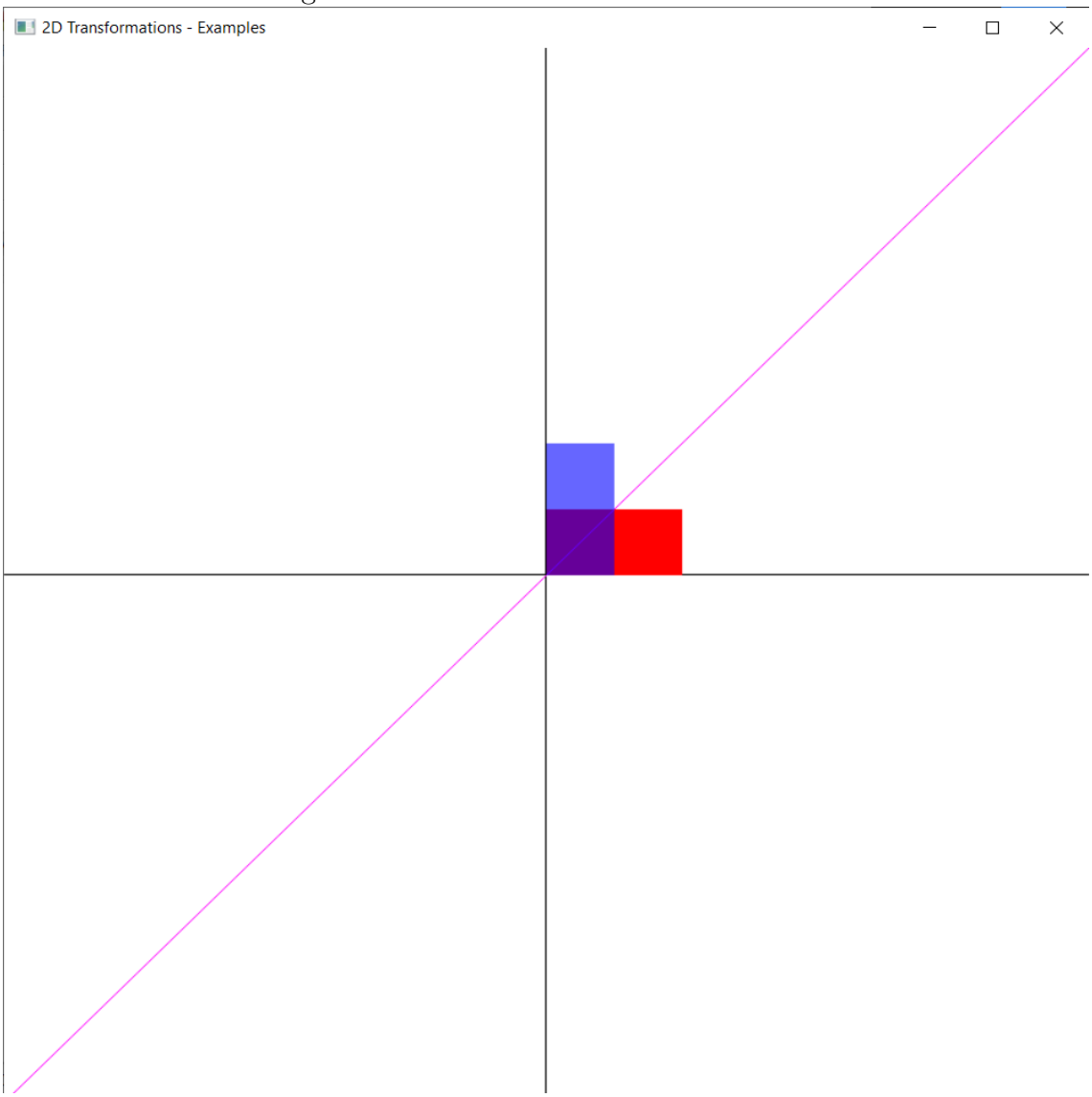
Output: Reflection About Origin

Figure 9: Reflection About Origin.



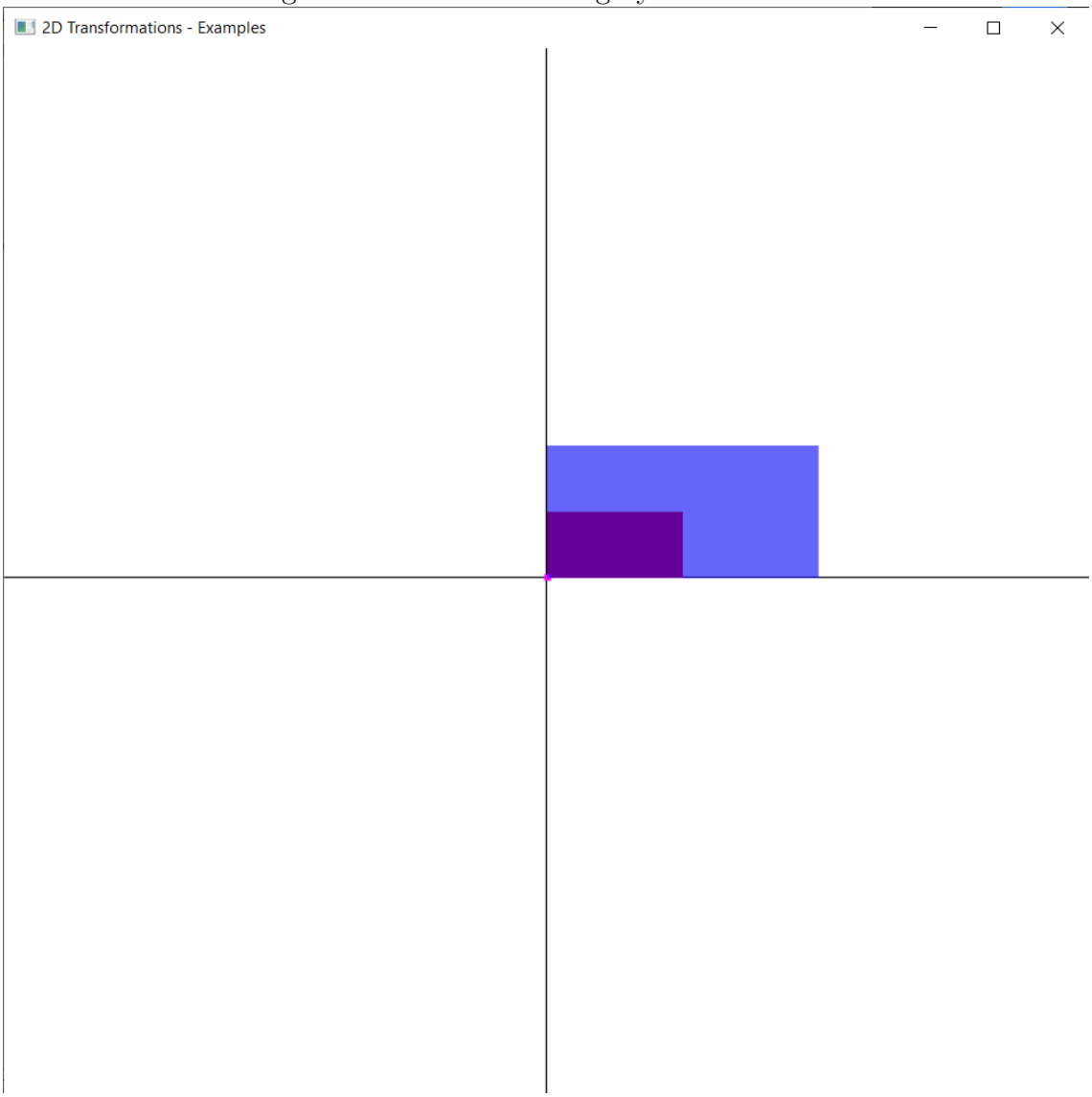
Output: Reflection About Line $X = Y$

Figure 10: Reflection About Line $X = Y$.



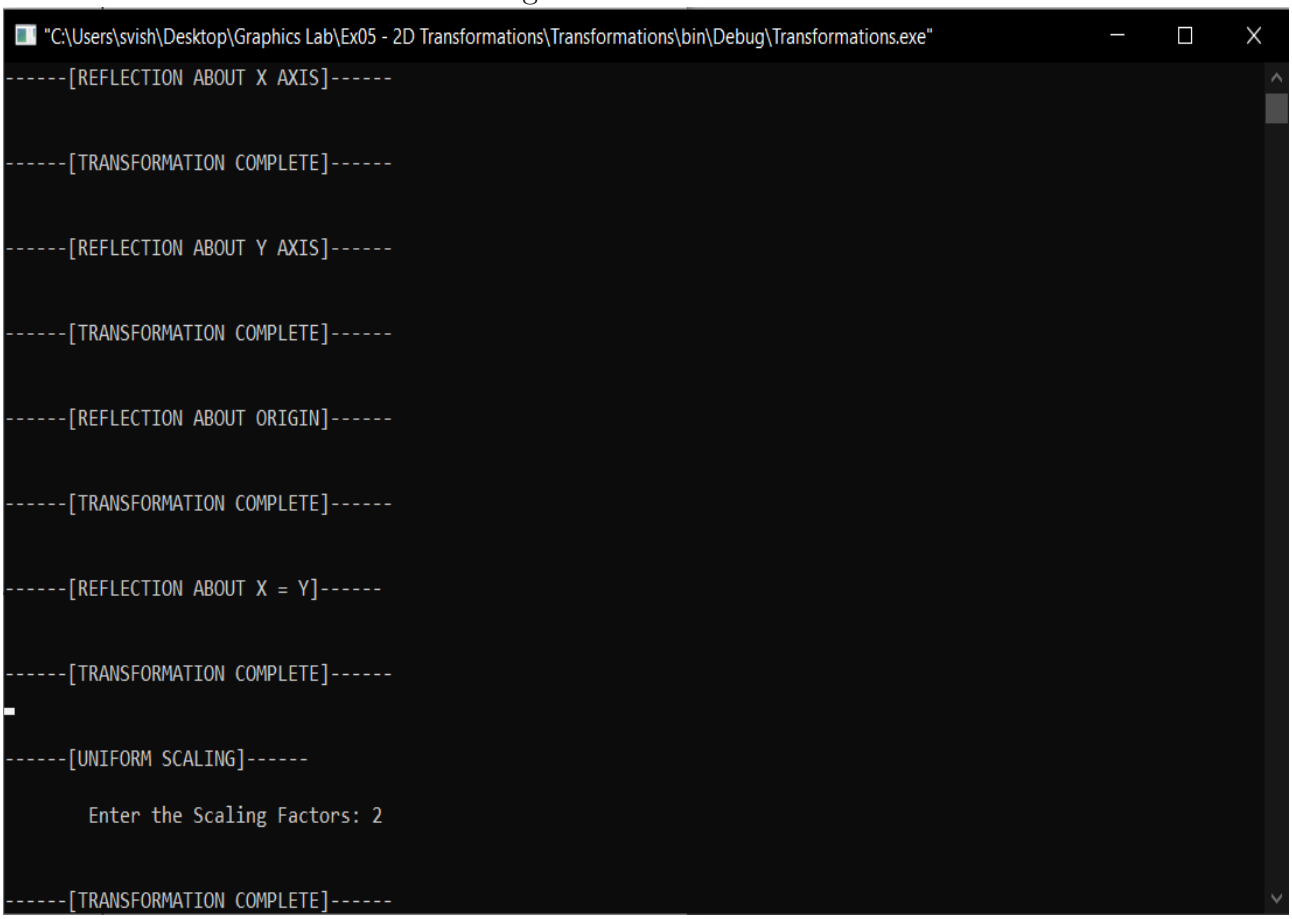
Output: Uniform Scaling

Figure 11: Uniform Scaling by a Factor of 2.



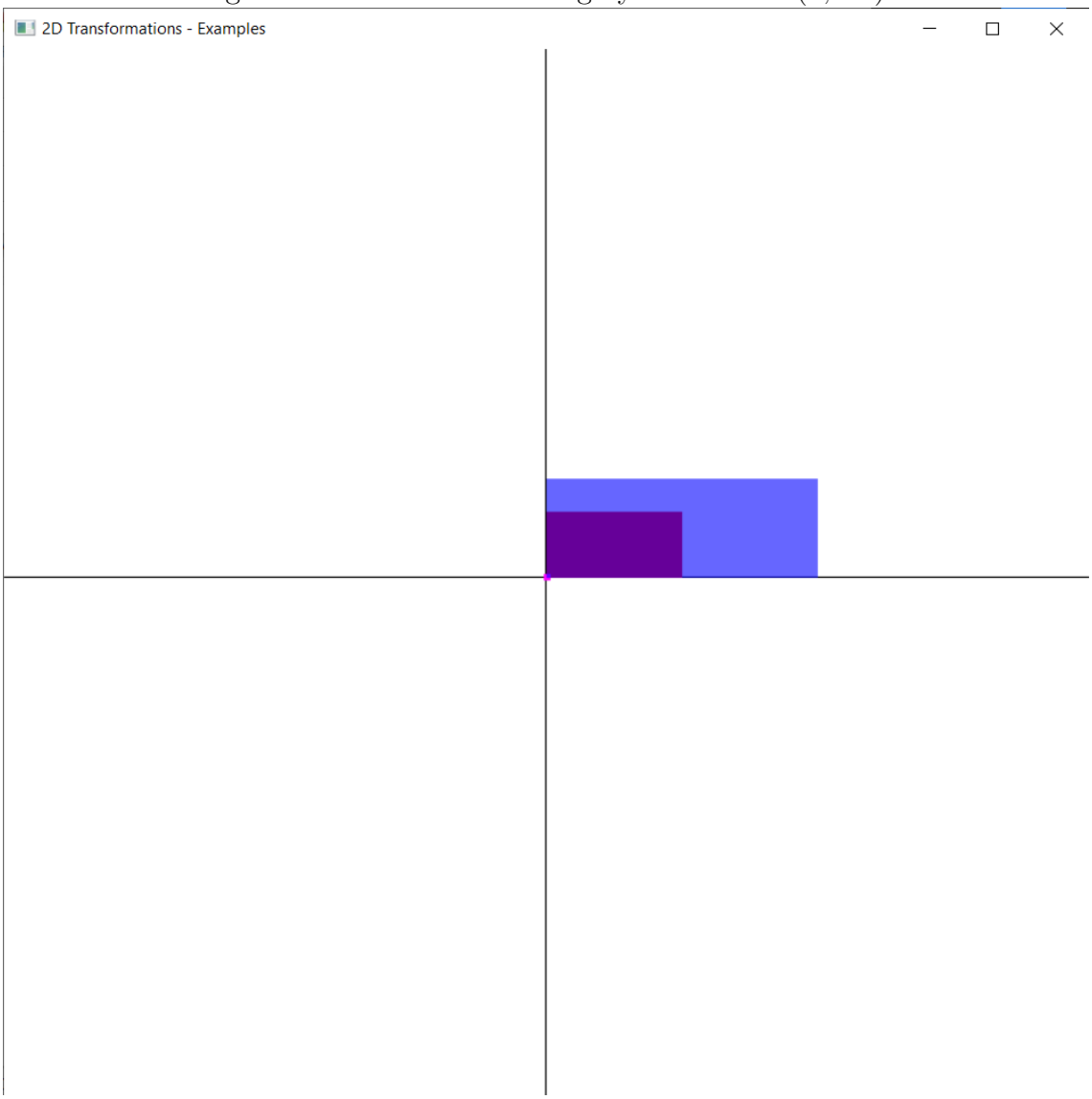
Output: Console

Figure 12: Console.



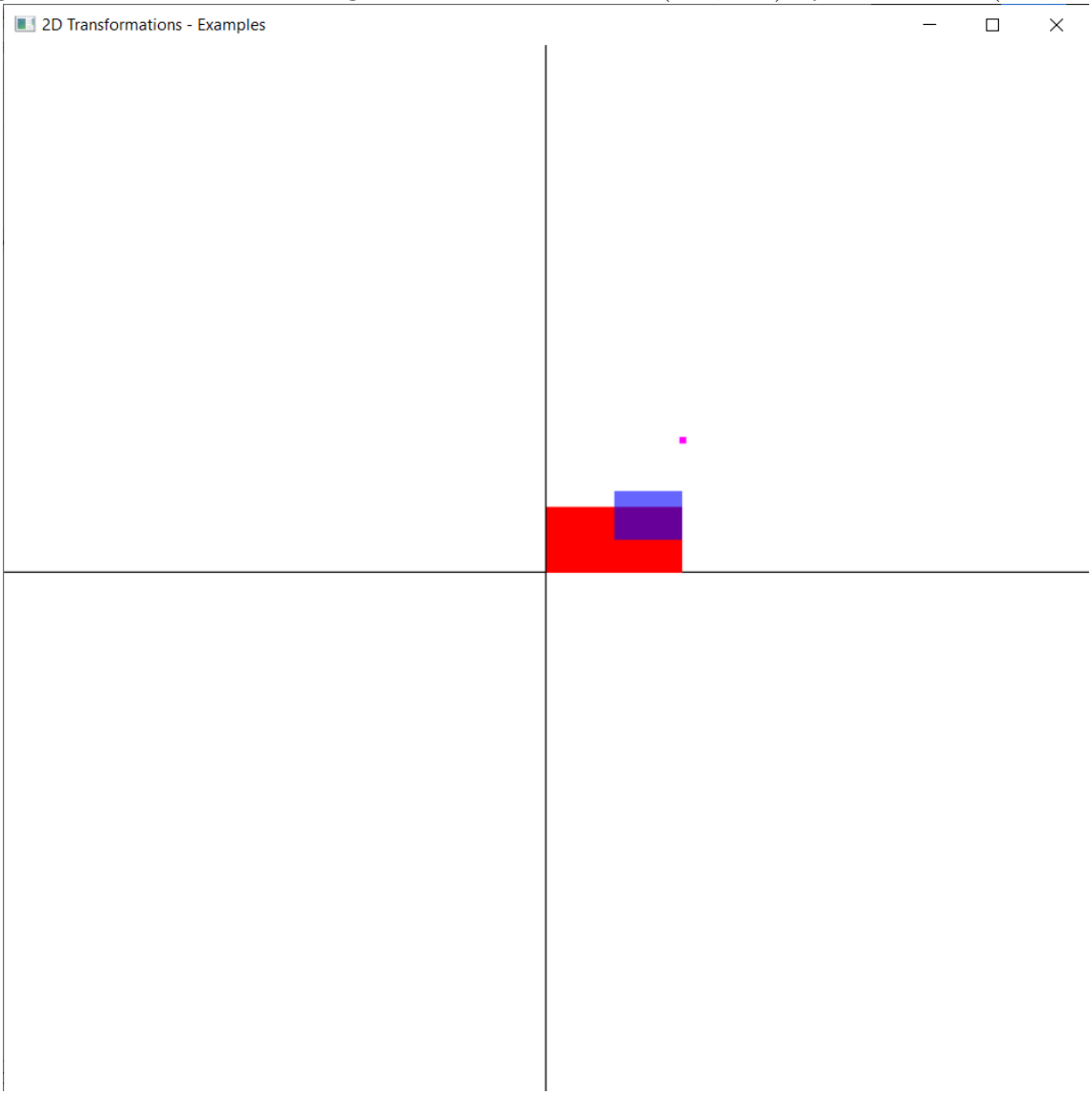
Output: Differential Scaling

Figure 13: Differential Scaling by a Factor of $(2, 1.5)$.



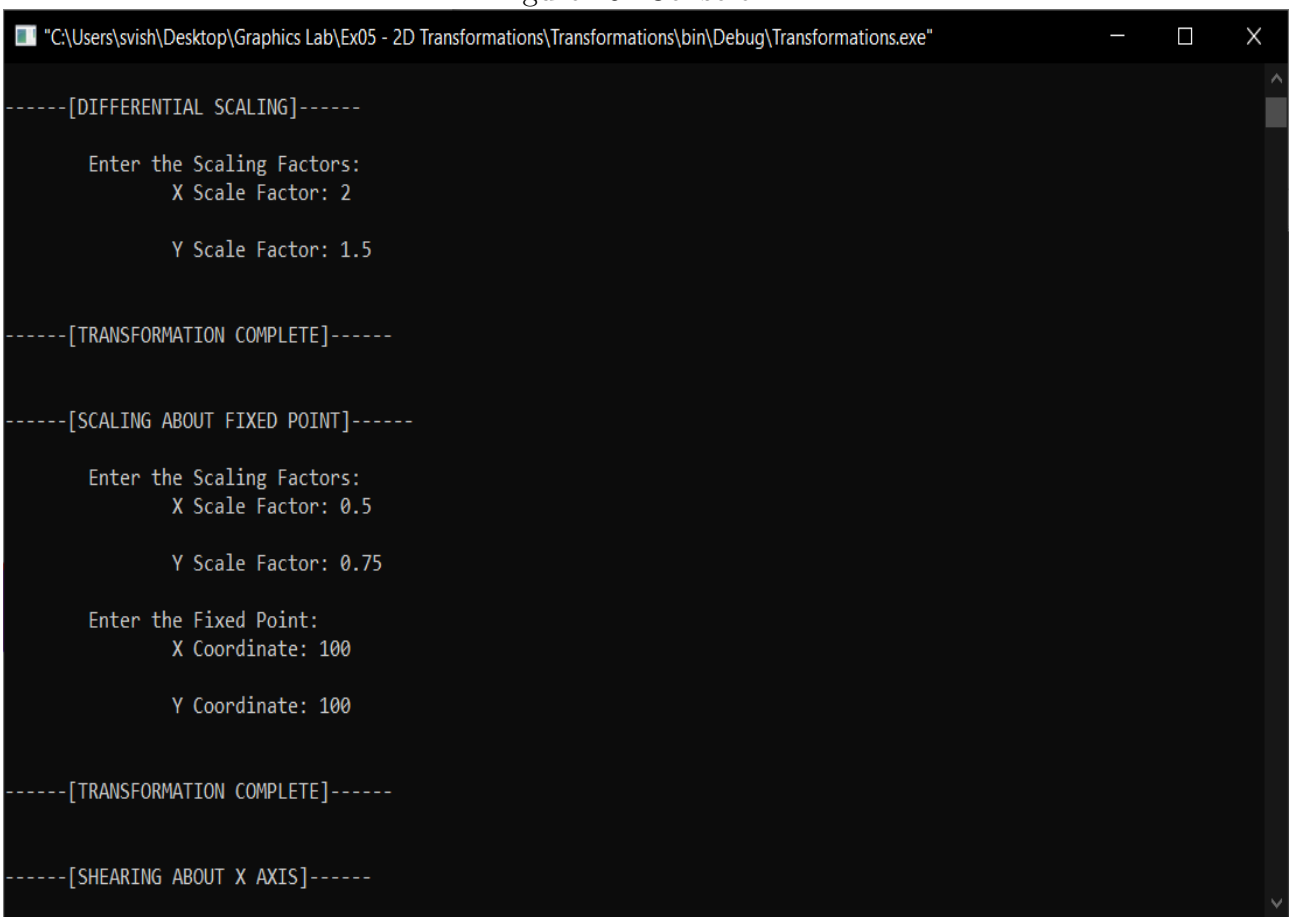
Output: Differential Scaling About A Fixed Point

Figure 14: Differential Scaling About A Fixed Point $(100, 100)$ by a Factor of $(0.5, 0.75)$.



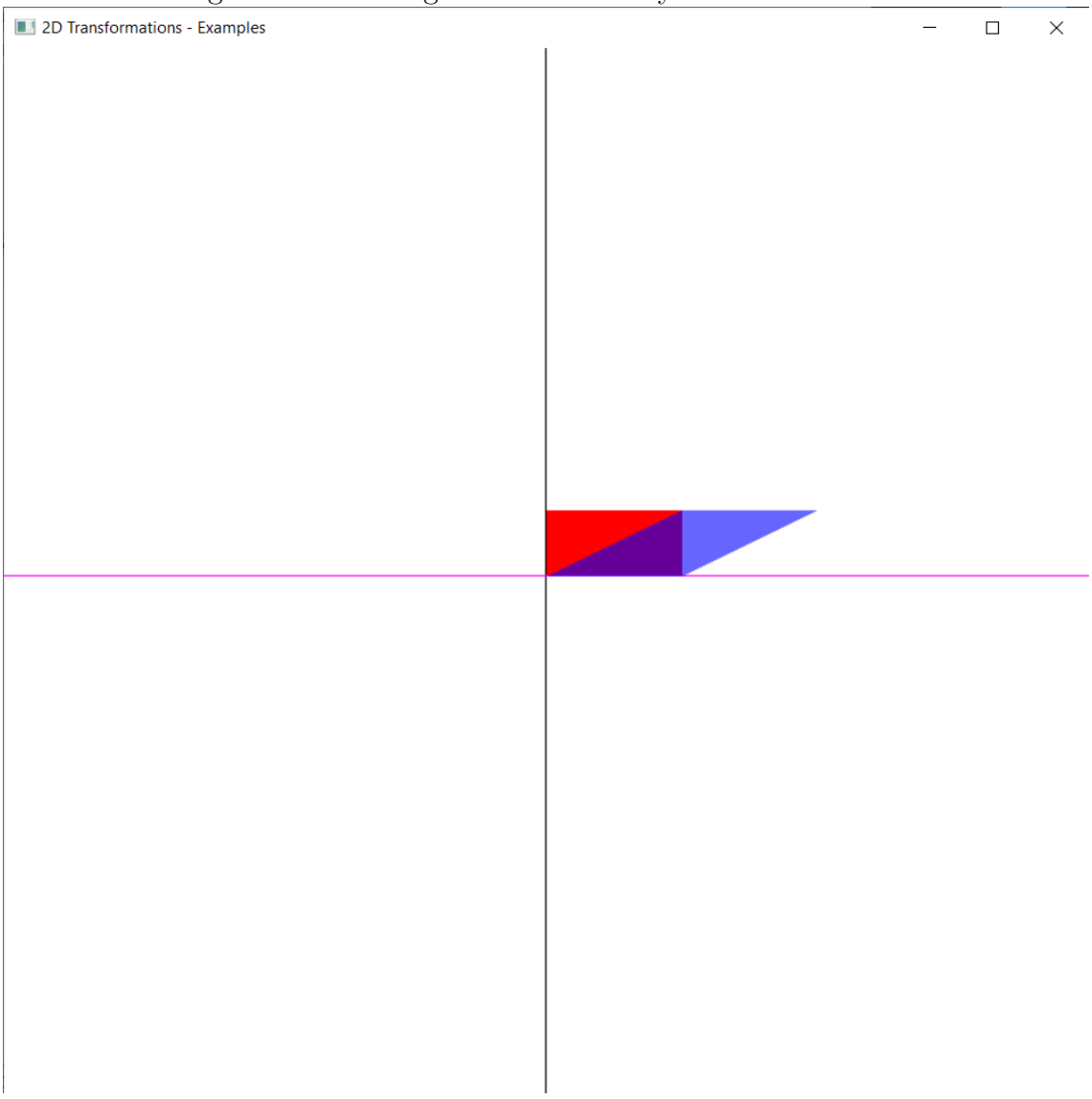
Output: Console

Figure 15: Console.



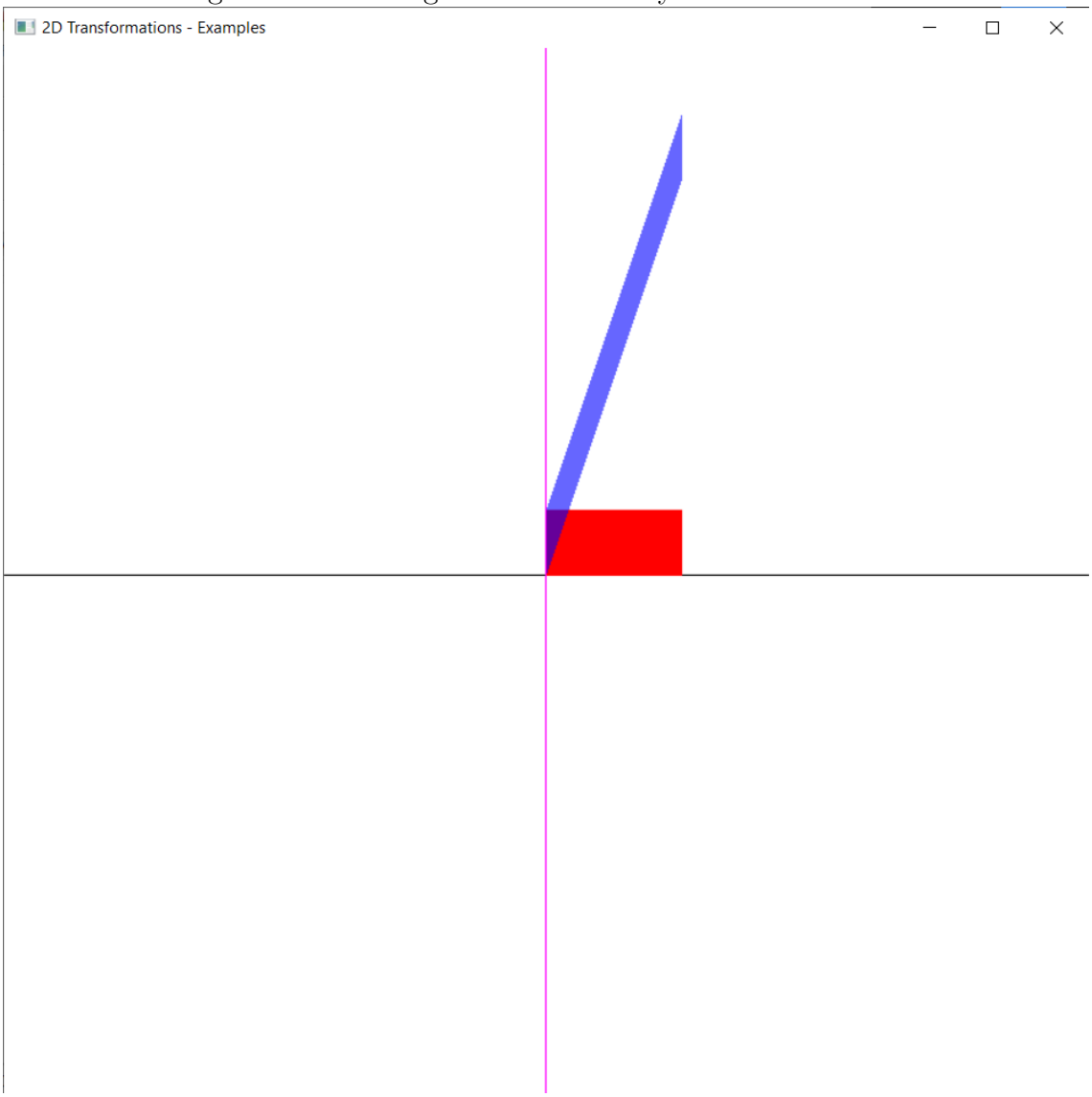
Output: Shearing About X Axis

Figure 16: Shearing About X Axis by a Parameter of 2.



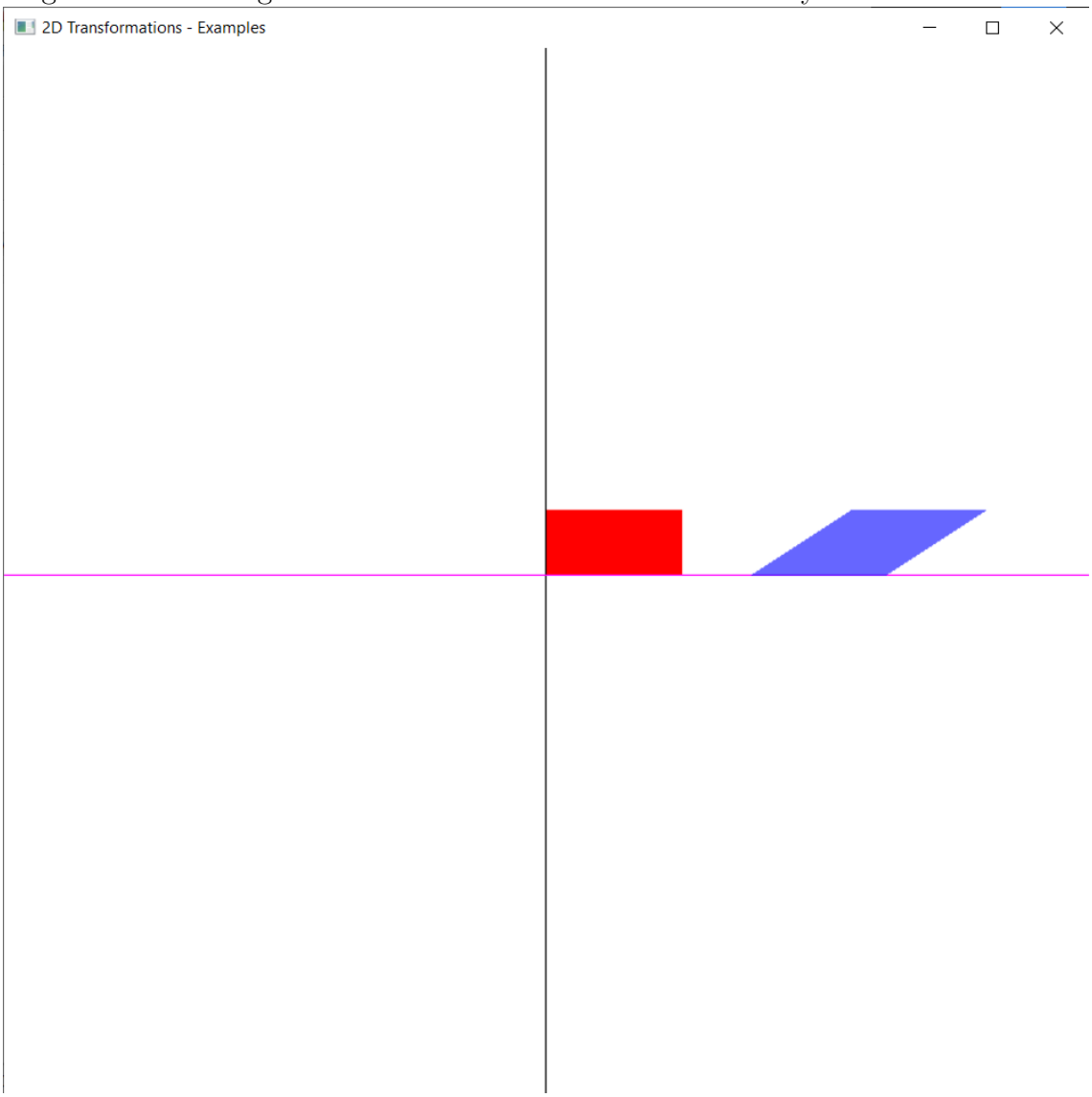
Output: Shearing About Y Axis

Figure 17: Shearing About Y Axis by a Parameter of 3.



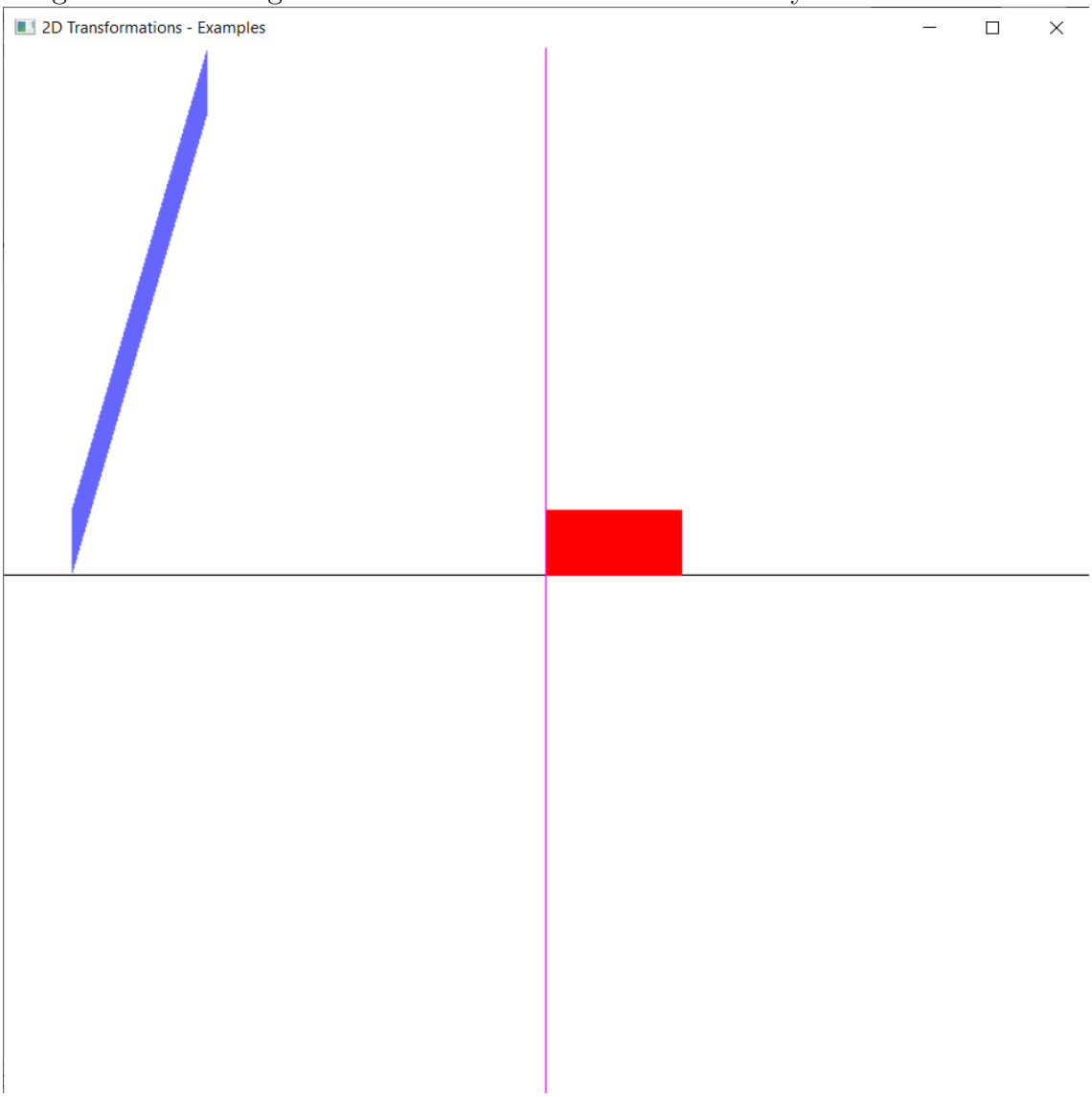
Output: Shearing About X Axis & Ref. Line $Y = y$

Figure 18: Shearing About X Axis & Ref. Line $Y = -100$ by a Parameter of 1.5.



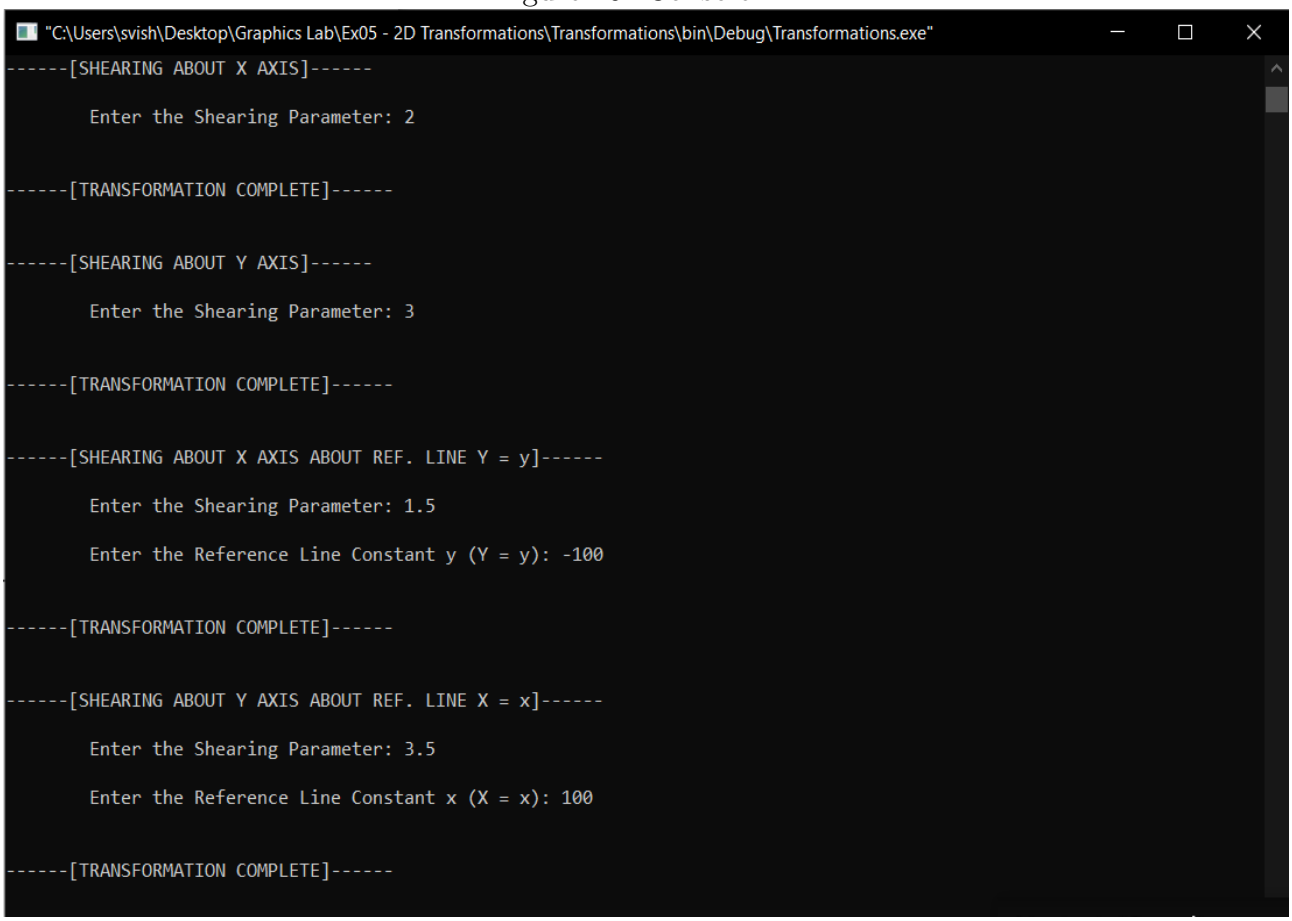
Output: Shearing About Y Axis & Ref. Line $X = x$

Figure 19: Shearing About Y Axis & Ref. Line $X = 100$ by a Parameter of 3.5.



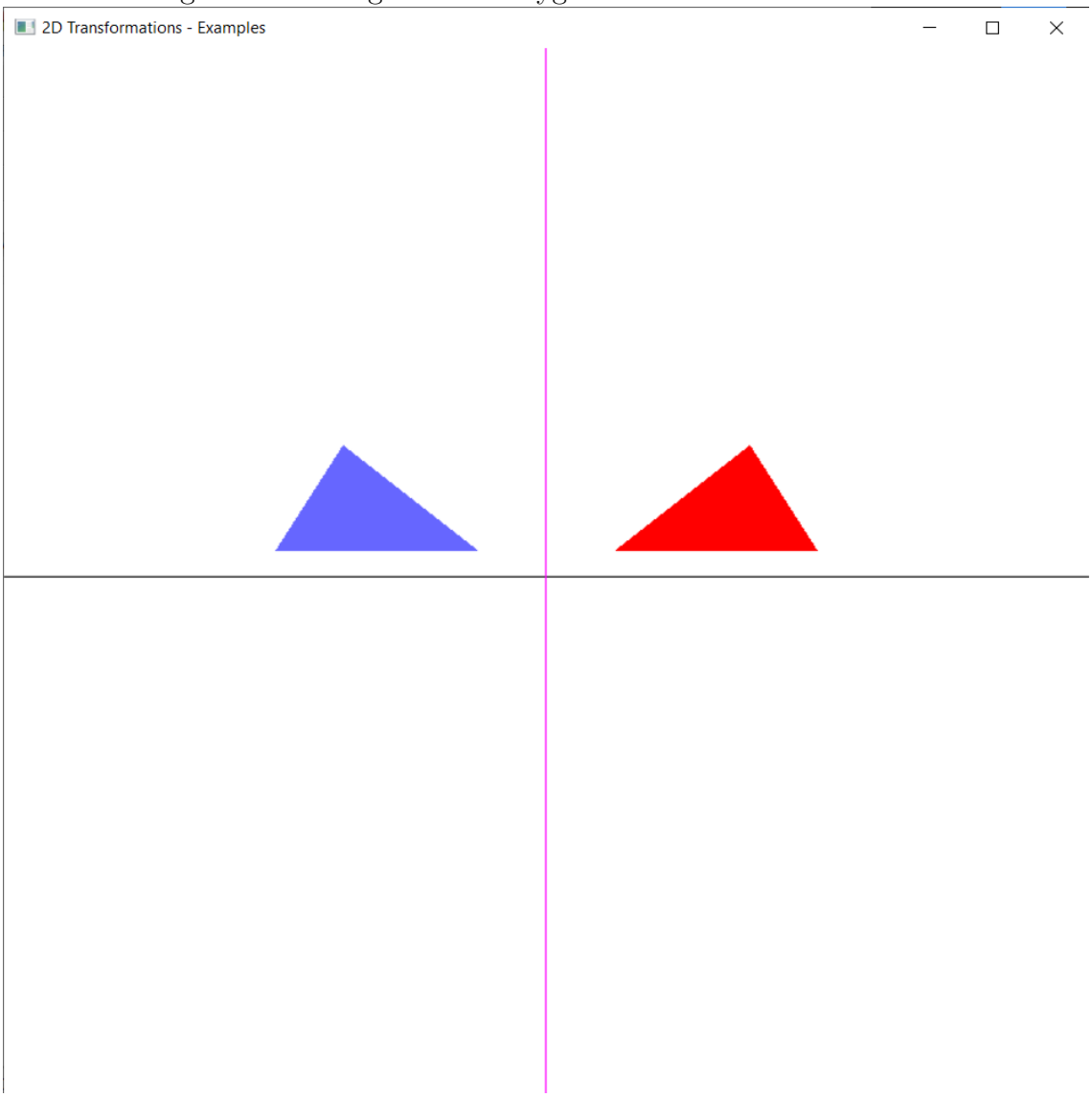
Output: Console

Figure 20: Console.



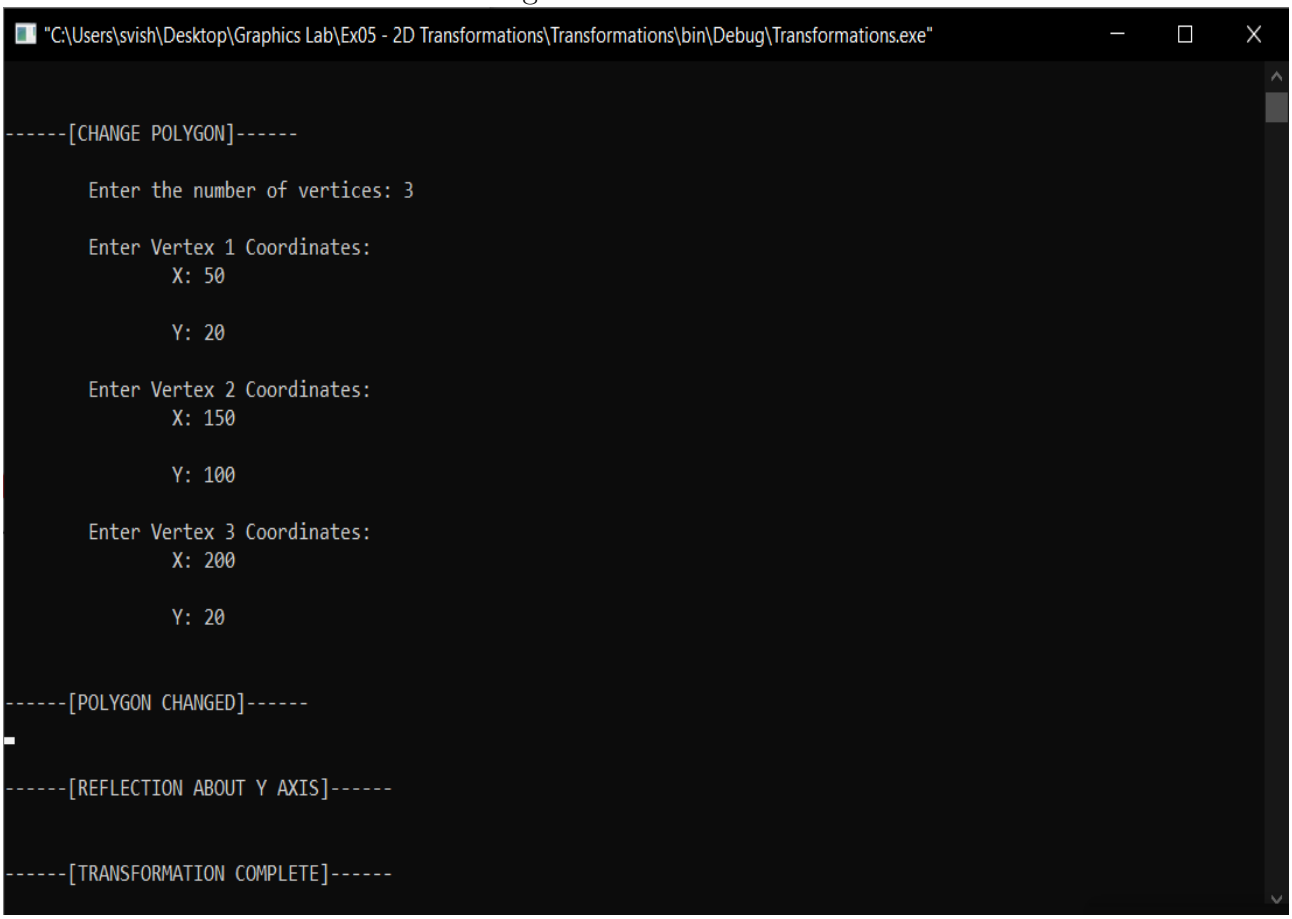
Output: Changed Base Polygon and its Y Axis Reflection

Figure 21: Changed Base Polygon and its Y Axis Reflection



Output: Console

Figure 22: Console.



Learning Outcome:

- I understood how to convert (x, y) global coordinates into **homogeneous coordinates** and its relevance in applying transformations.
- I understood how to apply **matrix multiplication** operations to achieve various 2-D Transformations.
- I learnt about the transformation matrices for translation, rotation, reflection, scaling & shearing.
- I implemented separate classes for **PolygonShape** and **Point** for ease of use and modularizing the program.
- I understood how to implement a **GLUT Menu** for a menu-based approach to apply transformations.
- I understood how to draw translucent objects with the help of **glDepthMask(), glBlendFunc() and glColor4f()** with parameter **ALPHA**.
- I learnt how to project a **Cartesian Plane** with the use of **gluOrtho2D()**.
- I learnt to use **enum** to simplify and enhance readability for my menu-driven program.
- I learnt how to use default arguments in C++ to provide default variables.
- I implemented **translation** about a given translation vector.
- I implemented **rotation** about an angle θ and optionally about a pivot point (x_r, y_r).
- I implemented **reflection** about X-Axis, Y-Axis, Origin and the line $X = Y$.
- I implemented **scaling** uniformly, differentially and optionally about a fixed point (x_f, y_f).
- I implemented **shearing** about X-Axis and Y-Axis and optionally to include a reference line $Y = y$ and $X = x$ respectively.
- I created a function that allows the user to change the base polygon shape outputted in the window.
- I emphasized the use of **different colors** to highlight the transformed image, fixed points (if any) and reference lines (if any).
- I understood that OpenGL code executes in an **event-driven fashion**, thus while it waits for user-input, the output window might be stalled (unresponsive) and might need to be refreshed after the user I/O has finished.

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

05 September 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 6: 2D Composite Transformations and Windowing in C++ using OpenGL

Aim:

- To compute the composite transformation matrix for any 2 transformations given as input by the user and applying it on the object. The transformation can be any combination of the following:
  - Translation
  - Rotation
  - Scaling
  - Reflection
  - Shearing

  Display the original and the transformed object. Calculate the final transformation matrix by multiplying the two individual transformation matrices and then apply it to the object.

  Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x and y axis)
- Create a window with any 2D object and a different sized viewport. Apply window to viewport transformation on the object. Display both window and viewport.

Code: 2D Composite Transformations:

```
/*
To compute the composite transformation matrix for any 2 transformations given as input by
the user and applying it on the object. The transformation can be any combination of the following:
Translation, Rotation, Scaling, Reflection & Shearing.
*/

#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
#include <iostream>     //for cin, cout
#include <cstring>      //for memcpy

using namespace std;

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 800;
const int X_MIN = -400;
const int X_MAX = 400;
const int Y_MIN = -400;
const int Y_MAX = 400;
const int FPS = 60;

enum Axes {xAxis = 0, yAxis = 1};
enum Lines {XAxis = 0, YAxis = 1, Origin = 2, XEqualsY = 3};
enum Transforms {Translation = 1, Rotation = 2, Scaling = 3, Reflection = 4, Shearing = 5};

class Point{
private:
    GLdouble x, y, h;

public:
    Point(){
        x = y = 0;
        h = 1;
    }

    Point(GLint xCoord, GLint yCoord){
        x = xCoord;
        y = yCoord;
        h = 1;
    }

    Point(GLint xCoord, GLint yCoord, GLint H){
        x = xCoord;
```

```
45          y = yCoord;
46          h = H;
47      }
48
49      void setCoords(GLdouble xCoord, GLdouble yCoord){
50          x = xCoord;
51          y = yCoord;
52      }
53
54      void setHomogeneousCoords(GLdouble xCoord, GLdouble yCoord, GLdouble H
        ){
55          x = xCoord;
56          y = yCoord;
57          h = H;
58      }
59
60      GLdouble getX() const{
61          return x;
62      }
63
64      GLdouble getY() const{
65          return y;
66      }
67
68      GLdouble getH() const{
69          return h;
70      }
71
72      GLdouble getHomogenousX() const{
73          return x * h;
74
75      }
76
77      GLdouble getHomogenousY() const{
78          return y * h;
79      }
80  };
81
82
83  class PolygonShape{
84  private:
85      int numVertices;
86      Point *points;
87      bool matrix1Flag, matrix2Flag;
88      double matrix1[3][3], matrix2[3][3], compositeMatrix[3][3];
89
90  public:
91      PolygonShape(){
92          numVertices = 0;
93          matrix1Flag = false;
94          matrix2Flag = false;
```

```
95      }
96
97      PolygonShape(int noVertices){
98          numVertices = noVertices;
99          points = new Point[numVertices];
100         matrix1Flag = false;
101         matrix2Flag = false;
102     }
103
104     int getVertexCount() const{
105         return numVertices;
106     }
107
108     Point getPoint(int i){
109         return points[i];
110     }
111
112     void setVertices(int noVertices){
113         numVertices = noVertices;
114         points = new Point[numVertices];
115     }
116
117     void setPoint(int i, GLdouble x, GLdouble y, GLdouble h = 1){
118         points[i].setHomogeneousCoords(x, y, h);
119     }
120
121     void clearMatrices(){
122        //Clear the transformation matrices to identity matrices
123
124         matrix1Flag = false;
125         matrix2Flag = false;
126
127         for(int i = 0; i < 3; i++){
128             for(int j = 0; j < 3; j++){
129                 if(i == j){
130                     matrix1[i][j] = 1;
131                     matrix2[i][j] = 1;
132                 } else{
133                     matrix1[i][j] = 0;
134                     matrix2[i][j] = 0;
135                 }
136             }
137         }
138     }
139
140     void setTranslationMatrix(Point translationVector){
141         //Sets a translation matrix to one of the transformation matrices
142
143         double translationMatrix[3][3] = {  {1, 0, translationVector.
    getHomogenousX()},
```

```
144                                             {0, 1, translationVector.
    getHomogenousY()},
145                                             {0, 0, 1}};
146
147         if(!matrix1Flag){
148             memcpy(matrix1, translationMatrix, sizeof(translationMatrix));
149             matrix1Flag = true;
150         } else{
151             memcpy(matrix2, translationMatrix, sizeof(translationMatrix));
152             matrix2Flag = true;
153         }
154     }
155
156     void setRotationMatrix(int rotationAngle, Point pivot = Point(0, 0, 1)
    ){
157         //Sets a rotation matrix to one of the transformation matrices
158
159         double rotationAngleInRadians = rotationAngle * 3.14159/180;
160         double cosAngle = cos(rotationAngleInRadians);
161         double sinAngle = sin(rotationAngleInRadians);
162
163         double xPivotValue = (pivot.getX() * (1 - cosAngle)) + (pivot.getY
    () * sinAngle);
164         double yPivotValue = (pivot.getY() * (1 - cosAngle)) - (pivot.getX
    () * sinAngle);
165
166         double rotationMatrix[3][3] = { {cosAngle, -sinAngle, xPivotValue
    },
167                                         {sinAngle, cosAngle, yPivotValue},
168                                         {0, 0, 1}};
169
170         if(!matrix1Flag){
171             memcpy(matrix1, rotationMatrix, sizeof(rotationMatrix));
172             matrix1Flag = true;
173         } else{
174             memcpy(matrix2, rotationMatrix, sizeof(rotationMatrix));
175             matrix2Flag = true;
176         }
177     }
178
179     void setReflectionMatrix(int line){
180         //Sets a reflection matrix to one of the transformation matrices
181
182         double reflectionMatrix[3][3];
183
184         switch(line){
185             case XAxis:{
186                 double temp[3][3] = {   {1, 0, 0},
187                                         {0, -1, 0},
188                                         {0, 0, 1}};
189
```

```
                memcpy(reflectionMatrix, temp, sizeof(temp));
                break;
            }


            case YAxis:{
                double temp[3][3] = {   {-1, 0, 0},
                                        {0, 1, 0},
                                        {0, 0, 1}};

                memcpy(reflectionMatrix, temp, sizeof(temp));
                break;
            }


            case Origin:{
                double temp[3][3] = {   {-1, 0, 0},
                                        {0, -1, 0},
                                        {0, 0, 1}};

                memcpy(reflectionMatrix, temp, sizeof(temp));
                break;
            }


            case XEqualsY:{
                double temp[3][3] = {   {0, 1, 0},
                                        {1, 0, 0},
                                        {0, 0, 1}};

                memcpy(reflectionMatrix, temp, sizeof(temp));
                break;
            }
        }

        if(!matrix1Flag){
            memcpy(matrix1, reflectionMatrix, sizeof(reflectionMatrix));
            matrix1Flag = true;
        } else{
            memcpy(matrix2, reflectionMatrix, sizeof(reflectionMatrix));
            matrix2Flag = true;
        }
    }

    void setScaleMatrix(double ScaleX, double ScaleY, Point fixed = Point
    (0, 0, 1)){
        //Sets a scale matrix to one of the transformation matrices

        double xFixedValue = fixed.getX() * (1 - ScaleX);
        double yFixedValue = fixed.getY() * (1 - ScaleY);

```

```
240         double scaleMatrix[3][3] = {  {ScaleX, 0, xFixedValue},
241                                       {0, ScaleY, yFixedValue},
242                                       {0, 0, 1}};
243 
244         if(!matrix1Flag){
245             memcpy(matrix1, scaleMatrix, sizeof(scaleMatrix));
246             matrix1Flag = true;
247         } else{
248             memcpy(matrix2, scaleMatrix, sizeof(scaleMatrix));
249             matrix2Flag = true;
250         }
251     }
252 
253     void setShearMatrix(double shearParam, int axis, double refConst = 0){
254         //Sets a shear matrix to one of the transformation matrices
255 
256         double shearMatrix[3][3];
257 
258         switch(axis){
259             case xAxis:{
260                 double temp[3][3] = {   {1, shearParam, -shearParam *
    refConst},
261                                         {0, 1, 0},
262                                         {0, 0, 1}};
263 
264                 memcpy(shearMatrix, temp, sizeof(temp));
265                 break;
266             }
267 
268 
269             case yAxis:{
270                 double temp[3][3] = {   {1, 0, -shearParam * refConst},
271                                         {shearParam, 1, 0},
272                                         {0, 0, 1}};
273 
274                 memcpy(shearMatrix, temp, sizeof(temp));
275                 break;
276             }
277         }
278 
279         if(!matrix1Flag){
280             memcpy(matrix1, shearMatrix, sizeof(shearMatrix));
281             matrix1Flag = true;
282         } else{
283             memcpy(matrix2, shearMatrix, sizeof(shearMatrix));
284             matrix2Flag = true;
285         }
286     }
287 
288     void setCompositeMatrix(){
289         //Sets the composite matrix based on matrix multiplication
```

```
290         //of the two transformation matrices
291
292         if(!matrix1Flag || !matrix2Flag){
293             //if any one matrix is not set, don't multiply
294             return;
295         }
296
297         for(int i = 0; i < 3; i++){
298             for(int j = 0; j < 3; j++){
299                 double tempSum = 0;
300
301                 for(int k = 0; k < 3; k++){
302                     tempSum += matrix1[i][k] * matrix2[k][j];
303                 }
304
305                 compositeMatrix[i][j] = tempSum;
306             }
307         }
308     }
309
310     PolygonShape getTransformedPolygon(){
311         //Obtain the transformed polygon based upon the composite
    transformation
312
313         PolygonShape polyDash(numVertices);
314         double values[3];
315
316         for(int i = 0; i < numVertices; i++){
317             Point p = getPoint(i);
318
319             //[3 x 3] x [3 x 1] = [3 x 1] matrix
320             for(int j = 0; j < 3; j++){
321                 values[j] = compositeMatrix[j][0] * p.getHomogenousX() +
322                             compositeMatrix[j][1] * p.getHomogenousY() +
323                             compositeMatrix[j][2] * p.getH();
324             }
325
326             polyDash.setPoint(i, values[0]/p.getH(), values[1]/p.getH(),
    values[2]);
327         }
328
329         return polyDash;
330     }
331 };
332
333
334 void initializeDisplay();
335 void plotComponents();
336 void dummyFunction();
337 void renderContents();
338 void mainLoop(int val);
```

```
339 void setTransformMatrices();
340 void plotTransformation();
341 void drawAxes();
342 void drawPolygon(PolygonShape polygon, bool transformed = false);
343
344 PolygonShape polygon;                    //Global PolygonShape object to be
        plotted on the graph
345 int transform1 = 0, transform2 = 0;      //Global variable to keep track of
        chosen transformation
346
347 int main(int argc, char **argv){
348     glutInit(&argc, argv);
349     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
350     glutInitWindowPosition(0, 0);
351     glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
352     glutCreateWindow("2D Composite Transformations - Examples");
353
354     printf("\n------[2D COMPOSITE TRANSFORMATIONS]------\n");
355     printf("\nUsage:\tSelect the required transformations in the console."
       );
356     printf("\n\tEnter the parameters for the specified transformations.");
357     printf("\n\tView the output in the GLUT window.");
358     printf("\n\n------[2D COMPOSITE TRANSFORMATIONS]------\n\n");
359
360     //Set the initial default polygon for the graph
361     polygon.setVertices(4);
362     polygon.setPoint(0, 0, 0);
363     polygon.setPoint(1, 0, 50);
364     polygon.setPoint(2, 100, 50);
365     polygon.setPoint(3, 100, 0);
366
367     initializeDisplay();
368
369     glutDisplayFunc(dummyFunction);
370
371     //important - to refresh screen periodically
372     glutTimerFunc(1000/FPS, mainLoop, 0);
373
374     glutMainLoop();
375
376     return 1;
377 }
378
379 void initializeDisplay(){
380     //Initialize the display parameters
381
382     glClearColor(1, 1, 1, 0);
383     glMatrixMode(GL_PROJECTION);
384     gluOrtho2D(X_MIN, X_MAX, Y_MIN, Y_MAX);
385     glClear(GL_COLOR_BUFFER_BIT);   //Clear the display window
386
```

```
387     glEnable(GL_BLEND);      //enable blending (translucent colors)
388     glDepthMask(GL_FALSE);
389     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);  //set the blend
        function for translucency
390 }
391
392 void plotComponents(){
393     //Plot the axes and the base polygon
394
395     glClear(GL_COLOR_BUFFER_BIT);   //Clear the display window
396     drawAxes();
397     drawPolygon(polygon);
398     glFlush();
399 }
400
401 void dummyFunction(){
402     //Placeholder function to be called in glutDisplayFunc
403 }
404
405 void mainLoop(int val){
406     //Function to be called within the glutTimerFunc periodically to
        refresh screen at 60FPS
407     renderContents();
408 }
409
410
411 void renderContents(){
412     //to render the graph along with a user-defined composite
        transformation
413
414     plotComponents();
415
416     while(true){
417         //Await user input
418
419         cout << "\nChoose Transformation 1: " << endl;
420         cout << "\t1 for Translation" << endl;
421         cout << "\t2 for Rotation" << endl;
422         cout << "\t3 for Scaling" << endl;
423         cout << "\t4 for Reflection" << endl;
424         cout << "\t5 for Shearing" << endl;
425         cout << "\t0 to Exit" << endl;
426         cout << "\tYour Option -> ";
427         cin >> transform1;
428
429         if(!transform1){    //user chooses to exit
430             exit(0);
431         }
432
433         cout << "\nChoose Transformation 2: " << endl;
434         cout << "\t1 for Translation" << endl;
```

```
435         cout << "\t2 for Rotation" << endl;
436         cout << "\t3 for Scaling" << endl;
437         cout << "\t4 for Reflection" << endl;
438         cout << "\t5 for Shearing" << endl;
439         cout << "\tYour Option -> ";
440         cin >> transform2;
441 
442         plotComponents();
443         polygon.clearMatrices();        //clear previous transformations
444         setTransformMatrices();         //set the new transform matrices
445         polygon.setCompositeMatrix();   //multiply to form composite
    matrix
446 
447         PolygonShape polygonDash;
448         polygonDash = polygon.getTransformedPolygon();
449 
450         //To print the transformed polygon's coordinates
451         // for(int i = 0; i < polygonDash.getVertexCount(); i++){
452         //     Point pDash = polygonDash.getPoint(i);
453         //     cout << "(" << pDash.getX() << ", " << pDash.getY() << ")"
    << endl;
454         // }
455 
456         drawPolygon(polygonDash, true);
457         glFlush();
458         //glutPostRedisplay();
459     }
460 }
461 
462 void setTransformMatrices(){
463     int i = 0;
464 
465     while(i < 2){
466         int currentTransform = (i == 0) ? transform1 : transform2;
467         i++;
468 
469         switch(currentTransform){
470             case Translation:{
471                 double x, y;
472                 cout << "\n\n------[TRANSLATION]------" << endl;
473                 cout << "\n\tEnter the Translation Vector Magnitude: ";
474                 cout << "\n\t\tX Component: "; cin >> x;
475                 cout << "\n\t\tY Component: "; cin >> y;
476 
477                 polygon.setTranslationMatrix(Point(x, y, 1));
478                 cout << "\n\n------[TRANSLATION NOTED]------" << endl;
479                 break;
480             }
481 
482             case Rotation:{
483                 double rotationAngle, x = 0, y = 0;
```

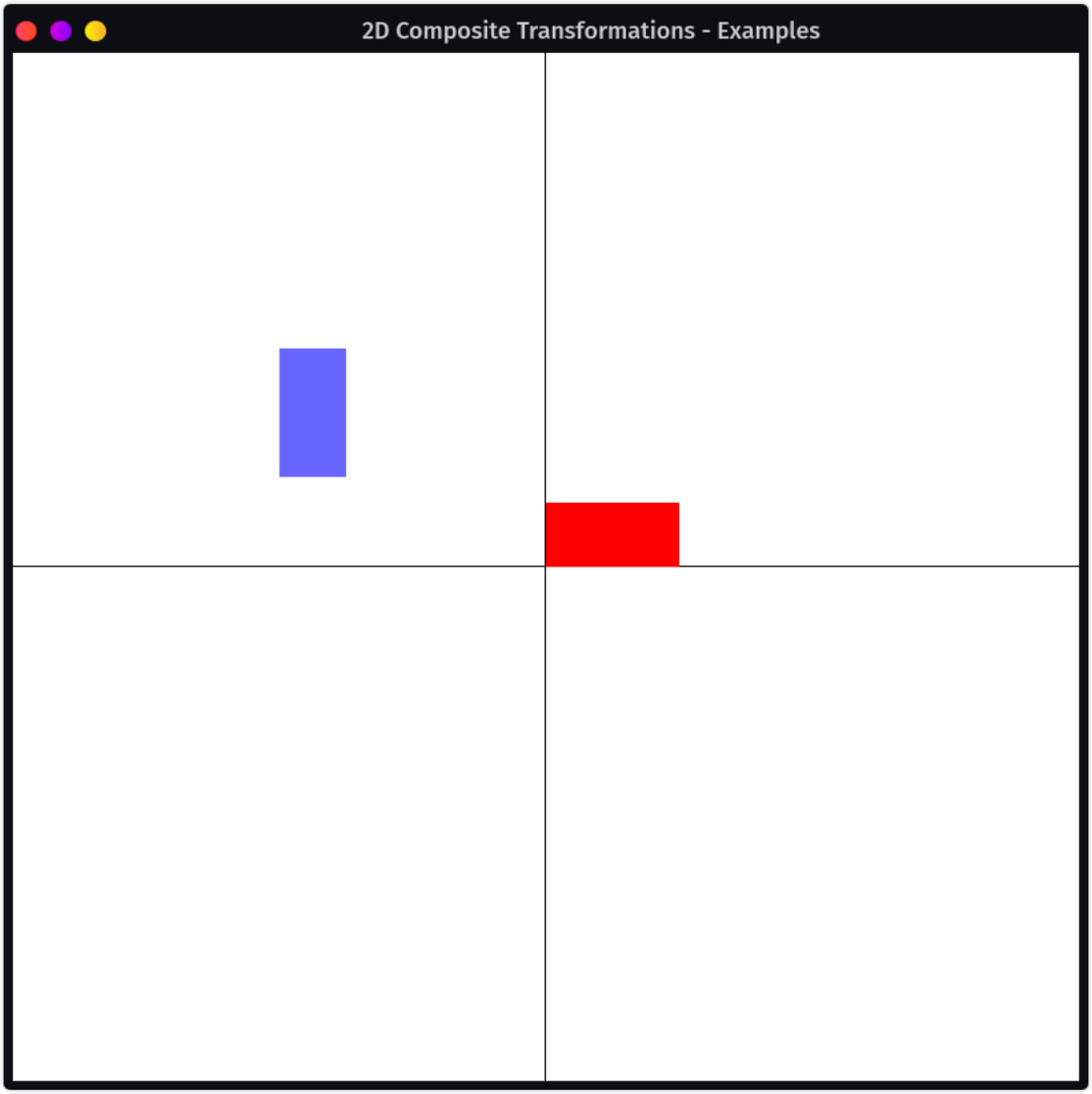
```
484                 int pivot = 0;
485                 cout << "\n\n------[ROTATION]------" << endl;
486                 cout << "\n\tEnter the Rotation Angle: ";
487                 cin >> rotationAngle;
488 
489                 cout << "\n\tEnter 1 for Rotating about Pivot, else enter
    0.";
490                 cout << "\n\t\tYour Choice -> "; cin >> pivot;
491 
492                 if(pivot){
493                     cout << "\n\tEnter Pivot Point: ";
494                     cout << "\n\t\tEnter X Coordinate: "; cin >> x;
495                     cout << "\n\t\tEnter Y Coordinate: "; cin >> y;
496                 }
497 
498                 polygon.setRotationMatrix(rotationAngle, Point(x, y, 1));
499                 cout << "\n\n------[ROTATION NOTED]------" << endl;
500                 break;
501             }
502 
503             case Scaling:{
504                 double xScale, yScale, xFixed = 0, yFixed = 0;
505                 int uniform = 0, fixed = 0;
506                 cout << "\n\n------[SCALING]------" << endl;
507                 cout << "\n\tEnter an option:";
508                 cout << "\n\t\t0 for Uniform Scaling";
509                 cout << "\n\t\t1 for Differential Scaling";
510                 cout << "\n\t\tYour Choice -> "; cin >> uniform;
511 
512                 if(uniform){
513                     cout << "\n\tEnter the Scaling Factors: ";
514                     cout << "\n\t\tX Scale Factor: "; cin >> xScale;
515                     cout << "\n\t\tY Scale Factor: "; cin >> yScale;
516                 } else{
517                     cout << "\n\tEnter the Scaling Factor: "; cin >>
    xScale;
518                     yScale = xScale;
519                 }
520 
521                 cout << "\n\tEnter 1 for Scaling about Fixed Point, else
    enter 0.";
522                 cout << "\n\t\tYour Choice -> "; cin >> fixed;
523 
524                 if(fixed){
525                     cout << "\n\tEnter Fixed Point: ";
526                     cout << "\n\t\tEnter X Coordinate: "; cin >> xFixed;
527                     cout << "\n\t\tEnter Y Coordinate: "; cin >> yFixed;
528                 }
529 
530                 polygon.setScaleMatrix(xScale, yScale, Point(xFixed,
    yFixed, 1));
```

```
531                 cout << "\n\n------[SCALING NOTED]------" << endl;
532                 break;
533             }
534 
535             case Reflection:{
536                 int reflectionOption = 4;
537                 cout << "\n\n------[REFLECTION]------" << endl;
538 
539                 while(reflectionOption < 0 || reflectionOption > 3){
540                     cout << "\n\tEnter an option:";
541                     cout << "\n\t\t0 for Reflection About X Axis";
542                     cout << "\n\t\t1 for Reflection About Y Axis.";
543                     cout << "\n\t\t2 for Reflection About Origin.";
544                     cout << "\n\t\t3 for Reflection About Line X = Y.";
545                     cout << "\n\t\tYour Choice -> "; cin >>
    reflectionOption;
546                 }
547 
548                 polygon.setReflectionMatrix(reflectionOption);
549 
550                 cout << "\n\n------[REFLECTION NOTED]------" << endl;
551                 break;
552             }
553 
554             case Shearing:{
555                 double shearParam, refConst = 0;
556                 int axis = 0, refLine = 0;
557                 cout << "\n\n------[SHEARING]------" << endl;
558 
559                 cout << "\n\tEnter an option:";
560                 cout << "\n\t\t0 for Shearing About X Axis";
561                 cout << "\n\t\t1 for Shearing About Y Axis";
562                 cout << "\n\t\tYour Choice -> "; cin >> axis;
563 
564                 cout << "\n\tEnter the Shearing Parameter: "; cin >>
    shearParam;
565 
566                 cout << "\n\tEnter 1 for Shearing About Reference Line,
    else enter 0.";
567                 cout << "\n\t\tYour Choice -> "; cin >> refLine;
568 
569                 if(refLine){
570                     if(!axis){
571                         cout << "\n\tEnter c for Ref. Line Y = c: ";
572                         cin >> refConst;
573                     } else{
574                         cout << "\n\tEnter c for Ref. Line X = c: ";
575                         cin >> refConst;
576                     }
577                 }
578 
```

```
                polygon.setShearMatrix(shearParam, axis, refConst);

                cout << "\n\n------[SHEARING NOTED]------" << endl;
                break;
            }
        }

        if(i == 1){
            cout << "\n\n------[TRANSFORMATION 1 NOTED]------" << endl;
        } else{
            cout << "\n\n------[TRANSFORMATION 2 NOTED]------" << endl;
        }
    }
}

void drawAxes(){
    //To draw the X and Y axes

    glColor3d(0, 0, 0); //Black color
    glBegin(GL_LINES);

    //X-axis
    glVertex2f(X_MIN, 0);
    glVertex2f(X_MAX, 0);

    //Y-axis
    glVertex2f(0, Y_MIN);
    glVertex2f(0, Y_MAX);

    glEnd();
}

void drawPolygon(PolygonShape polygon, bool transformed){
    //To draw a given polygon

    if(!transformed){
        glColor3d(1, 0, 0); //Red color
    } else{
        glColor4f(0, 0, 1, 0.6); //Blue Color
    }

    glBegin(GL_POLYGON);

    for(int i = 0; i < polygon.getVertexCount(); i++){
        Point p = polygon.getPoint(i);
        glVertex2f(p.getX(), p.getY());
    }

    glEnd();
}
```

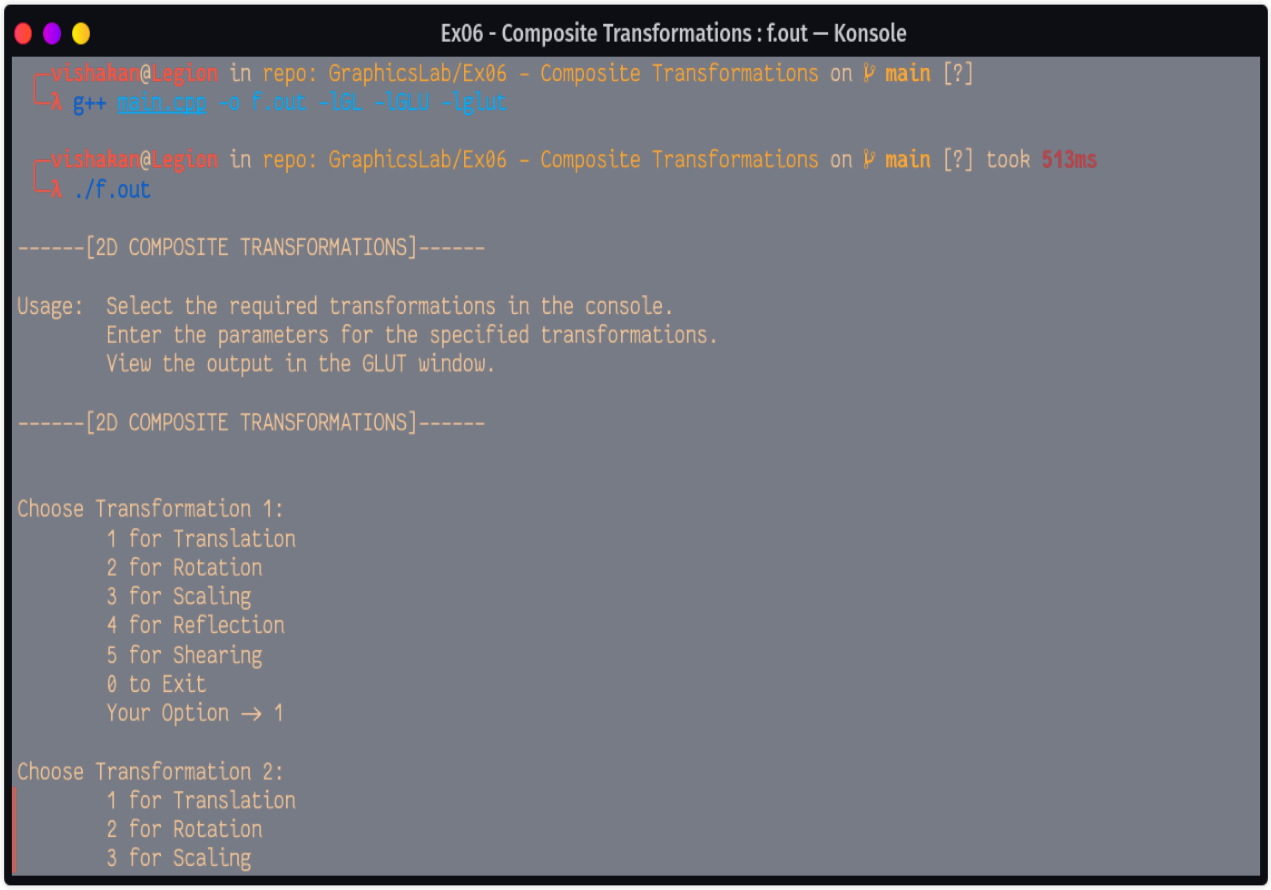
Output: Plot With Translation and Rotation

Figure 1: Plot With Translation and Rotation.



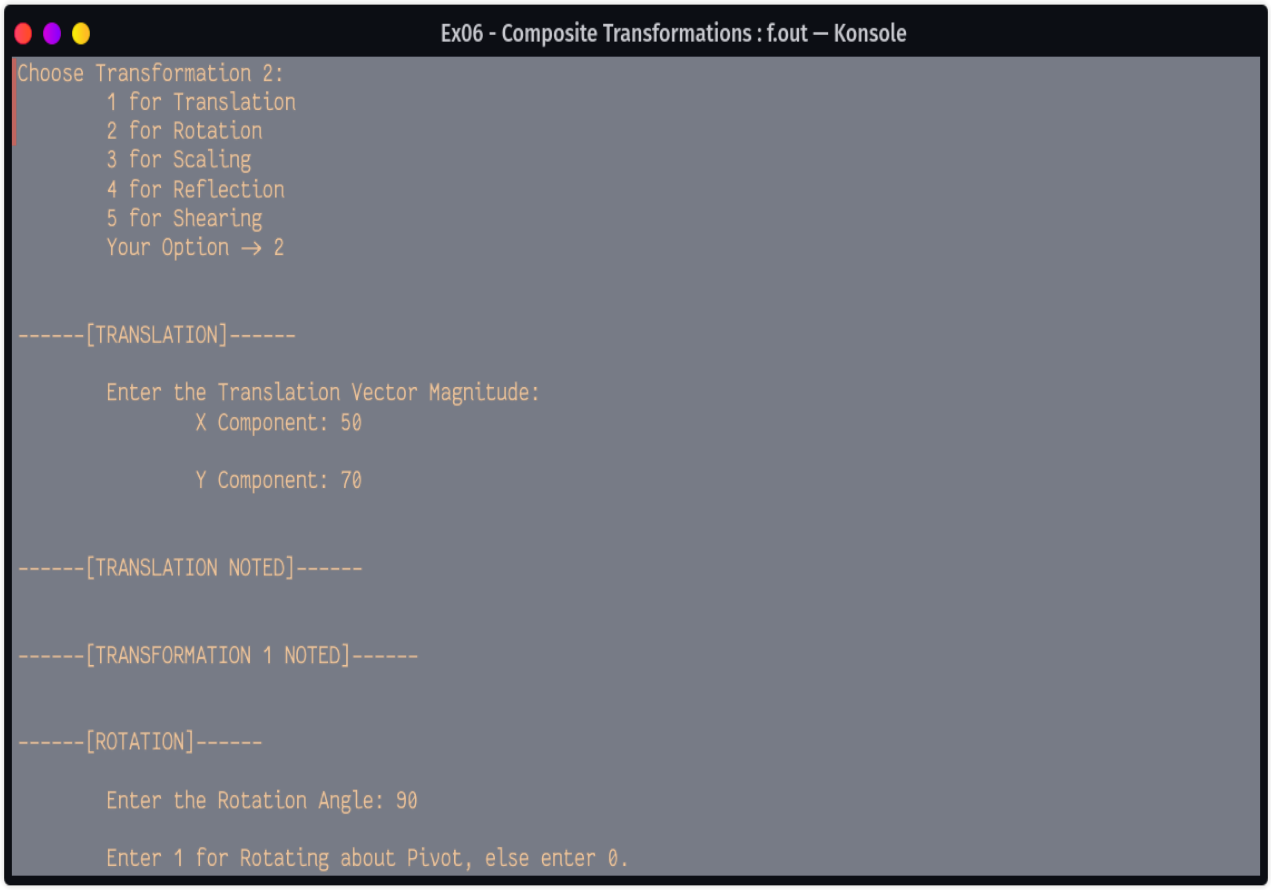
Output: Console

Figure 2: Output: Console.



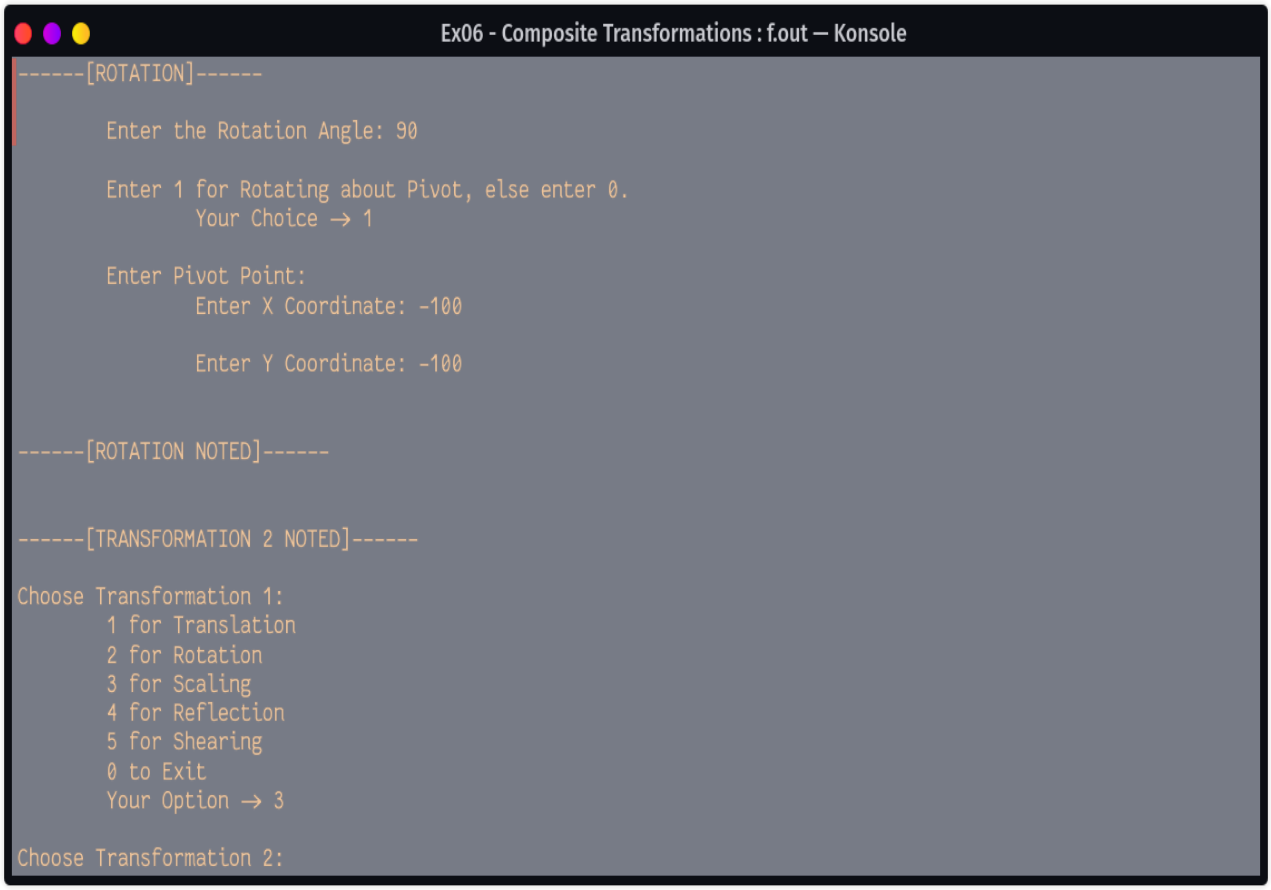
Output: Console

Figure 3: Output: Console.



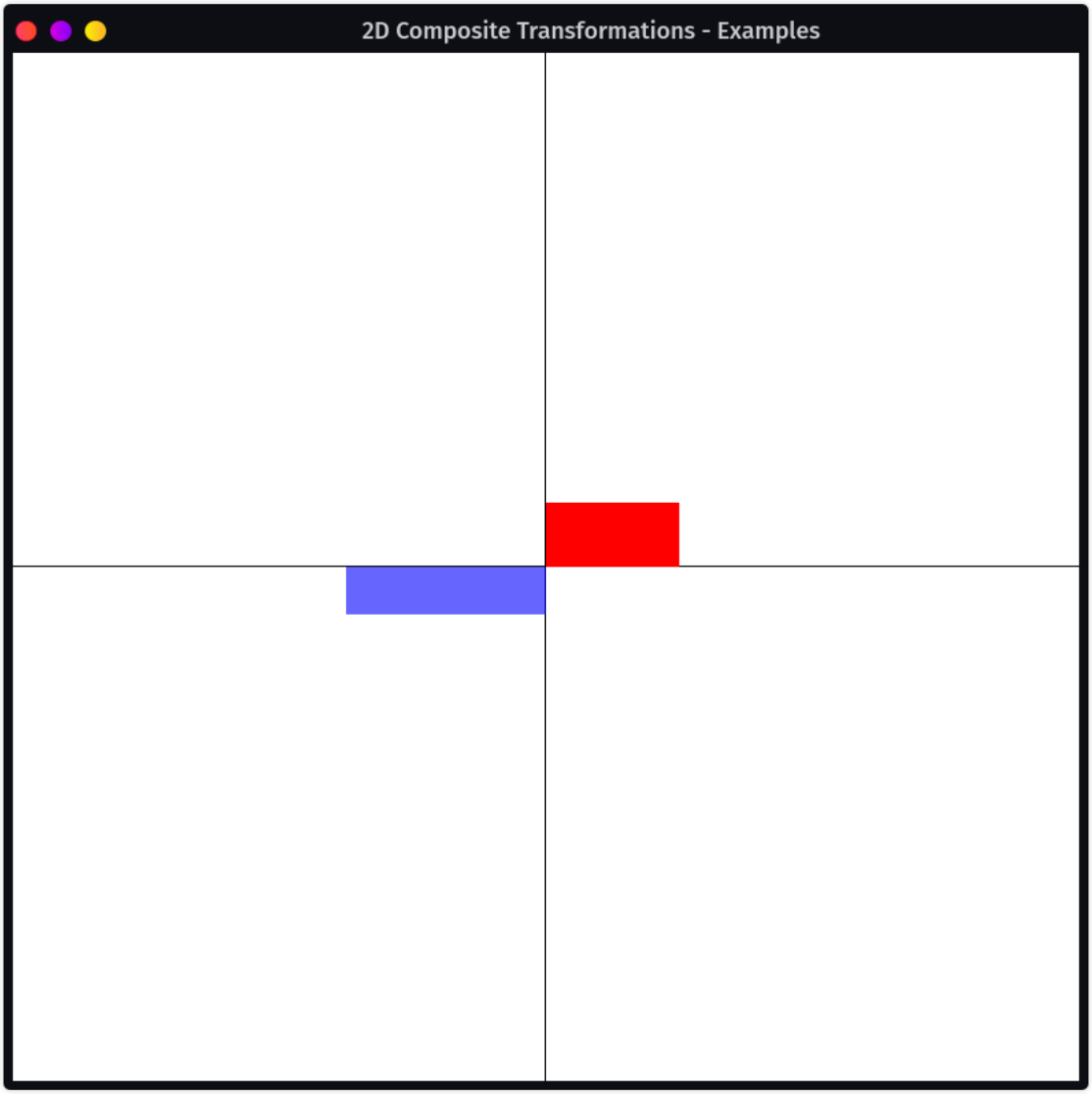
Output: Console

Figure 4: Output: Console.



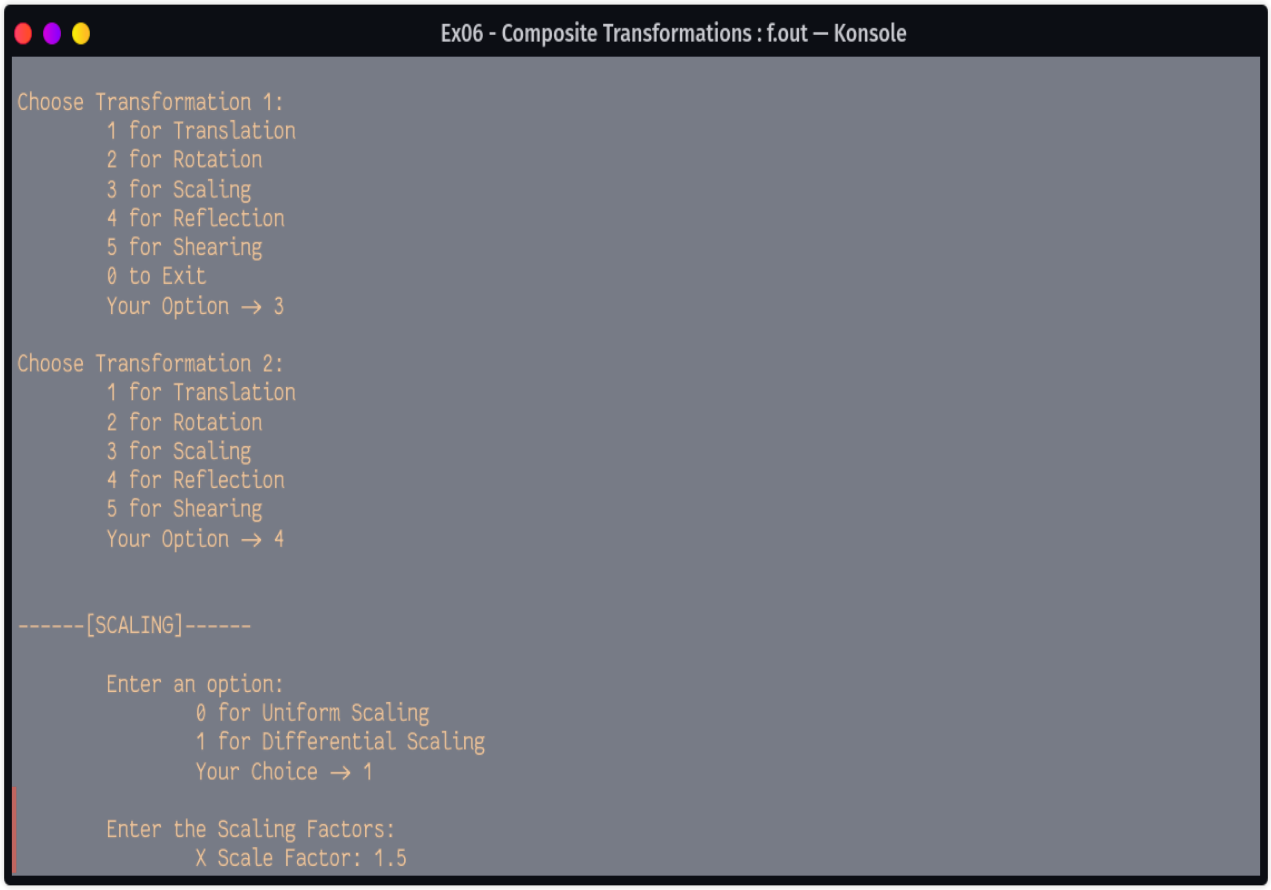
Output: Plot With Scaling and Reflection

Figure 5: Plot With Scaling and Reflection.



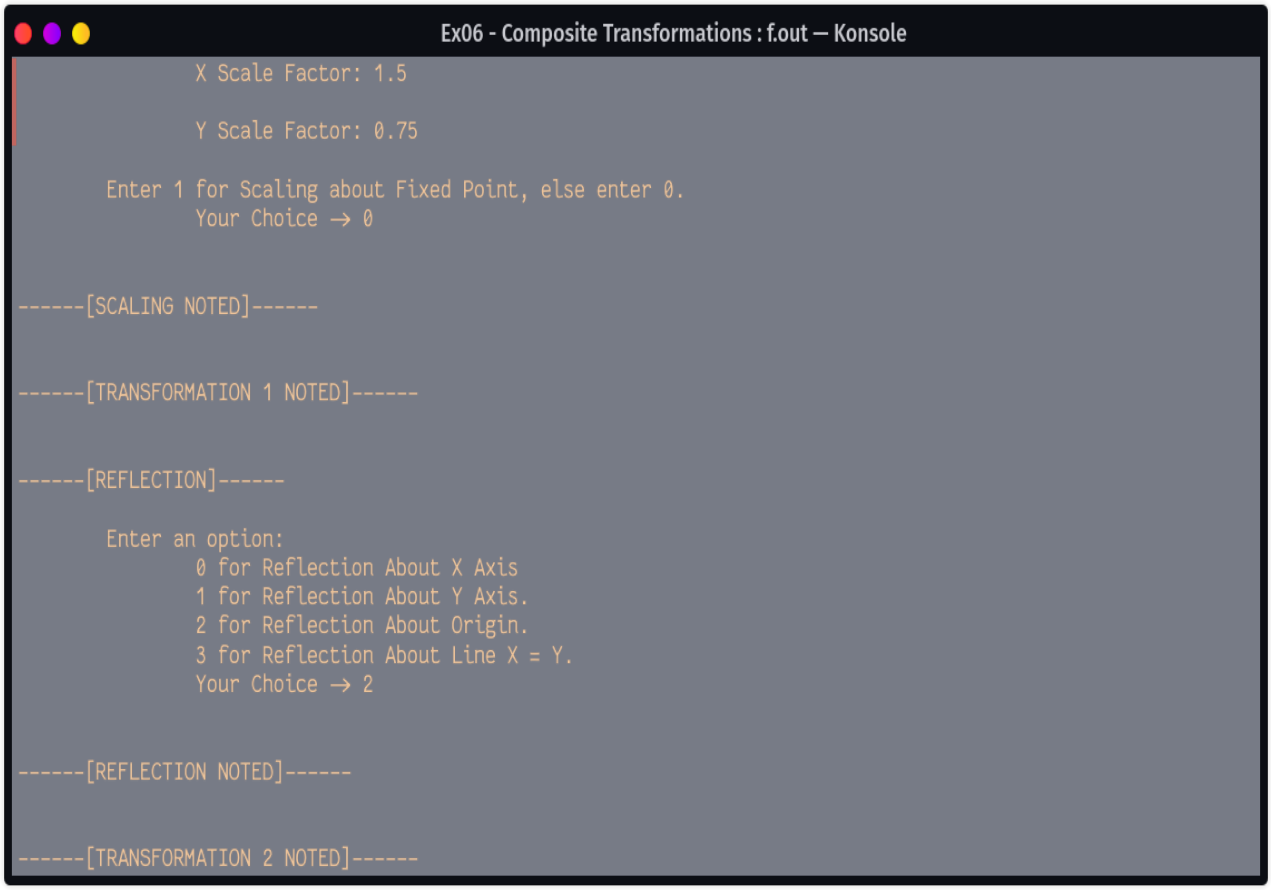
Output: Console

Figure 6: Console.



Output: Console

Figure 7: Console.



Code: Window To Viewport Transformation:

```
/*
Create a window with any 2D object and a different sized viewport. Apply window to viewport
transformation on the object. Display both window and viewport.
*/

#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
#include <iostream>     //for cin, cout
#include <cstring>      //for memcpy

using namespace std;

const int HEIGHT = 870;
const int WIDTH = 1010;
const int WINDOW_XMIN = -500;
const int WINDOW_XMAX = 0;
const int WINDOW_YMIN = -400;
const int WINDOW_YMAX = 400;
const int VIEWPORT_XMIN = 100;
const int VIEWPORT_XMAX = 500;
const int VIEWPORT_YMIN = -300;
const int VIEWPORT_YMAX = 300;

class Point{
private:
    GLdouble x, y, h;

public:
    Point(){
        x = y = 0;
        h = 1;
    }

    Point(GLint xCoord, GLint yCoord){
        x = xCoord;
        y = yCoord;
        h = 1;
    }

    Point(GLint xCoord, GLint yCoord, GLint H){
        x = xCoord;
        y = yCoord;
        h = H;
    }

```

```
47      void setCoords(GLdouble xCoord, GLdouble yCoord){
48          x = xCoord;
49          y = yCoord;
50      }
51
52      void setHomogeneousCoords(GLdouble xCoord, GLdouble yCoord, GLdouble H
        ){
53          x = xCoord;
54          y = yCoord;
55          h = H;
56      }
57
58      GLdouble getX() const{
59          return x;
60      }
61
62      GLdouble getY() const{
63          return y;
64      }
65
66      GLdouble getH() const{
67          return h;
68      }
69
70      GLdouble getHomogenousX() const{
71          return x * h;
72
73      }
74
75      GLdouble getHomogenousY() const{
76          return y * h;
77      }
78  };
79
80  class PolygonShape{
81  private:
82      int numVertices;
83      Point *points;
84      double transformMatrix[3][3];
85
86  public:
87      PolygonShape(){
88          numVertices = 0;
89      }
90
91      PolygonShape(int noVertices){
92          numVertices = noVertices;
93          points = new Point[numVertices];
94      }
95
96      int getVertexCount() const{
```

```
97          return numVertices;
98      }
99
100     Point getPoint(int i){
101         return points[i];
102     }
103
104     void setVertices(int noVertices){
105         numVertices = noVertices;
106         points = new Point[numVertices];
107     }
108
109     void setPoint(int i, GLdouble x, GLdouble y, GLdouble h = 1){
110         points[i].setHomogeneousCoords(x, y, h);
111     }
112
113     void setTransformMatrix(){
114         //Perform Window->Viewport Transformation using Translation and
    Scaling
115
116         double xShift = VIEWPORT_XMIN - WINDOW_XMIN;
117         double yShift = VIEWPORT_YMIN - WINDOW_YMIN;
118
119         double translateMatrix[3][3] = {    {1, 0, xShift},
120                                             {0, 1, yShift},
121                                             {0, 0, 1}};
122
123         double xScale = (double) (VIEWPORT_XMAX - VIEWPORT_XMIN) / (
    WINDOW_XMAX - WINDOW_XMIN);
124         double yScale = (double) (VIEWPORT_YMAX - VIEWPORT_YMIN) / (
    WINDOW_YMAX - WINDOW_YMIN);
125
126         Point pivot(VIEWPORT_XMIN, VIEWPORT_YMIN);
127
128         double scaleMatrix[3][3] = {    {xScale, 0, pivot.getHomogenousX()
     * (1 - xScale)},
129                                         {0, yScale, pivot.getHomogenousY()
     * (1 - yScale)},
130                                         {0, 0, 1}};
131
132         double product = 0;
133
134         //Composite Transformation = Scaling * Translation
135         for(int i = 0; i < 3; i++){
136             for(int j = 0; j < 3; j++){
137                 product = 0;
138
139                 for(int k = 0; k < 3; k++){
140                     product += scaleMatrix[i][k] * translateMatrix[k][j];
141                 }
142
```

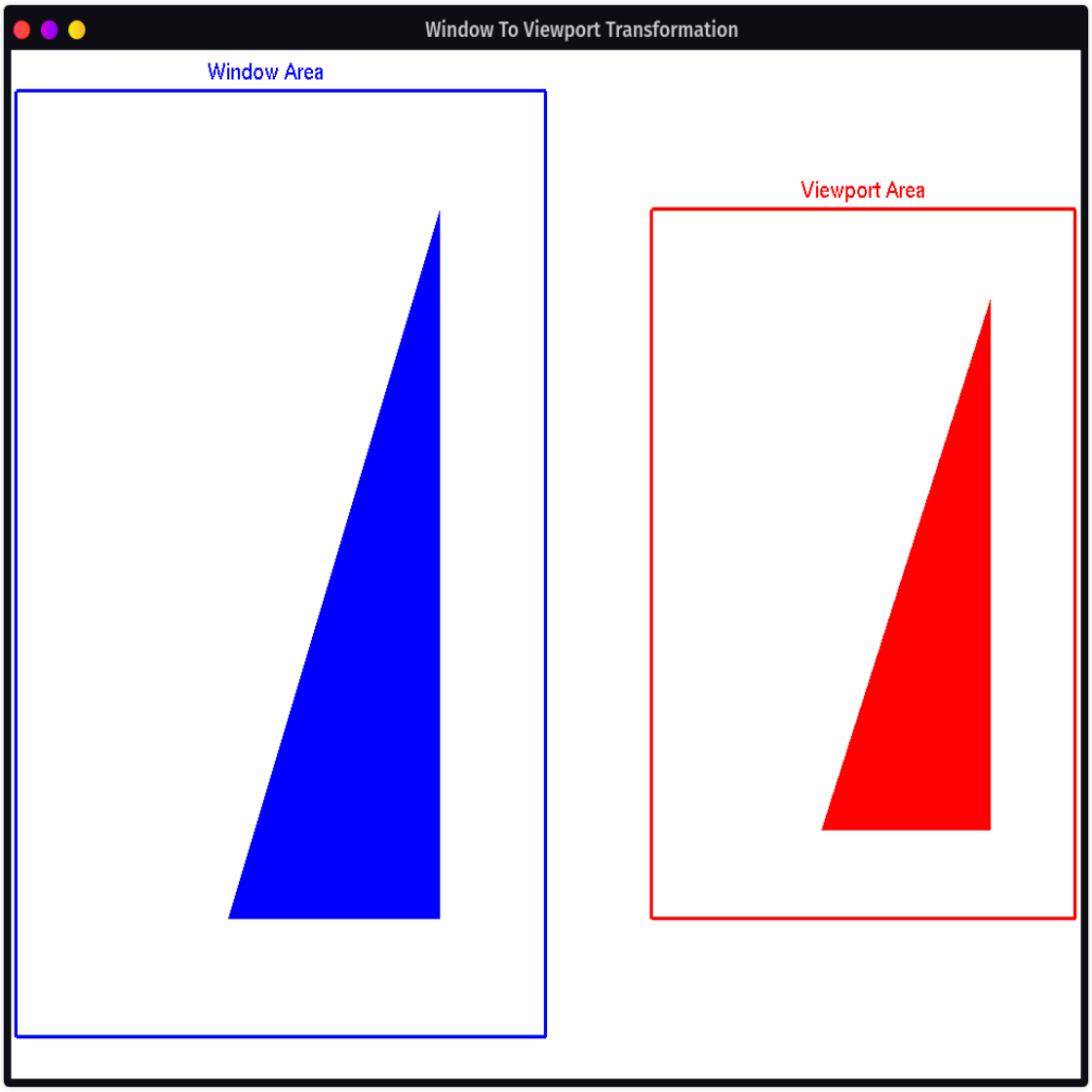
```
                transformMatrix[i][j] = product;
            }
        }
    }

    PolygonShape getViewportPolygon(){

        PolygonShape polyDash(numVertices);
        double values[3];

        for(int i = 0; i < numVertices; i++){
            Point p = getPoint(i);

            //[3 x 3] x [3 x 1] = [3 x 1] matrix
            for(int j = 0; j < 3; j++){
                values[j] = transformMatrix[j][0] * p.getHomogenousX() +
                            transformMatrix[j][1] * p.getHomogenousY() +
                            transformMatrix[j][2] * p.getH();
            }

            polyDash.setPoint(i, values[0]/p.getH(), values[1]/p.getH(),
    values[2]);
        }

        return polyDash;
    }
};


void initializeDisplay();
void plotComponents();
void plotBoundaries();
void plotPolygon(PolygonShape polygon, bool transformed = false);

PolygonShape polygon;

int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(WIDTH, HEIGHT);
    glutCreateWindow("Window To Viewport Transformation");

    polygon.setVertices(3);
    polygon.setPoint(0, -300, -300);
    polygon.setPoint(1, -100, -300);
    polygon.setPoint(2, -100, 300);
    //polygon.setPoint(3, -300, 300);

    initializeDisplay();
    glutDisplayFunc(plotComponents);
```

```
193
194     glutMainLoop();
195
196     return 1;
197 }
198
199 void initializeDisplay(){
200     //Initialize the display parameters
201
202     glClearColor(1, 1, 1, 0);
203     glMatrixMode(GL_PROJECTION);
204     gluOrtho2D(-WIDTH/2, WIDTH/2, -HEIGHT/2, HEIGHT/2);
205     glClear(GL_COLOR_BUFFER_BIT);
206 }
207
208 void plotComponents(){
209     //Plot the window, viewport and transformations
210
211     plotBoundaries();
212     plotPolygon(polygon);
213     polygon.setTransformMatrix();
214     PolygonShape polygonDash = polygon.getViewportPolygon();
215     plotPolygon(polygonDash, true);
216 }
217
218 void plotBoundaries(){
219     //Plot the window and viewport boundaries
220
221     glLineWidth(3);
222
223     //Title of window area
224     glColor3d(0, 0, 1); //Blue color
225     unsigned char windowString[] = "Window Area";
226     glutBitmapLength(GLUT_BITMAP_HELVETICA_18, windowString);
227     glRasterPos2d(-320, 410);
228
229     for(int i = 0; i < strlen((const char *)windowString); i++) {
230         glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, windowString[i]);
231     }
232
233     //Plot the window area
234     glBegin(GL_LINE_LOOP);
235     glVertex2f(WINDOW_XMIN, WINDOW_YMIN);
236     glVertex2f(WINDOW_XMAX, WINDOW_YMIN);
237     glVertex2f(WINDOW_XMAX, WINDOW_YMAX);
238     glVertex2f(WINDOW_XMIN, WINDOW_YMAX);
239     glEnd();
240
241     //Title of viewport area
242     glColor3d(1, 0, 0); //Red color
243     unsigned char viewportString[] = "Viewport Area";
```

```
244     glutBitmapLength(GLUT_BITMAP_HELVETICA_18, viewportString);
245     glRasterPos2d(240, 310);
246 
247     for(int i = 0; i < strlen((const char *)viewportString); i++) {
248         glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, viewportString[i]);
249     }
250 
251     //Plot the viewport area
252     glBegin(GL_LINE_LOOP);
253     glVertex2f(VIEWPORT_XMIN, VIEWPORT_YMIN);
254     glVertex2f(VIEWPORT_XMAX, VIEWPORT_YMIN);
255     glVertex2f(VIEWPORT_XMAX, VIEWPORT_YMAX);
256     glVertex2f(VIEWPORT_XMIN, VIEWPORT_YMAX);
257     glEnd();
258 
259     glFlush();
260 }
261 
262 void plotPolygon(PolygonShape polygon, bool transformed){
263     //To draw a given polygon
264 
265     if(!transformed){
266         glColor3d(0, 0, 1); //Blue color
267     } else{
268         glColor3d(1, 0, 0); //Red Color
269     }
270 
271     glBegin(GL_POLYGON);
272 
273     for(int i = 0; i < polygon.getVertexCount(); i++){
274         Point p = polygon.getPoint(i);
275         glVertex2f(p.getX(), p.getY());
276     }
277 
278     glEnd();
279 }
```

Output: Window To Viewport Transformation

Figure 8: Window To Viewport Transformation.



Learning Outcome:

- I understood how to perform **composite transformations** using OpenGL and C++ programming.
- I learnt to perform proper matrix multiplication of the base transformation matrices to get the appropriate composite transformation matrix.
- I applied the composite transformation on the base polygon and displayed the transformed polygon on the graph window.
- I learnt how to overcome the screen refreshing issue/asynchronous event handling while getting user I/O with the help of **glutTimerFunc()** and setting a **60 FPS** refresh rate.
- I was able to perform composite transformations based on translation, rotation, scaling, reflection and shearing.
- I learnt about **Window to Viewport Transformation**.
- I learnt how to set up 2 windows, each to simulate a window and a viewport output window for the purpose of demonstrating the transformation.
- I learnt to perform the transformation from window to viewport using **Translation & Scaling** transformations.
- I learnt the formula for performing the specific translation and scaling required for viewport transformation.
- I understood how to display raster text using the **glutBitmapCharacter() and glRasterPos2d()** methods.
- I refreshed my C/C++ concepts regarding **cin, cout and memcpy()** methods.
- I learnt how to configure **OpenGL** in **Linux**.

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

29 September 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 7: Cohen-Sutherland Line Clipping in C++ Using OpenGL

Aim:

Apply Cohen-Sutherland line clipping on a line $(x_1, y_1)(x_2, y_2)$ with respect to a clipping window $(X_W min, Y_W min)(X_W max, Y_W max)$.

After clipping with respect to an edge, display the line segment with the calculated intermediate intersection points and the vertex list.

Input: The clipping window co-ordinates and the line endpoints.

Note: The output should show the clipping window and the line to be clipped in different colors.

You can show the intermediate steps using time delay.

Code: Cohen-Sutherland Algorithm:

```
/*
To perform the Cohen-Sutherland Line Clipping Algorithm on a given line,
based upon a rectangular clipping window
*/

#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
#include <iostream>
#include <cstring>

using namespace std;

const int LeftBit = 0x1;
const int RightBit = 0x2;
const int BottomBit = 0x4;
const int TopBit = 0x8;

const int WINDOW_WIDTH = 800;
const int WINDOW_HEIGHT = 800;
const int FPS = 60;

class Point
{
private:
    GLfloat x, y;

public:
    Point()
    {
        x = y = 0;
    }

    Point(GLfloat X, GLfloat Y)
    {
        x = X;
        y = Y;
    }

    GLfloat getX()
    {
        return x;
    }

    GLfloat getY()
    {
        return y;
```

```
48      }
49
50      void setX(GLfloat X)
51      {
52          x = X;
53      }
54
55      void setY(GLfloat Y)
56      {
57          y = Y;
58      }
59
60      int encode(Point windowMin, Point windowMax)
61      {
62          int RC = 0x00;
63
64          if (x < windowMin.getX())
65          {
66              RC = RC | LeftBit;
67          }
68          if (x > windowMax.getX())
69          {
70              RC = RC | RightBit;
71          }
72          if (y < windowMin.getY())
73          {
74              RC = RC | BottomBit;
75          }
76          if (y > windowMax.getY())
77          {
78              RC = RC | TopBit;
79          }
80
81          return RC;
82      }
83  };
84
85  class Line
86  {
87  private:
88      Point p, q;
89
90  public:
91      Line()
92      {
93          p.setX(0); p.setY(0);
94          q.setX(0); q.setY(0);
95      }
96
97      Line(float x1, float y1, float x2, float y2)
98      {
```

```
99          p.setX(x1); p.setY(y1);
100         q.setX(x2); q.setY(y2);
101     }
102
103     Point getP()
104     {
105         return p;
106     }
107
108     Point getQ()
109     {
110         return q;
111     }
112
113     void setP(float x, float y)
114     {
115         p.setX(x); p.setY(y);
116     }
117
118     void setQ(float x, float y)
119     {
120         q.setX(x); q.setY(y);
121     }
122
123     int isInside(int RC)
124     {
125         return !RC;
126     }
127
128     int trivialReject(int RC1, int RC2)
129     {
130         return (RC1 & RC2);
131     }
132
133     int trivialAccept(int RC1, int RC2)
134     {
135         return (!(RC1 | RC2));
136     }
137
138     void swapPoints()
139     {
140         Point x;
141         x.setX(p.getX()); x.setY(p.getY());
142         p.setX(q.getX()); p.setY(q.getY());
143         q.setX(x.getX()); q.setY(x.getY());
144     }
145
146     bool lineClipCohenSutherland(Point windowMin, Point windowMax)
147     {
148         int RC1, RC2;
149         int plotLine = false, done = false;
```

```
150         GLfloat m;
151
152         while (!done)
153         {
154             RC1 = p.encode(windowMin, windowMax);
155             RC2 = q.encode(windowMin, windowMax);
156
157             //cout << "RC1: " << RC1 << "RC2: " << RC2 << endl;
158
159             if (trivialAccept(RC1, RC2))
160             {
161                 //Line coordinates are inside boundary
162                 done = true;
163                 continue;
164             }
165
166             if (trivialReject(RC1, RC2))
167             {
168                 //Line coordinates are outside boundary
169                 done = true;
170                 continue;
171             }
172
173             plotLine = true;    //Clipped Line needs to be highlighted
174
175             if (isInside(RC1))
176             {
177                 //If P is inside, then swap P with Q.
178                 swapPoints();
179                 RC1 = p.encode(windowMin, windowMax);
180                 RC2 = q.encode(windowMin, windowMax);
181             }
182
183             if (q.getX() != p.getX())
184             {
185                 //Avoid dx = 0 case
186                 m = (q.getY() - p.getY()) / (q.getX() - p.getX());
187             }
188
189             if (RC1 & LeftBit)
190             {
191                 p.setY(p.getY() + (windowMin.getX() - p.getX()) * m);
192                 p.setX(windowMin.getX());
193             }
194
195             else if (RC1 & RightBit)
196             {
197                 p.setY(p.getY() + (windowMax.getX() - p.getX()) * m);
198                 p.setX(windowMax.getX());
199             }
200
```

```
            else if (RC1 & BottomBit)
            {
                if (p.getX() != q.getX())
                {
                    p.setX(p.getX() + (windowMin.getY() - p.getY()) / m);
                    p.setY(windowMin.getY());
                }
            }

            else if (RC1 & TopBit)
            {
                if (p.getX() != q.getX())
                {
                    p.setX(p.getX() + (windowMax.getY() - p.getY()) / m);
                    p.setY(windowMax.getY());
                }
            }
        }

        return plotLine;
    }
};

void dummyFunction();
void mainLoop(int val);
void initializeDisplay();
void drawLine(Point p, Point q, bool clip = false);
void drawClippingWindow(Point windowMin, Point windowMax);
void getParams();

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("Cohen Sutherland Line Clipping Algorithm");

    cout << "\n\t\t-----[COHEN SUTHERLAND LINE CLIPPING ALGORITHM]-----\n"
    ;

    initializeDisplay();
    glutDisplayFunc(dummyFunction);
    glutTimerFunc(1000/FPS, mainLoop, 0);
    glutMainLoop();

    return 1;
}

void mainLoop(int val)
{
```

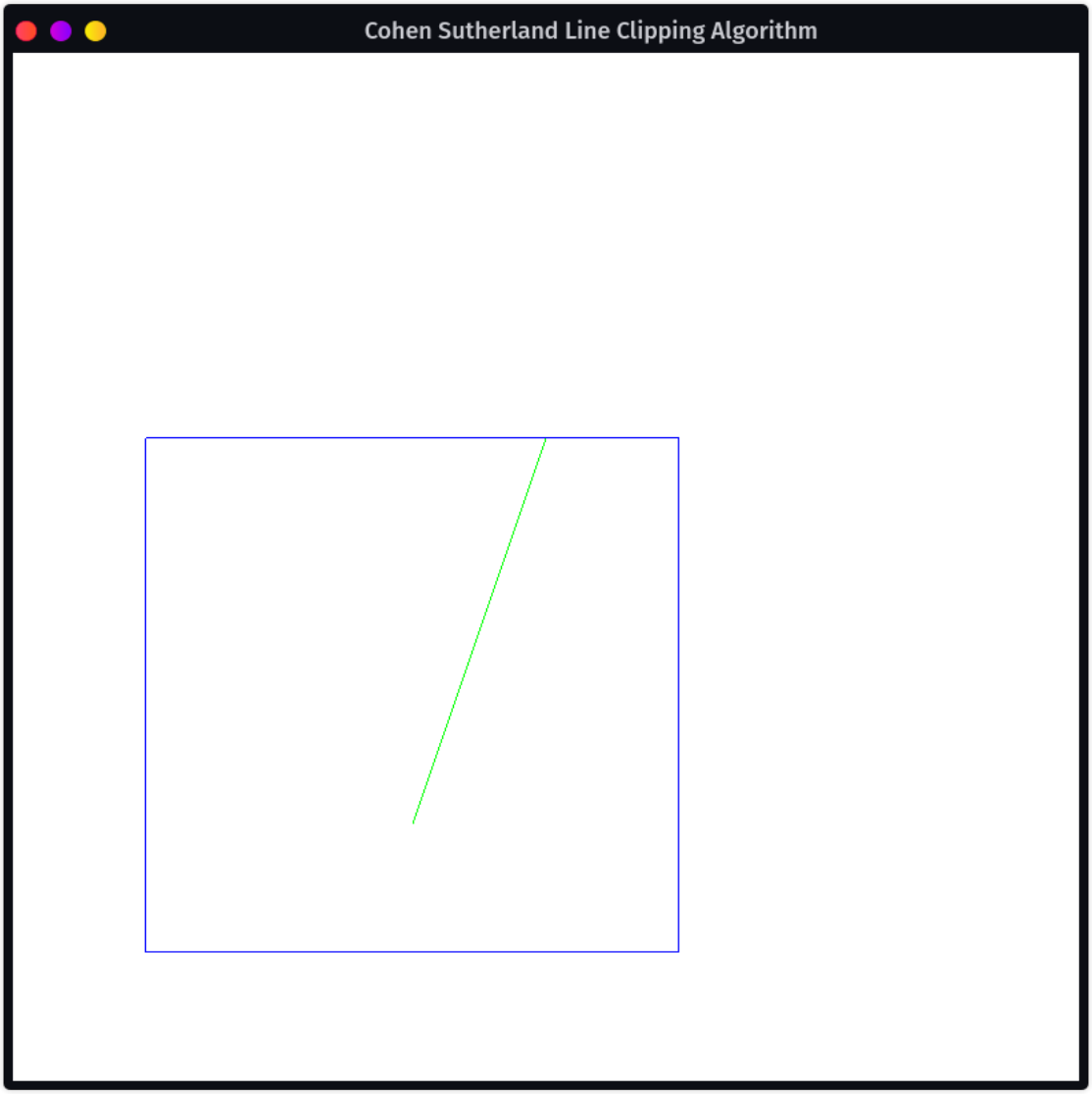
```
251     //Render the display using the timer function
252     getParams();
253 }
254 
255 void dummyFunction()
256 {
257     //Placeholder function
258 }
259 
260 void initializeDisplay()
261 {
262     //Initialize the display parameters
263 
264     glClearColor(1, 1, 1, 0);
265     glMatrixMode(GL_PROJECTION);
266     gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HEIGHT);
267     glClear(GL_COLOR_BUFFER_BIT);
268 }
269 
270 void getParams()
271 {
272     //To get user parameters for the line clipping process
273 
274     Point windowMin = Point(100, 100), windowMax = Point(500, 500);
275     Line lineSegment = Line(300, 200, 400, 500);
276 
277     int option = 0;
278 
279     drawClippingWindow(windowMin, windowMax);
280     drawLine(lineSegment.getP(), lineSegment.getQ());
281     glFlush();
282 
283 
284     while (true)
285     {
286         //Await user input
287         cout << "\n\t1. Set Line Coordinates" << endl;
288         cout << "\t2. Set Clipping Window" << endl;
289         cout << "\t0. Exit" << endl;
290         cout << "\tYour Option -> ";
291         cin >> option;
292 
293         if (!option)
294         {
295             cout << "\n\t\t-----[COHEN SUTHERLAND LINE CLIPPING ALGORITHM
    ]-----\n";
296             exit(0);
297         }
298 
299         if (option == 1)
300         {
```

```
301             float x, y;
302 
303             cout << "\n\n\tEnter Point P:" << endl;
304             cout << "\t\tEnter X: ";
305             cin >> x;
306             cout << "\t\tEnter Y: ";
307             cin >> y;
308             lineSegment.setP(x, y);
309 
310             cout << "\n\n\tEnter Point Q:" << endl;
311             cout << "\t\tEnter X: ";
312             cin >> x;
313             cout << "\t\tEnter Y: ";
314             cin >> y;
315             lineSegment.setQ(x, y);
316         }
317 
318         if (option == 2)
319         {
320             float x, y;
321             cout << "\n\n\tEnter Window Minimums:" << endl;
322             cout << "\t\tEnter X: ";
323             cin >> x;
324             cout << "\t\tEnter Y: ";
325             cin >> y;
326             windowMin.setX(x);
327             windowMin.setY(y);
328 
329             cout << "\n\n\tEnter Window Maximums:" << endl;
330             cout << "\t\tEnter X: ";
331             cin >> x;
332             cout << "\t\tEnter Y: ";
333             cin >> y;
334             windowMax.setX(x);
335             windowMax.setY(y);
336 
337             glClear(GL_COLOR_BUFFER_BIT);   //Clear the display window
338         }
339 
340         drawClippingWindow(windowMin, windowMax);
341         drawLine(lineSegment.getP(), lineSegment.getQ());
342         glFlush();
343 
344         bool plotLine = lineSegment.lineClipCohenSutherland(windowMin,
    windowMax);
345 
346         if (plotLine)
347         {
348             drawLine(lineSegment.getP(), lineSegment.getQ(), true);
349         }
350 
```

```
351             glFlush();
352         }
353 }
354
355 void drawLine(Point p, Point q, bool clip)
356 {
357     glBegin(GL_LINES);
358
359     if (clip)
360     {
361         glColor3d(1, 0, 0);
362     }
363     else
364     {
365         glColor3d(0, 1, 0);
366     }
367
368     glVertex2f(p.getX(), p.getY());
369     glVertex2f(q.getX(), q.getY());
370
371     glEnd();
372     glFlush();
373 }
374
375 void drawClippingWindow(Point windowMin, Point windowMax)
376 {
377     glBegin(GL_LINE_LOOP);
378     glColor3d(0, 0, 1);
379
380     glVertex2f(windowMin.getX(), windowMin.getY());
381     glVertex2f(windowMax.getX(), windowMin.getY());
382     glVertex2f(windowMax.getX(), windowMax.getY());
383     glVertex2f(windowMin.getX(), windowMax.getY());
384
385     glEnd();
386     glFlush();
387 }
```

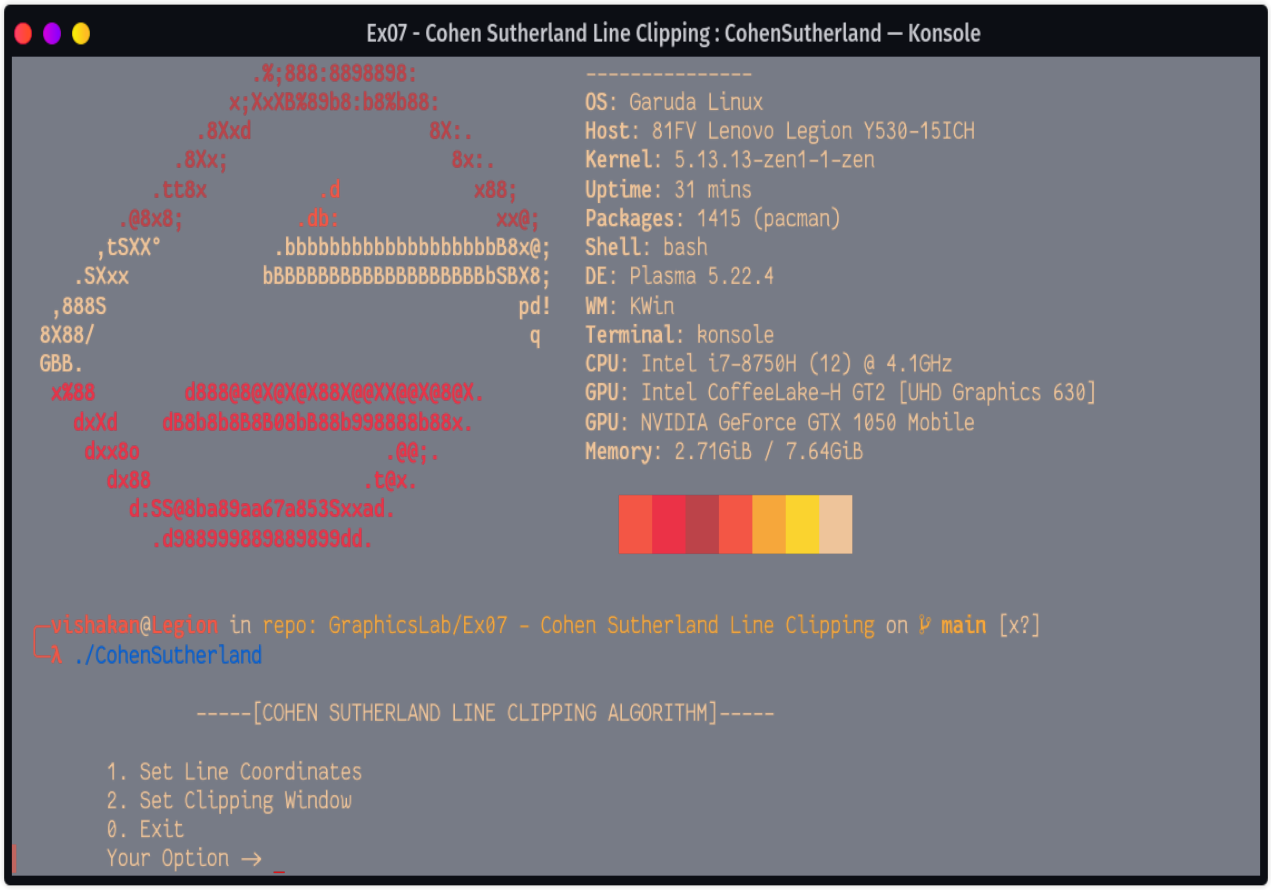
Output: Default Line & Clipping Window

Figure 1: Default Line & Clipping Window



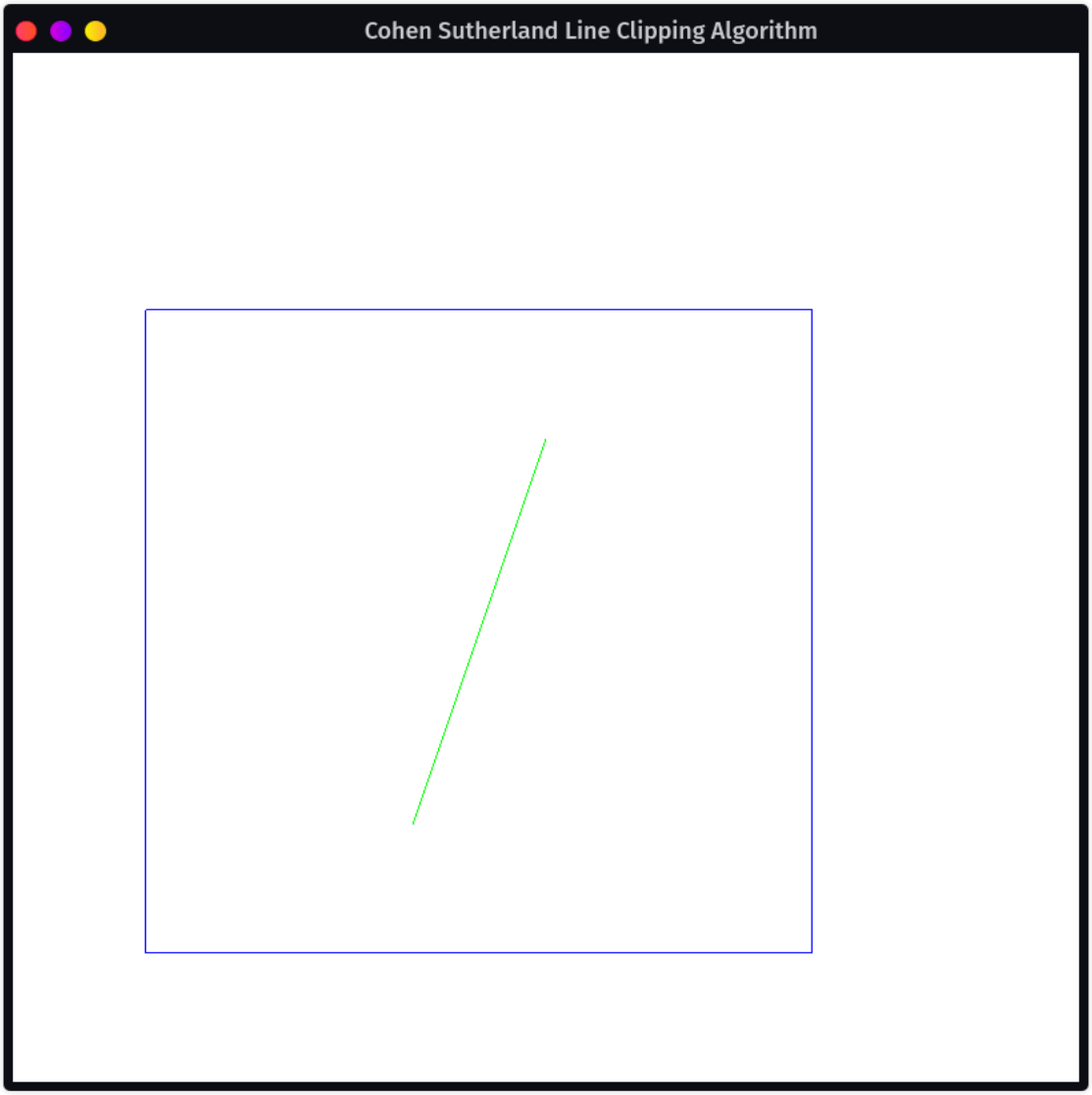
Output: Console, Window[(100, 100), (500, 500)]

Figure 2: Output: Console, Window[(100, 100), (500, 500)]



Output: Window[(100, 100), (600, 600)]

Figure 3: Output: Window[(100, 100), (600, 600)].

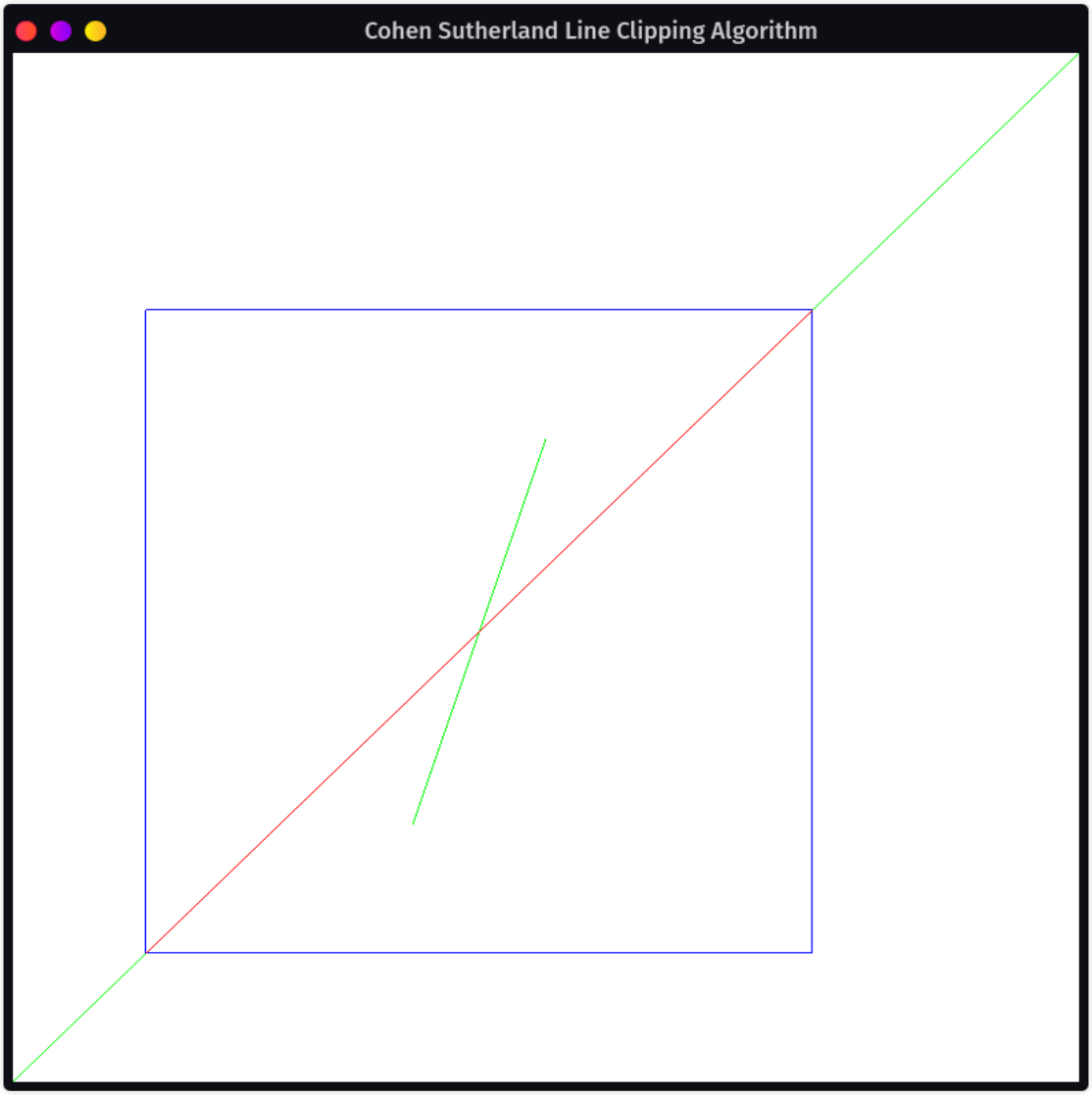


Output: Console, Window[(100, 100), (600, 600)]

Figure 4: Output: Console, Window[(100, 100), (600, 600)].

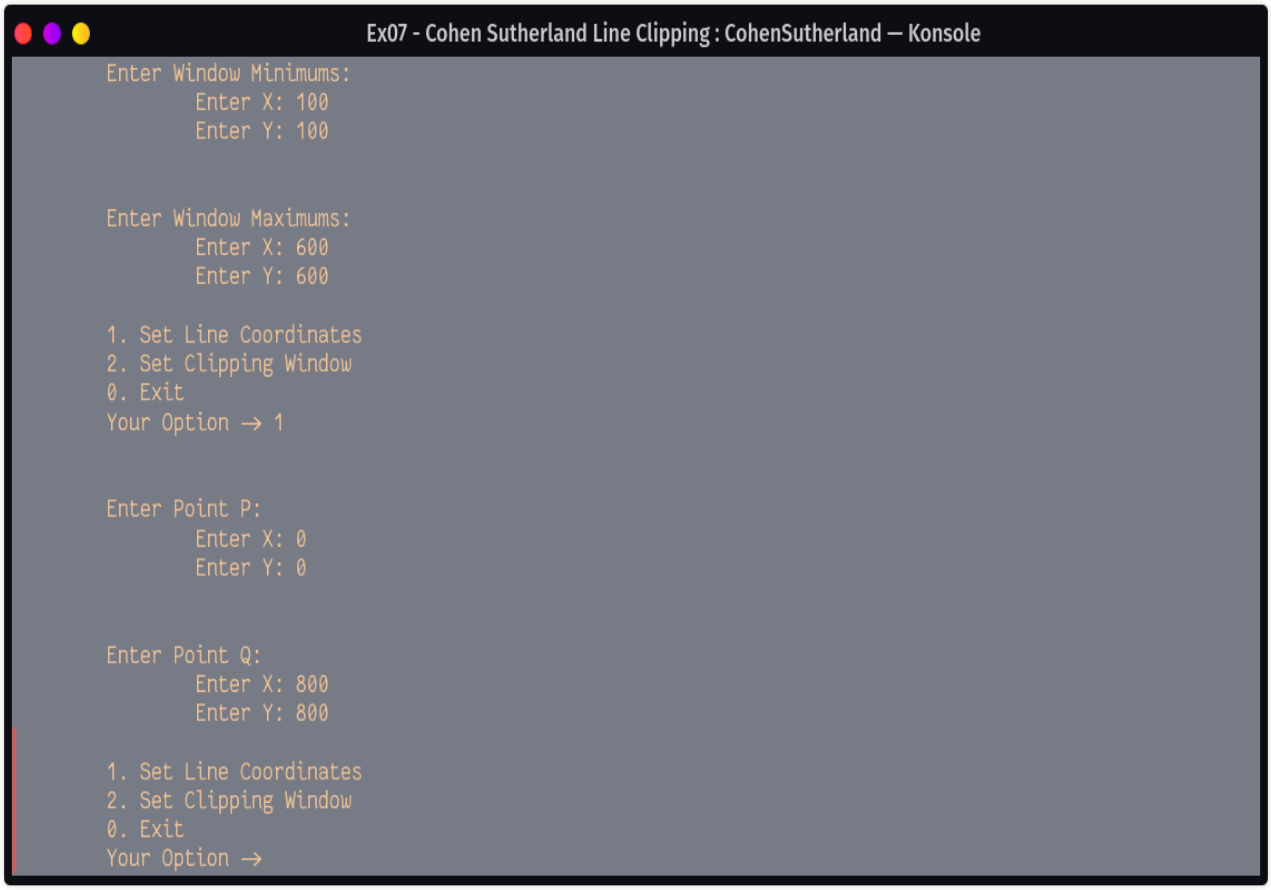
Output: Line[(0, 0), (800, 800)] & Clipping

Figure 5: Output: Line[(0, 0), (800, 800)] & Clipping.



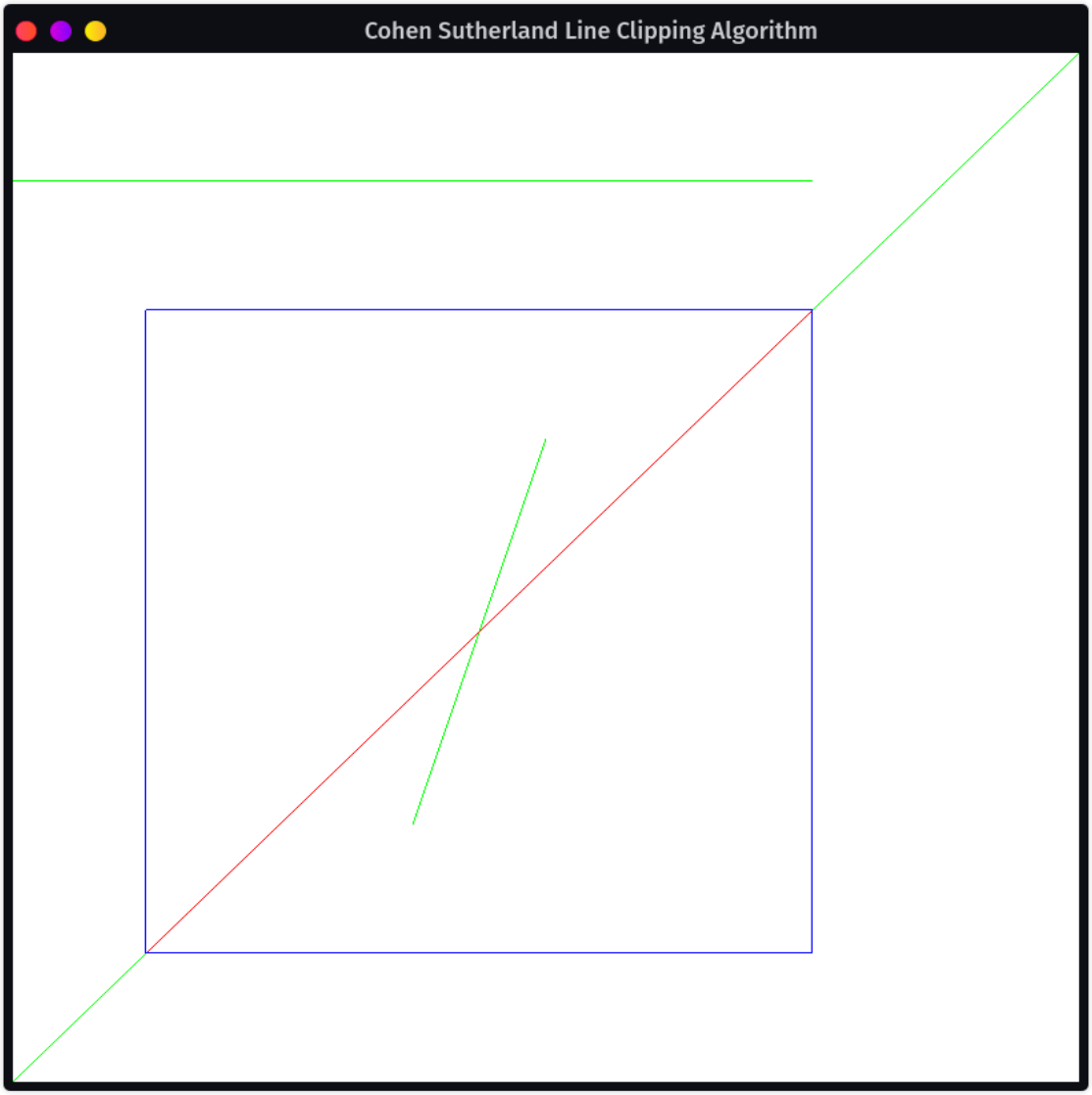
Output: Console, Line[(0, 0), (800, 800)] & Clipping

Figure 6: Output: Console, Line[(0, 0), (800, 800)] & Clipping.



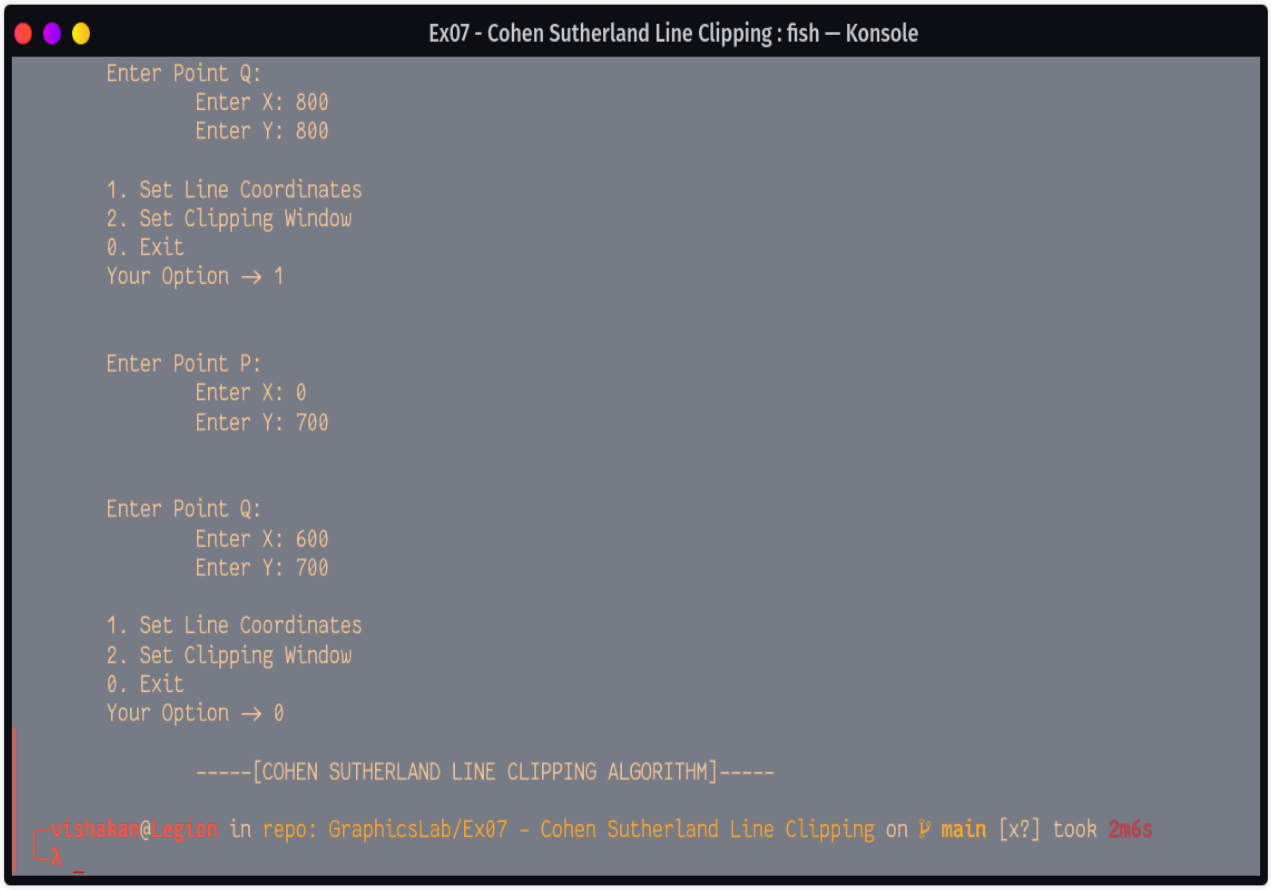
Output: Line[(0, 700), (600, 700)] & Clipping

Figure 7: Output: Line[(0, 700), (600, 700)] & Clipping.



Output: Console, Line[(0, 700), (600, 700)] & Clipping

Figure 8: Output: Console, Line[(0, 700), (600, 700)] & Clipping.



Learning Outcome:

- I learnt about **Cohen-Sutherland Line Clipping Algorithm**'s theoretical basis.
- I learnt how to compute and program **Region Codes** for a given point and clipping window.
- I understood about the advantages and pitfalls of the Cohen-Sutherland Algorithm.
- I understood how to implement **trivial accept & reject** conditions in the algorithm.
- I was able to implement the algorithm for any given line and a rectangular clipping window region.
- I demonstrated the clipped line, the original line and the clipping window using different colors.
- I handled corner cases where $dx = 0$ and thus avoiding divide by zero errors in slope calculation.
- I understood how to swap the points and continue clipping till region codes of both points become 0 for trivial acceptance.

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

14 September 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 8: 3D Transformations in C++ using OpenGL

Aim:

To apply the following 3D transformations on objects and to render the final output along with the original object.

- Translation
- Rotation
- Scaling
- Reflection
- Shearing

Display the original and the transformed object.

Code: 3D Transformations:

```
/*
3D Transformations - Translation, Rotation, Scaling, Reflection and
    Shearing
*/

#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
#include <iostream>     //for cin, cout
#include <cstring>      //for memcpy

using namespace std;

enum Axes {xAxis = 0, yAxis = 1, zAxis = 2};
enum Planes {xyPlane = 0, yzPlane = 1, zxPlane = 2};

const int WINDOW_HEIGHT = 800;
const int WINDOW_WIDTH = 800;
const int FPS = 60;

class Face{     //wrapper class for a face of a 3D object
    private:
        GLfloat r = 1, g = 1, b = 1, a = 1; //black
        GLfloat vertices[4][4];     // V * (x, y, z, h)
        int vertexCount = 4;        // V

    public:
        Face(){
            r = g = b = a = 1;  //black
            vertexCount = 4;
        }

        Face(GLfloat R, GLfloat G, GLfloat B, GLfloat A){
            //Set colors
            Face();
            r = R;
            g = G;
            b = B;
            a = A;
        }

        void setColor(GLfloat R, GLfloat G, GLfloat B, GLfloat A){
            //Set colors
            r = R;
            g = G;
            b = B;
            a = A;
```

```
47          }
48
49          void setIthVertex(int i, GLfloat x, GLfloat y, GLfloat z, GLfloat
    h = 1){
50              //Set the ith vertex coordinates
51              vertices[i][0] = x;
52              vertices[i][1] = y;
53              vertices[i][2] = z;
54              vertices[i][3] = h;
55          }
56
57          void drawFace(){
58              glColor4f(r, g, b, a);
59
60              glBegin(GL_POLYGON);
61
62              for(int i = 0; i < vertexCount; i++){
63                  glVertex3f(vertices[i][0], vertices[i][1], vertices[i][2])
    ;
64              }
65
66              glEnd();
67          }
68
69          Face transform(double transformationMatrix[4][4]){
70              Face fDash(r, g, b, a - 0.5);
71              double values[4];
72
73              // [V x 4] x [4 x 1] = [V x 1] matrix
74              for(int i = 0; i < vertexCount; i++){
75                  for(int j = 0; j < 4; j++){
76                      values[j] = transformationMatrix[j][0] * vertices[i
    ][0] +
77                                  transformationMatrix[j][1] * vertices[i
    ][1] +
78                                  transformationMatrix[j][2] * vertices[i
    ][2] +
79                                  transformationMatrix[j][3] * vertices[i
    ][3];
80                  }
81
82                  fDash.setIthVertex(i, values[0], values[1], values[2],
    values[3]);
83                  //cout << "\nVertex" << values[0] << " " << values[1] << "
     " << values[2] << " " << values[3];
84
85              }
86
87              return fDash;
88          }
89  };
```

```
90
91  class Object3D{     //wrapper class for a 3D object with multiple faces
92      private:
93          Face *faces;
94          int numFaces;
95
96      public:
97          Object3D(){
98              numFaces = 0;
99          }
100
101         Object3D(int noFaces){
102             numFaces = noFaces;
103             faces = new Face[numFaces];
104         }
105
106         void setIthFace(int i, Face face){
107             faces[i] = face;
108         }
109
110         void drawObject3D(){
111             for(int i = 0; i < numFaces; i++){
112                 faces[i].drawFace();
113             }
114         }
115
116         Object3D translateObject3D(GLfloat tx, GLfloat ty, GLfloat tz){
117             //To translate the 3D object wrt. a translation vector
118
119             double translationMatrix[4][4] = {  {1, 0, 0, tx},
120                                                 {0, 1, 0, ty},
121                                                 {0, 0, 1, tz},
122                                                 {0, 0, 0, 1}};
123
124             Object3D transformedObject(numFaces);
125
126             for(int i = 0; i < numFaces; i++){
127                 Face fDash = faces[i].transform(translationMatrix);
128                 transformedObject.setIthFace(i, fDash);
129             }
130
131             return transformedObject;
132         }
133
134         Object3D scaleObject3D(GLfloat sx, GLfloat sy, GLfloat sz){
135             //To scale the 3D object wrt. the given scaling factors
136
137             double scalingMatrix[4][4] = {  {sx, 0, 0, 0},
138                                             {0, sy, 0, 0},
139                                             {0, 0, sz, 0},
140                                             {0, 0, 0, 1}};
```

```
141
142             Object3D transformedObject(numFaces);
143
144             for(int i = 0; i < numFaces; i++){
145                 Face fDash = faces[i].transform(scalingMatrix);
146                 transformedObject.setIthFace(i, fDash);
147             }
148
149             return transformedObject;
150         }
151
152         Object3D rotateObject3D(int axis, double rotationAngle){
153             //To rotate the 3D object about an axis and an angle
154
155             double rotationAngleInRadians = rotationAngle * 3.14159 / 180;
156             double cosAngle = cos(rotationAngleInRadians);
157             double sinAngle = sin(rotationAngleInRadians);
158             double rotationMatrix[4][4];
159
160             switch(axis){
161                 case xAxis:{
162                     double temp[4][4] = {   {1, 0, 0, 0},
163                                             {0, cosAngle, -sinAngle, 0},
164                                             {0, sinAngle, cosAngle, 0},
165                                             {0, 0, 0, 1}};
166
167                     memcpy(rotationMatrix, temp, sizeof(temp));
168                     break;
169                 }
170
171                 case yAxis:{
172                     double temp[4][4] = {   {cosAngle, 0, sinAngle, 0},
173                                             {0, 1, 0, 0},
174                                             {-sinAngle, 0, cosAngle, 0},
175                                             {0, 0, 0, 1}};
176
177                     memcpy(rotationMatrix, temp, sizeof(temp));
178                     break;
179                 }
180
181                 case zAxis:{
182                     double temp[4][4] = {   {cosAngle, -sinAngle, 0, 0},
183                                             {sinAngle, cosAngle, 0, 0},
184                                             {0, 0, 1, 0},
185                                             {0, 0, 0, 1}};
186
187                     memcpy(rotationMatrix, temp, sizeof(temp));
188                     break;
189                 }
190             }
191
```

```
192             Object3D transformedObject(numFaces);
193 
194             for(int i = 0; i < numFaces; i++){
195                 Face fDash = faces[i].transform(rotationMatrix);
196                 transformedObject.setIthFace(i, fDash);
197             }
198 
199             return transformedObject;
200         }
201 
202         Object3D shearObject3D(int axis, double shx = 0, double shy = 0,
    double shz = 0){
203             //To shear the 3D object wrt. an axis and the given shear
    parameters
204 
205             double shearingMatrix[4][4];
206 
207             switch(axis){
208                 case xAxis:{
209                     double temp[4][4] = {   {1, 0, 0, 0},
210                                             {shy, 1, 0, 0},
211                                             {shz, 0, 1, 0},
212                                             {0, 0, 0, 1}};
213 
214                     memcpy(shearingMatrix, temp, sizeof(temp));
215                     break;
216                 }
217 
218                 case yAxis:{
219                     double temp[4][4] = {   {1, shx, 0, 0},
220                                             {0, 1, 0, 0},
221                                             {0, shz, 1, 0},
222                                             {0, 0, 0, 1}};
223 
224                     memcpy(shearingMatrix, temp, sizeof(temp));
225                     break;
226                 }
227 
228                 case zAxis:{
229                     double temp[4][4] = {   {1, 0, shx, 0},
230                                             {0, 1, shy, 0},
231                                             {0, 0, 1, 0},
232                                             {0, 0, 0, 1}};
233 
234                     memcpy(shearingMatrix, temp, sizeof(temp));
235                     break;
236                 }
237             }
238 
239             Object3D transformedObject(numFaces);
240 
```

```
241             for(int i = 0; i < numFaces; i++){
242                 Face fDash = faces[i].transform(shearingMatrix);
243                 transformedObject.setIthFace(i, fDash);
244             }
245 
246             return transformedObject;
247         }
248 
249         Object3D reflectObject3D(int plane){
250             //To reflect the 3D Object about a given plane
251 
252             double reflectionMatrix[4][4];
253 
254             switch(plane){
255                 case xyPlane:{
256                     double temp[4][4] = {   {1, 0, 0, 0},
257                                             {0, 1, 0, 0},
258                                             {0, 0, -1, 0},
259                                             {0, 0, 0, 1}};
260 
261                     memcpy(reflectionMatrix, temp, sizeof(temp));
262                     break;
263                 }
264 
265                 case yzPlane:{
266                     double temp[4][4] = {   {-1, 0, 0, 0},
267                                             {0, 1, 0, 0},
268                                             {0, 0, 1, 0},
269                                             {0, 0, 0, 1}};
270 
271                     memcpy(reflectionMatrix, temp, sizeof(temp));
272                     break;
273                 }
274 
275                 case zxPlane:{
276                     double temp[4][4] = {   {1, 0, 0, 0},
277                                             {0, -1, 0, 0},
278                                             {0, 0, 1, 0},
279                                             {0, 0, 0, 1}};
280 
281                     memcpy(reflectionMatrix, temp, sizeof(temp));
282                     break;
283                 }
284             }
285 
286             Object3D transformedObject(numFaces);
287 
288             for(int i = 0; i < numFaces; i++){
289                 Face fDash = faces[i].transform(reflectionMatrix);
290                 transformedObject.setIthFace(i, fDash);
291             }
```

```
292
293             return transformedObject;
294         }
295 };
296
297 void dummyFunction();
298 void mainLoop(int val);
299 void initializeDisplay();
300 void initializeBaseCube();
301 void plotBase3DObject();
302 void plotTransformation();
303
304 Object3D cube;
305
306 int main(int argc, char **argv){
307     glutInit(&argc, argv);
308     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
309     glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
310     glutCreateWindow("3D Transformations - Examples");
311
312     initializeBaseCube();
313     initializeDisplay();
314
315     glutDisplayFunc(dummyFunction);
316     glutTimerFunc(1000/FPS, mainLoop, 0);
317     glutMainLoop();
318
319     return 1;
320 }
321
322 void mainLoop(int val){
323     //Render the display using the timer function
324     plotTransformation();
325 }
326
327 void dummyFunction(){
328     //Placeholder function
329 }
330
331 void initializeDisplay(){
332     glClearColor(1, 1, 1, 1);
333     glOrtho(-800, 800, -800, 800, -800, 800);   //Orthographic projection
334     glEnable(GL_DEPTH_TEST);    //Enable depth
335
336     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
337
338     //Rotate the entire display so that different sides of the 3D object
    can be seen
339     glRotatef(50, 1, 0, 0);
340     glRotatef(50, 0, 1, 0);
341     glRotatef(50, 0, 0, 1);
```

```
342
343     glEnable(GL_BLEND);     //enable blending (translucent colors)
344     glDepthMask(GL_FALSE);
345     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);  //set the blend
        function for translucency
346 }
347
348 void initializeBaseCube(){
349     //Set the coordinates for the base cube
350
351     cube = Object3D(6);
352
353     Face front, back, left, right, bottom, top;
354
355     front = Face(1, 0, 0, 0.75);   //Red
356     front.setIthVertex(0, -100, 100, 100);
357     front.setIthVertex(1, 100, 100, 100);
358     front.setIthVertex(2, 100, -100, 100);
359     front.setIthVertex(3, -100, -100, 100);
360
361     back = Face(0, 1, 0, 0.75);    //Green
362     back.setIthVertex(0, -100, 100, -100);
363     back.setIthVertex(1, 100, 100, -100);
364     back.setIthVertex(2, 100, -100, -100);
365     back.setIthVertex(3, -100, -100, -100);
366
367     left = Face(0, 0, 1, 0.75);    //Blue
368     left.setIthVertex(0, -100, 100, -100);
369     left.setIthVertex(1, -100, 100, 100);
370     left.setIthVertex(2, -100, -100, 100);
371     left.setIthVertex(3, -100, -100, -100);
372
373     right = Face(1, 1, 0, 0.75);   //Yellow
374     right.setIthVertex(0, 100, 100, -100);
375     right.setIthVertex(1, 100, 100, 100);
376     right.setIthVertex(2, 100, -100, 100);
377     right.setIthVertex(3, 100, -100, -100);
378
379     bottom = Face(0, 1, 1, 0.75);  //Cyan
380     bottom.setIthVertex(0, -100, -100, -100);
381     bottom.setIthVertex(1, 100, -100, -100);
382     bottom.setIthVertex(2, 100, -100, 100);
383     bottom.setIthVertex(3, -100, -100, 100);
384
385     top = Face(1, 0, 1, 0.75);     //Magenta
386     top.setIthVertex(0, -100, 100, -100);
387     top.setIthVertex(1, 100, 100, -100);
388     top.setIthVertex(2, 100, 100, 100);
389     top.setIthVertex(3, -100, 100, 100);
390
391     cube.setIthFace(0, front);
```

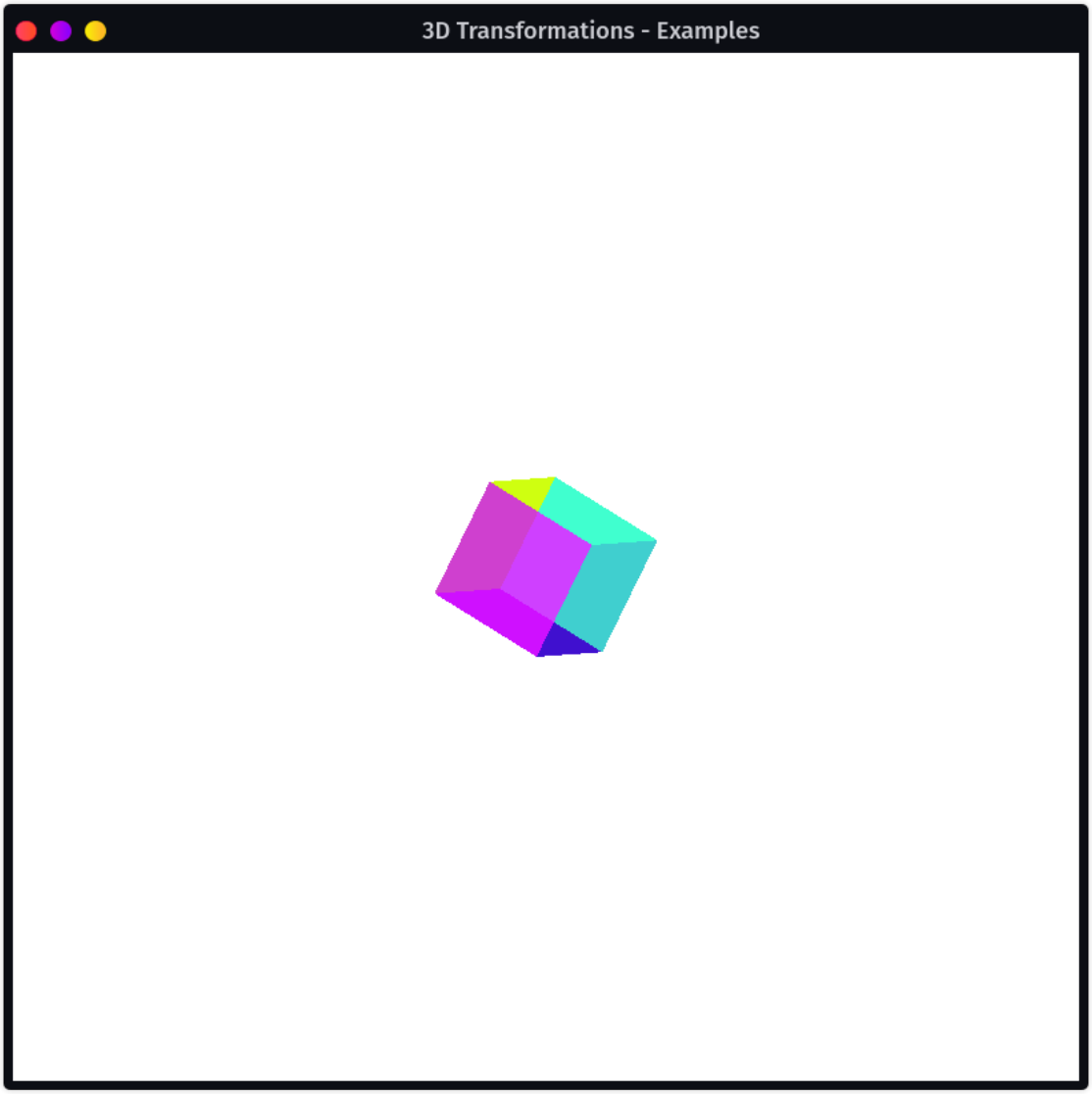
```
392     cube.setIthFace(1, back);
393     cube.setIthFace(2, left);
394     cube.setIthFace(3, right);
395     cube.setIthFace(4, bottom);
396     cube.setIthFace(5, top);
397 }
398 
399 void plotBase3DObject(){
400     //Plot the base 3D object without any transformations
401 
402     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);   //Clear window
403     cube.drawObject3D();
404     glFlush();
405 }
406 
407 void plotTransformation(){
408     //Plot the transformation
409 
410     plotBase3DObject();
411     glFlush();
412 
413     int transform = 0;
414     Object3D cubeDash;
415 
416     while(true){
417         //Await user input
418 
419         cout << "\nChoose Transformation: " << endl;
420         cout << "\t1 for Translation" << endl;
421         cout << "\t2 for Rotation" << endl;
422         cout << "\t3 for Scaling" << endl;
423         cout << "\t4 for Reflection" << endl;
424         cout << "\t5 for Shearing" << endl;
425         cout << "\t0 to Exit" << endl;
426         cout << "\tYour Option -> ";
427         cin >> transform;
428 
429         switch(transform){
430             case 0:{
431                 exit(0);
432             }
433 
434             case 1:{
435                 float tx, ty, tz;
436                 cout << endl << "-----TRANSLATION-----" << endl;
437                 cout << "\nEnter Translation Vector Coordinates: " << endl
    ;
438                 cout << "\nEnter X: "; cin >> tx;
439                 cout << "\nEnter Y: "; cin >> ty;
440                 cout << "\nEnter Z: "; cin >> tz;
441 
```

```
442                 cubeDash = cube.translateObject3D(tx, ty, tz);
443                 cout << endl << "------TRANSLATION DONE-----" << endl;
444 
445                 break;
446             }
447 
448             case 2:{
449                 double angle; int axis;
450                 cout << endl << "-----ROTATION-----" << endl;
451                 cout << "\nEnter Rotation Axis: " << endl;
452                 cout << "\t0 for X-Axis" << endl;
453                 cout << "\t1 for Y-Axis" << endl;
454                 cout << "\t2 for Z-Axis" << endl;
455                 cout << "\tYour Option -> ";
456                 cin >> axis;
457 
458                 cout << "\nEnter Rotation Angle: "; cin >> angle;
459 
460                 cubeDash = cube.rotateObject3D(axis, angle);
461                 cout << endl << "-----ROTATION DONE-----" << endl;
462                 break;
463             }
464 
465             case 3:{
466                 float sx, sy, sz;
467 
468                 cout << endl << "-----SCALING-----" << endl;
469                 cout << "\nEnter Scale Factors: " << endl;
470                 cout << "\nEnter X Factor: "; cin >> sx;
471                 cout << "\nEnter Y Factor: "; cin >> sy;
472                 cout << "\nEnter Z Factor: "; cin >> sz;
473 
474                 cubeDash = cube.scaleObject3D(sx, sy, sz);
475                 cout << endl << "------SCALING DONE-----" << endl;
476 
477                 break;
478             }
479 
480             case 4:{
481                 int plane;
482                 cout << endl << "-----REFLECTION-----" << endl;
483                 cout << "\nEnter Reflection Plane: " << endl;
484                 cout << "\t0 for XY-Plane" << endl;
485                 cout << "\t1 for YZ-Plane" << endl;
486                 cout << "\t2 for ZX-Plane" << endl;
487                 cout << "\tYour Option -> ";
488                 cin >> plane;
489 
490                 cubeDash = cube.reflectObject3D(plane);
491                 cout << endl << "-----REFLECTION DONE-----" << endl;
492                 break;
```

```
493             }
494
495             case 5:{
496                 double shx = 0, shy = 0, shz = 0; int axis;
497
498                 cout << endl << "-----SHEARING-----" << endl;
499                 cout << "\nEnter Shear Axis: " << endl;
500                 cout << "\t0 for X-Axis" << endl;
501                 cout << "\t1 for Y-Axis" << endl;
502                 cout << "\t2 for Z-Axis" << endl;
503                 cout << "\tYour Option -> ";
504                 cin >> axis;
505
506                 cout << "\nEnter Shear Factors: " << endl;
507
508                 switch(axis){
509                     case xAxis:{
510                         cout << "\n Enter Y Factor: "; cin >> shy;
511                         cout << "\n Enter Z Factor: "; cin >> shz;
512                         break;
513                     }
514
515                     case yAxis:{
516                         cout << "\n Enter X Factor: "; cin >> shx;
517                         cout << "\n Enter Z Factor: "; cin >> shz;
518                         break;
519                     }
520
521                     case zAxis:{
522                         cout << "\n Enter X Factor: "; cin >> shx;
523                         cout << "\n Enter Y Factor: "; cin >> shy;
524                         break;
525                     }
526                 }
527
528                 cubeDash = cube.shearObject3D(axis, shx, shy, shz);
529                 cout << endl << "-----SHEARING DONE-----" << endl;
530                 break;
531             }
532         }
533
534         plotBase3DObject();
535         cubeDash.drawObject3D();
536         glFlush();
537     }
538 }
```

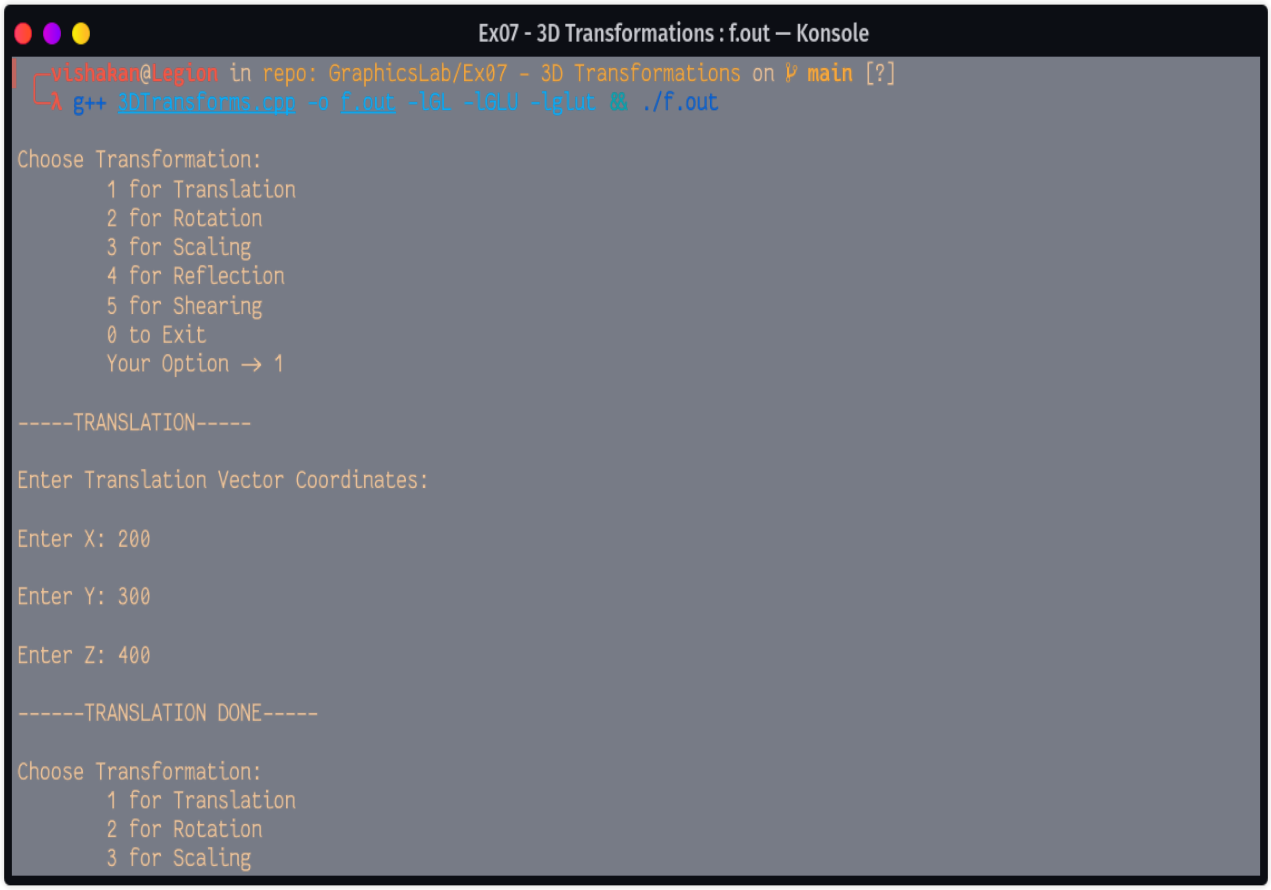
Output: Base 3D Object

Figure 1: Base 3D Object.



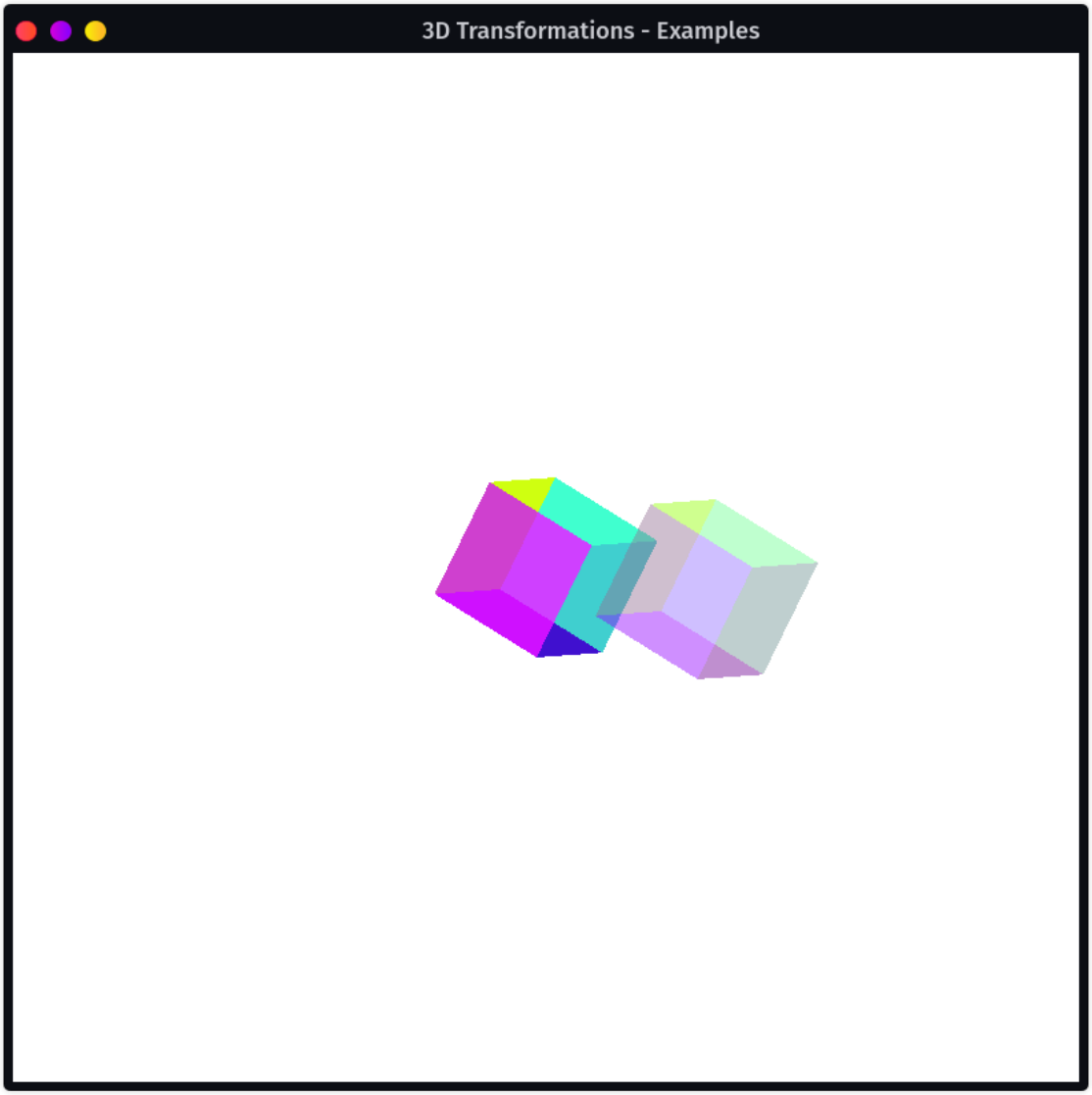
Output: Console - Translation, (200, 300, 400)

Figure 2: Output: Console - Translation, (200, 300, 400).



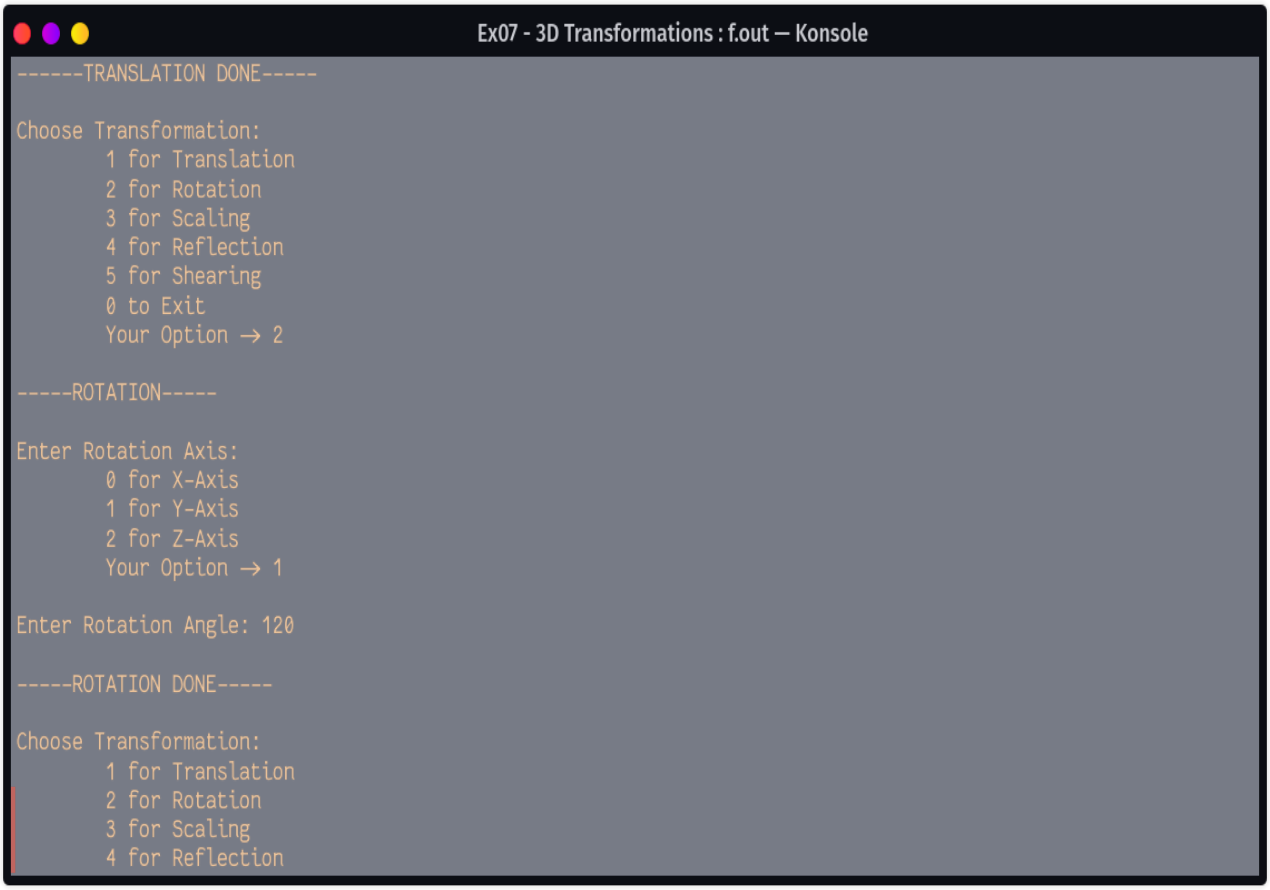
Output: Translated 3D Object, (200, 300, 400)

Figure 3: Translated 3D Object, (200, 300, 400).



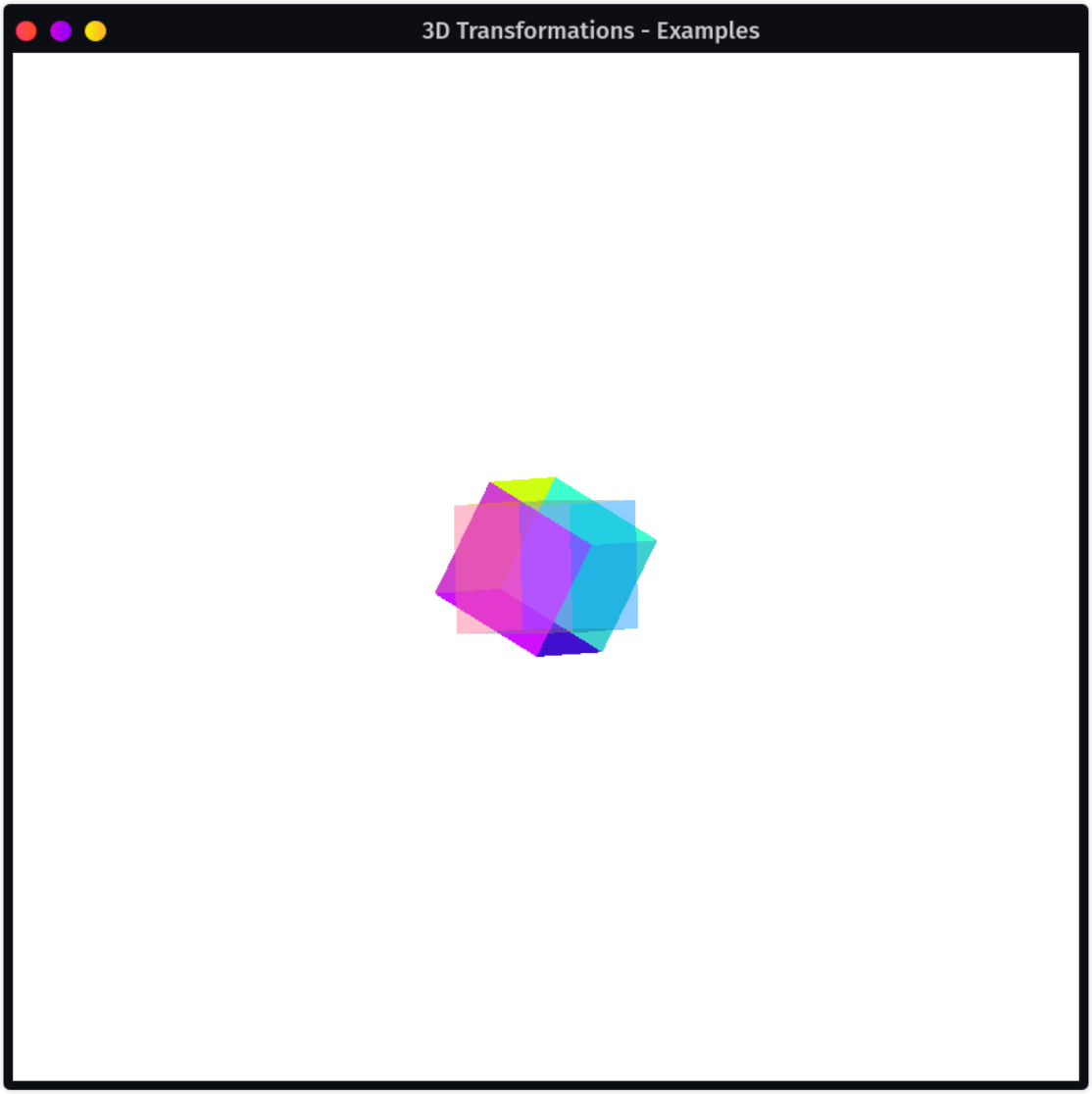
Output: Console - Rotation, Y-Axis, 120°

Figure 4: Output: Console - Rotation, Y-Axis, 120°.



Output: Rotated 3D Object, Y-Axis, 120°

Figure 5: Rotated 3D Object, Y-Axis, 120°.

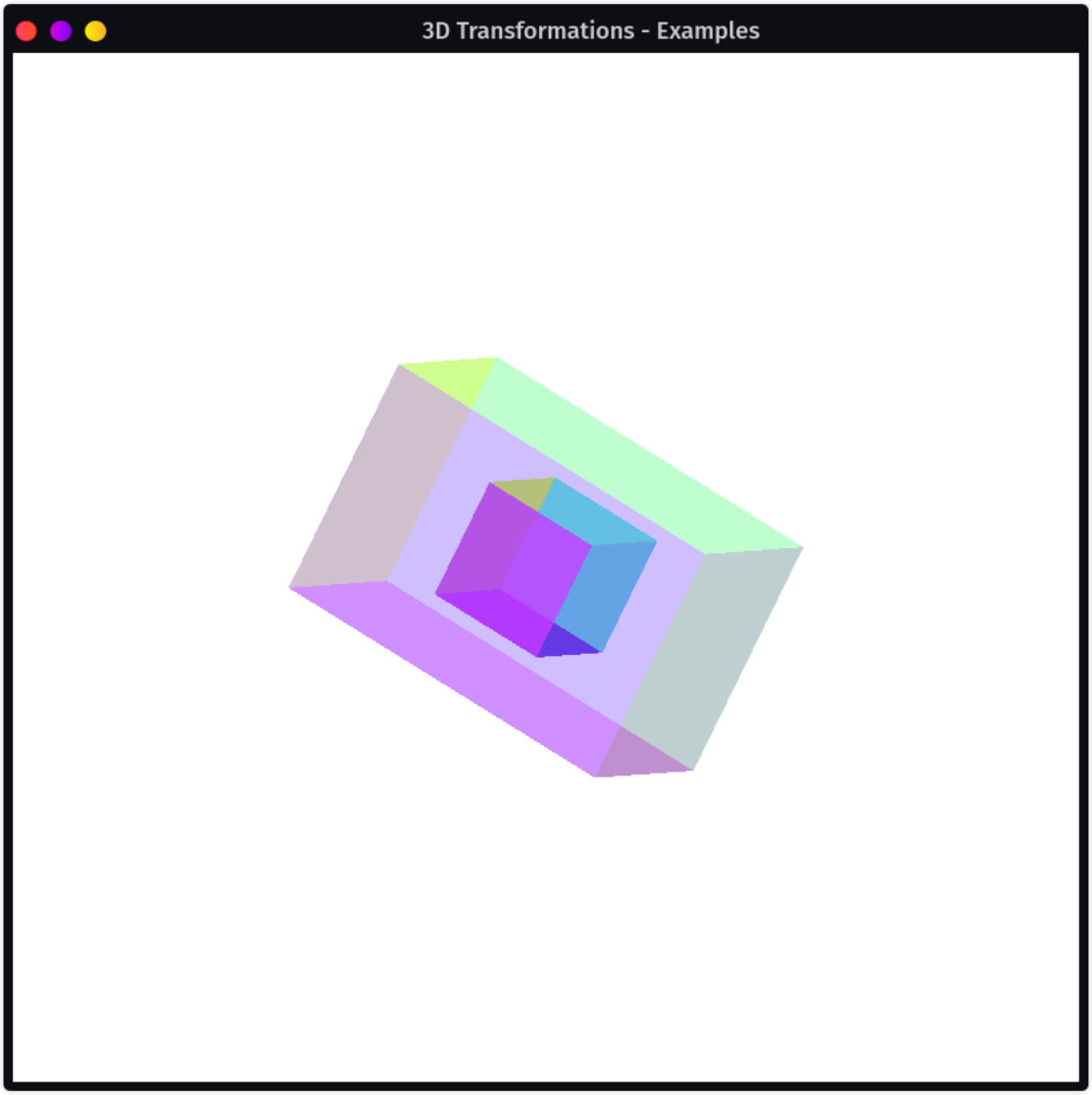


Output: Console - Scaling, (2, 1.5, 3)

Figure 6: Output: Console - Scaling, (2, 1.5, 3).

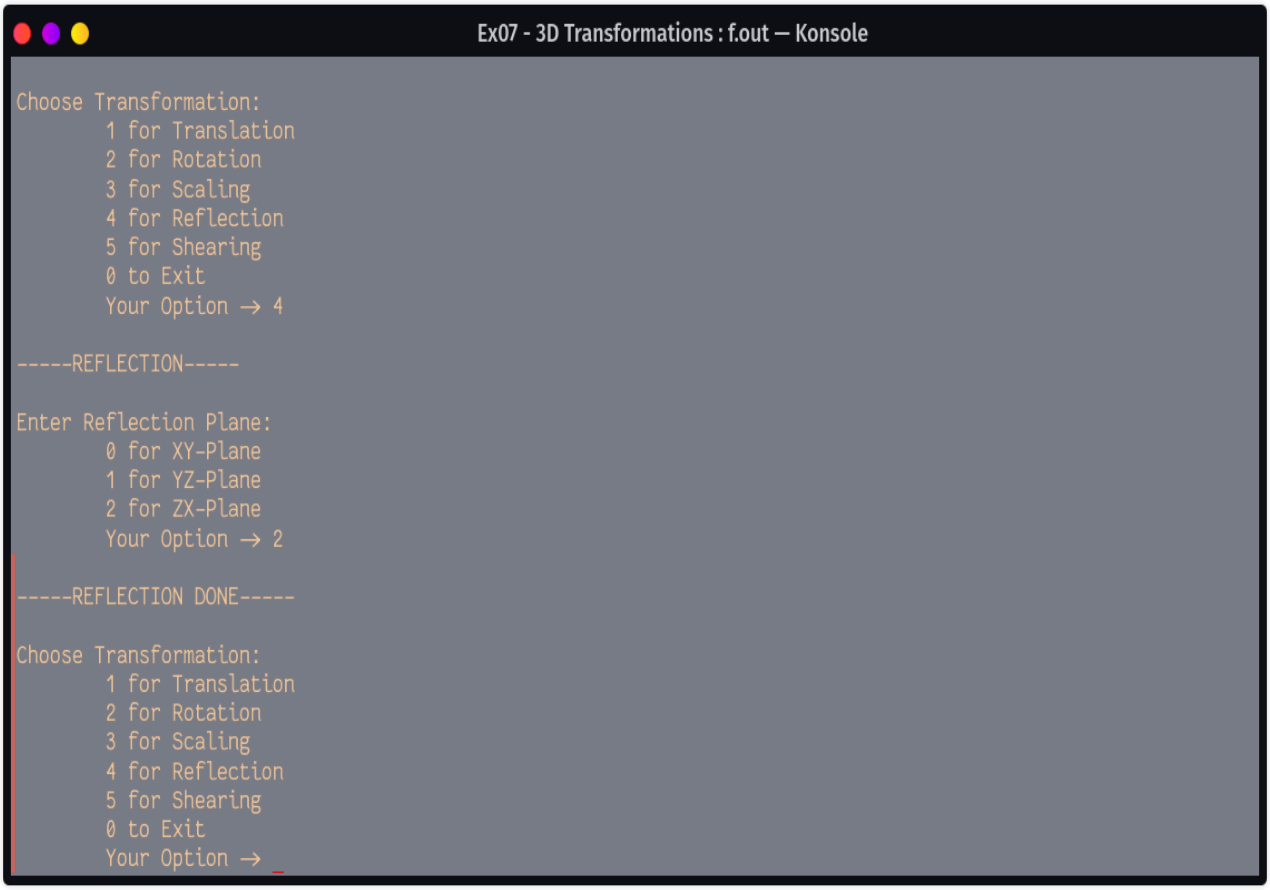
Output: Scaled 3D Object, (2, 1.5, 3)

Figure 7: Scaled 3D Object, (2, 1.5, 3).



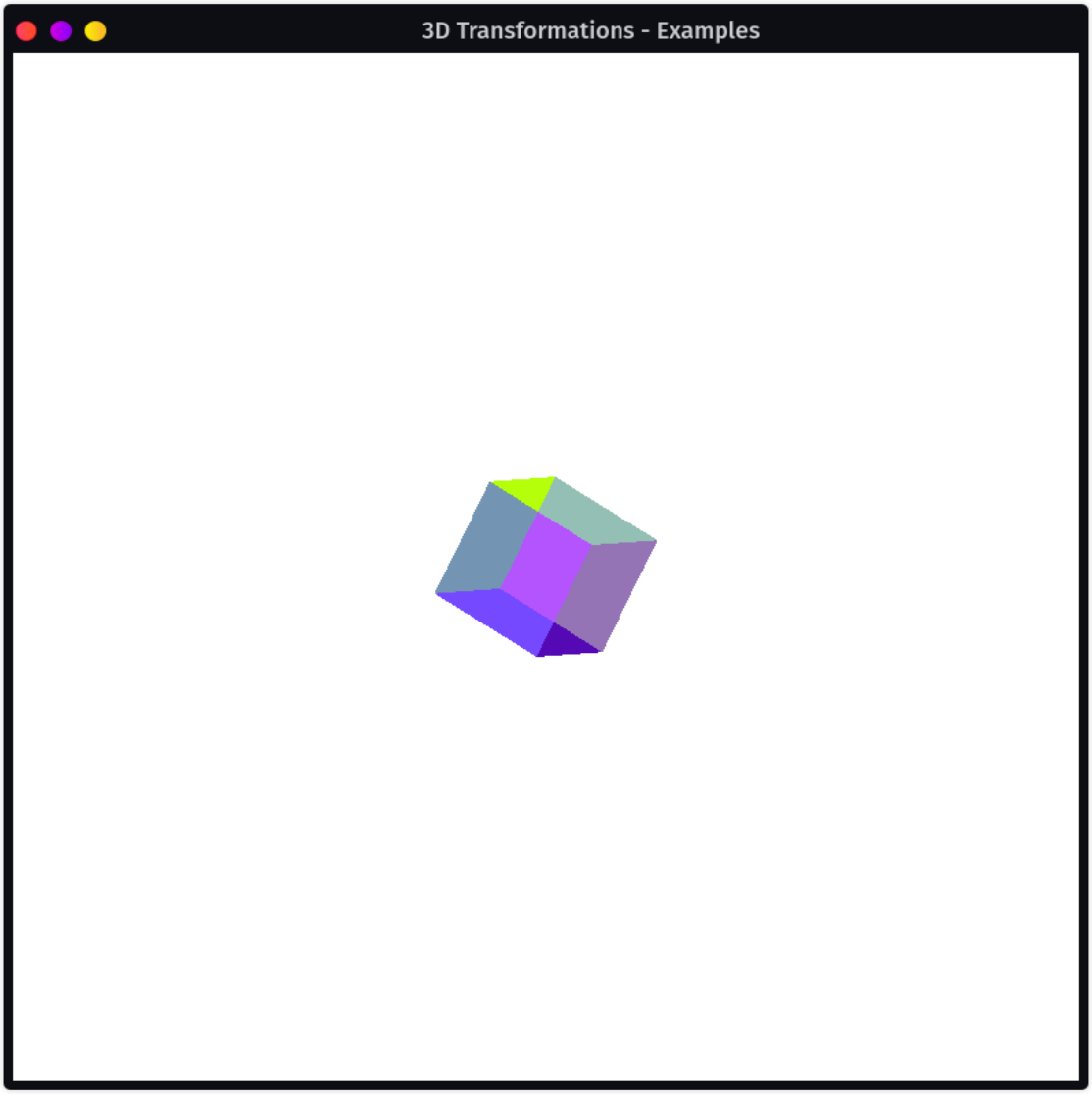
Output: Console - Reflection, ZX Plane

Figure 8: Output: Console - Reflection, ZX Plane.



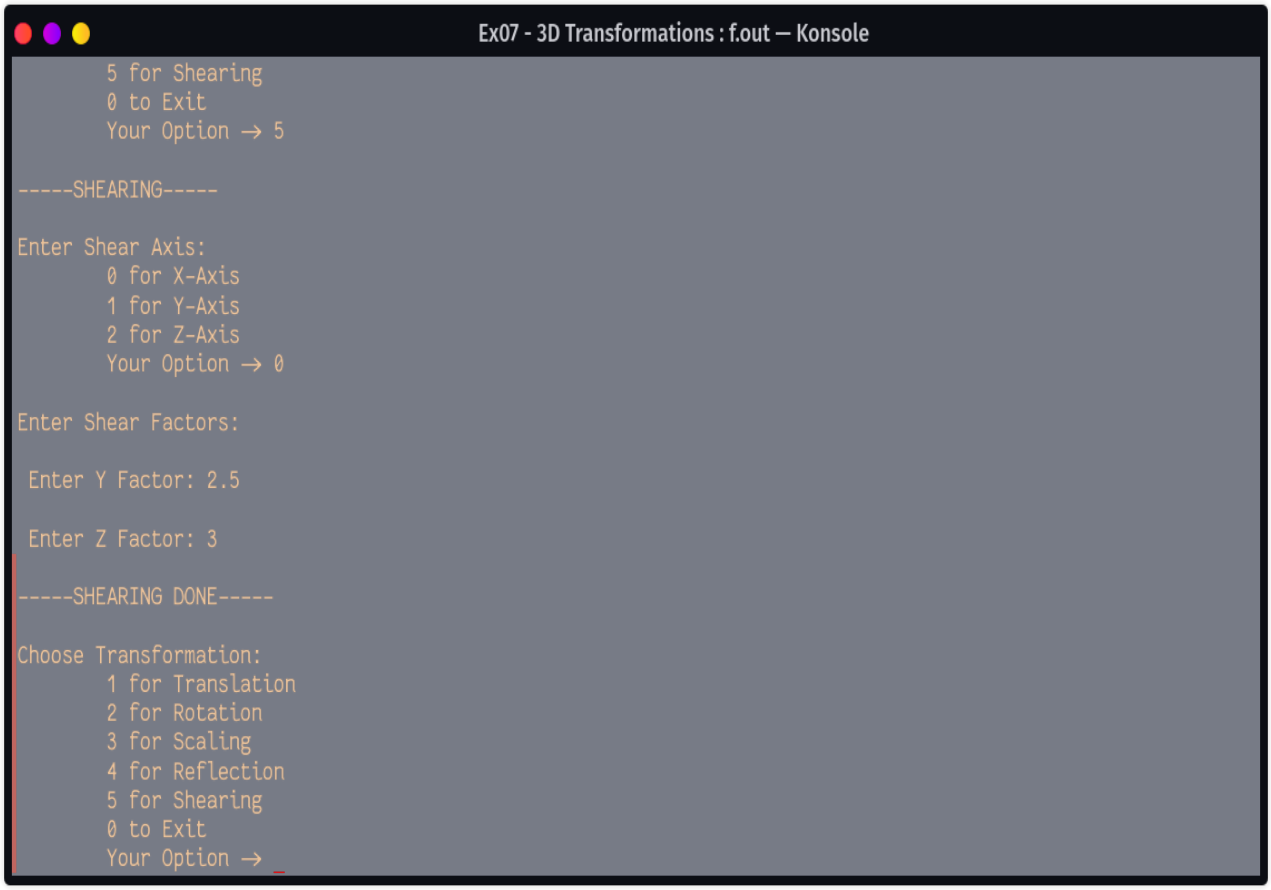
Output: Reflected 3D Object, ZX Plane

Figure 9: Reflected 3D Object, ZX Plane.



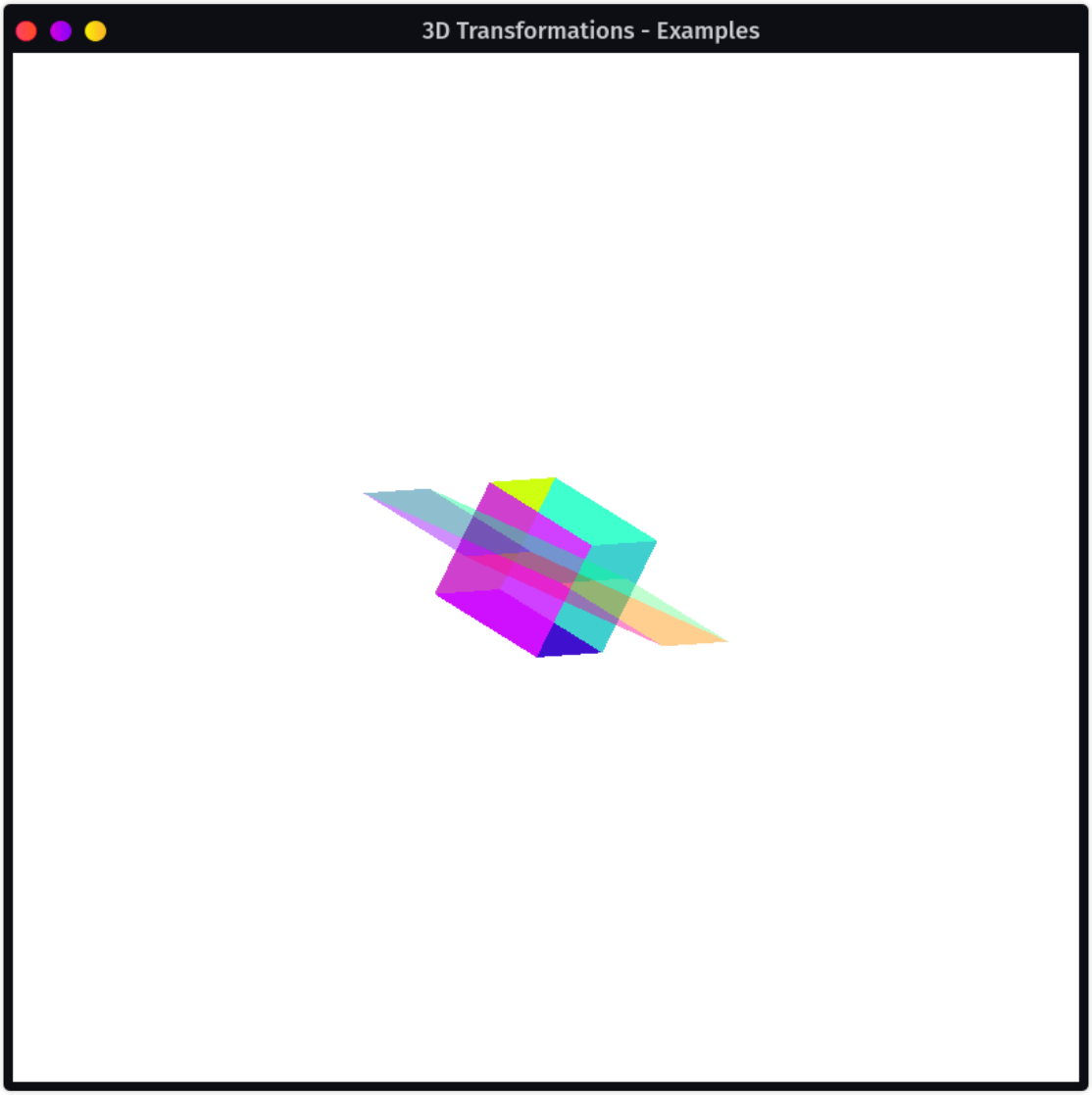
Output: Console - Shearing, X Axis, (2.5, 3)

Figure 10: Output: Console - Shearing, X Axis, (2.5, 3).



Output: Sheared 3D Object, X Axis, (2.5, 3)

Figure 11: Sheared 3D Object, X Axis, (2.5, 3).



Learning Outcome:

- I learnt how to represent a **3D Object** in terms of its **2D Faces** as planar polygons.
- I learnt how to perform **Translation, Rotation, Scaling, Reflection and Shearing** upon a given 3D Object with the use of 3D transformation matrices.
- I learnt about the different transformation matrices and how to calculate them.
- I learnt how to set **depth flags** to enable the **Z-Axis** projections.
- I came to know about inbuilt functions that perform the same transformations like **glRotate3f()** in OpenGL.
- I learnt how to do **orthographic projection** in 3D with OpenGL, using **glOrtho()**.
- I learnt to use parametrized and default constructors in C++ and to call them from other constructors.

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

17 October 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 9: 3-Dimensional Projections in C++ using OpenGL

Aim:

Write a menu driven program to perform Orthographic parallel projection and Perspective projection on any 3D object.

Set the camera to any position on the 3D space. Have $(0, 0, 0)$ at the center of the screen. Draw X, Y and Z axis. You can use gluPerspective() to perform perspective projection.

Use keyboard functions to rotate and show different views of the object.

Note: Can use built-in functions for 3D transformations.

Code: 3D Projections:

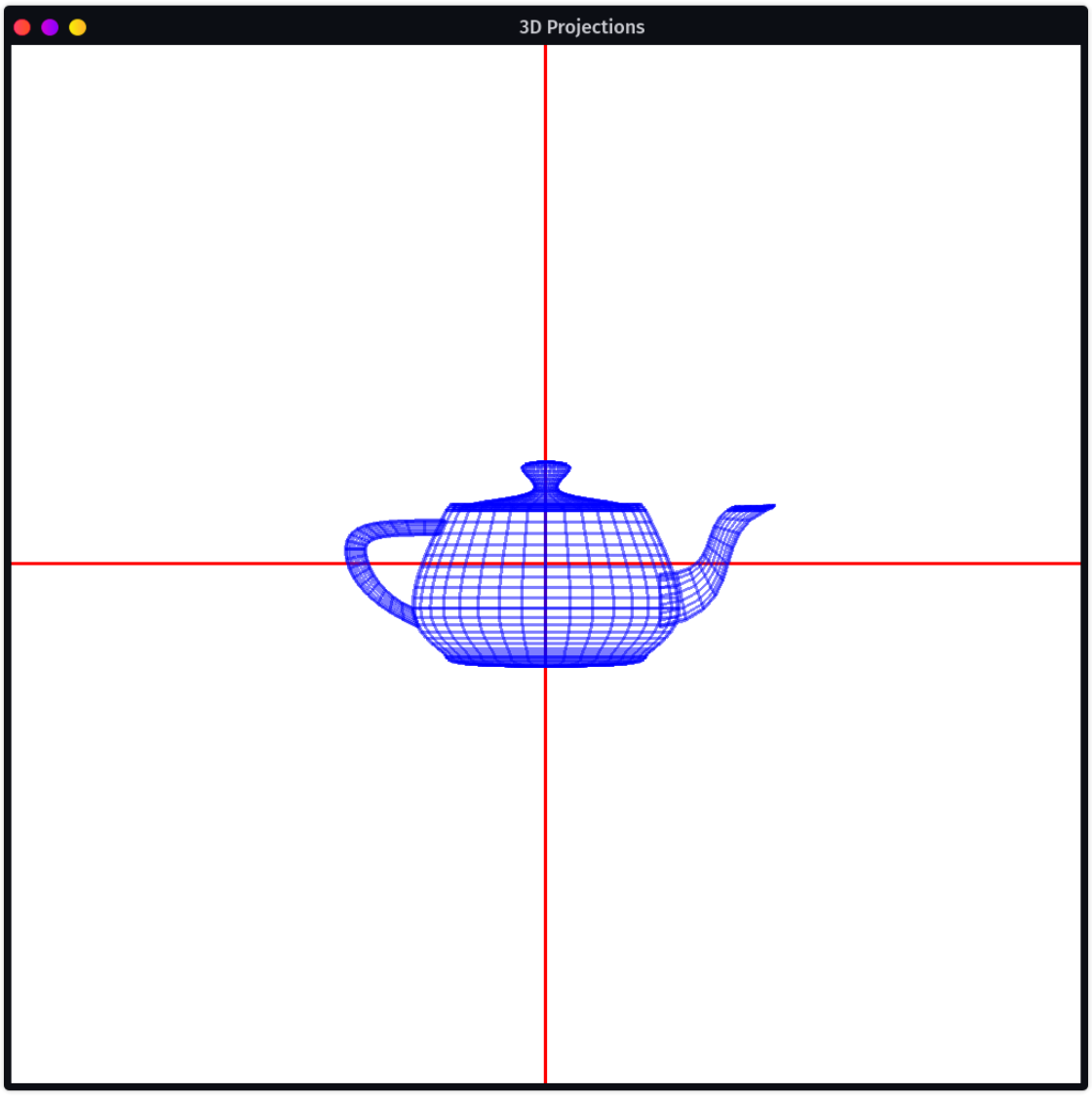
```
/*
 To demonstrate Orthographic Parallel and Perspective Projection using
    OpenGL
 and to also use keyboard functions and show different object views, along
     with
 setting the camera position.
*/

#include<iostream>
#include<cstring>
#include<GL/glut.h>
#include<math.h>

using namespace std;

//Global constants
const float WINDOW_WIDTH = 1000;
const float WINDOW_HEIGHT = 1000;
const float X_MIN = -500;
const float X_MAX = 500;
const float Y_MIN = -500;
const float Y_MAX = 500;
const int FPS = 60;

//Global variables to handle rotation
double x_rotate = 0;
double y_rotate = 0;

//Global variable for projection
bool isOrthoProjection = true;

void initializeDisplay();
void keyboardKeys(unsigned char key, int x, int y);
void drawAxes();

int main(int argc, char **argv){

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("3D Projections");

    //Register the callback functions
    glutDisplayFunc(initializeDisplay);
    glutKeyboardFunc(keyboardKeys);

    //Change to projection mode before applying glOrtho()/gluPerspective()
```

```
46      glMatrixMode(GL_PROJECTION);
47      glLoadIdentity();
48
49      glutMainLoop();
50
51      return 0;
52  }
53
54  void initializeDisplay(){
55      //Initialize display parameters
56
57      glClearColor(1, 1, 1, 1);
58      glClear(GL_COLOR_BUFFER_BIT);
59
60      //Translucency
61      glEnable(GL_BLEND);
62      glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
63
64      //Line width
65      glLineWidth(3);
66
67      //Apply the transformations & drawing on the model view matrix
68      glMatrixMode(GL_MODELVIEW);
69
70      //Draw the X and Y axis
71      drawAxes();
72
73      //Transform only the drawn object, so use the matrix stack accordingly
74      glPushMatrix();
75
76      if(isOrthoProjection){
77          //Parallel Projection
78          glOrtho(-2, 2, -2, 2, -2, 2);
79      } else{
80          //Perspective Projection
81          gluPerspective(120, 1, 0.1, 50); //FoVy = 120, Aspect Ratio = 1
82      }
83
84      gluLookAt(0, 0, 1, 0, 0, 0, 0, 1, 0);   //Camera, Center & Up Vector
85      glRotatef(x_rotate, 1, 0, 0);   //Keyboard based rotations
86      glRotatef(y_rotate, 0, 1, 0);
87
88      glColor4f(0, 0, 1, 0.3);    //Draw the object
89      glutWireTeapot(0.5);
90
91      glPopMatrix();  //Pop the matrix back into the model view stack
92
93      glFlush();
94  }
95
96  void drawAxes(){
```

```
97      //To draw X and Y axis
98
99      glColor3d(1, 0, 0);
100
101     glBegin(GL_LINES);
102
103     glVertex2f(-2, 0);
104     glVertex2f(2, 0);
105
106     glVertex2f(0, -2);
107     glVertex2f(0, 2);
108
109     glEnd();
110     glFlush();
111 }
112
113 void keyboardKeys(unsigned char key, int x, int y){
114     //Callback function for keyboard interactivity
115
116     key = tolower(key);
117
118     switch(key){
119         case 'w':{
120             x_rotate += 5;
121             break;
122         }
123         case 's':{
124             x_rotate -= 5;
125             break;
126         }
127         case 'd':{
128             y_rotate += 5;
129             break;
130         }
131         case 'a':{
132             y_rotate -= 5;
133             break;
134         }
135         case 32:{
136             //Spacebar for changing projections
137             isOrthoProjection = !isOrthoProjection;
138             break;
139         }
140     }
141
142     //Update the display
143     glutPostRedisplay();
144 }
```

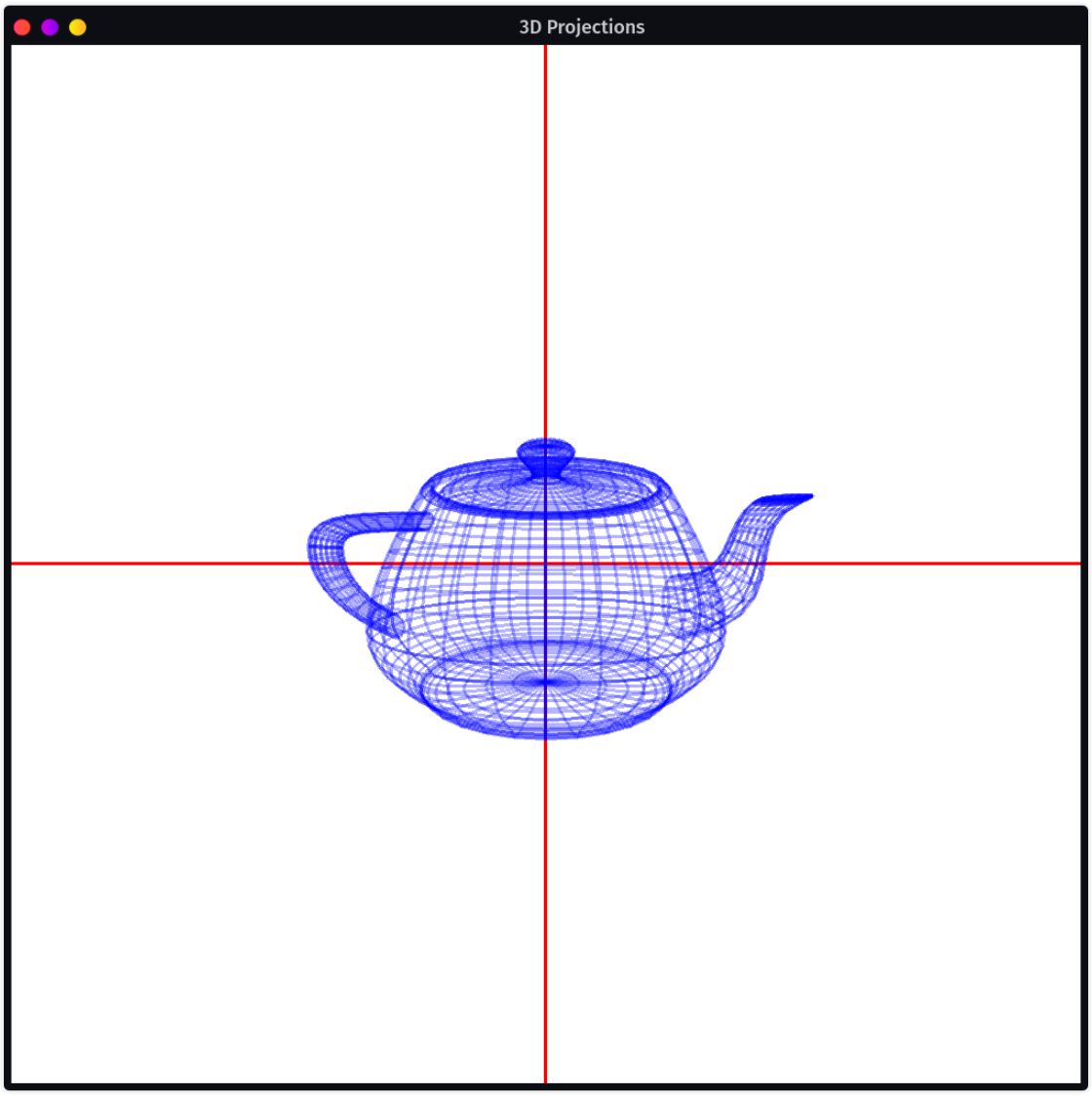
Output: 3D Object - Ortho

Figure 1: 3D Object - Ortho.



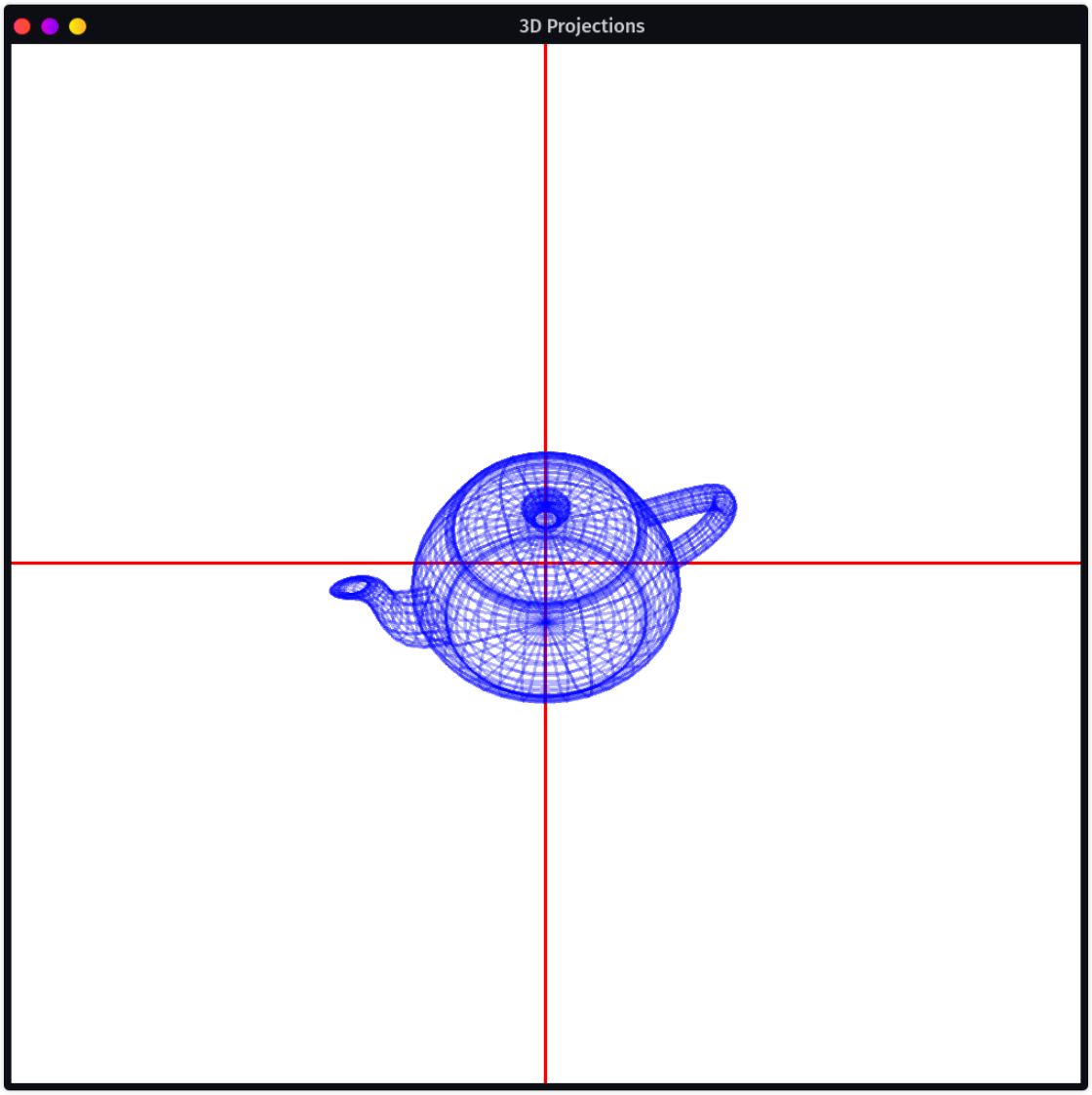
Output: 3D Object - Perspective

Figure 2: 3D Object - Perspective.



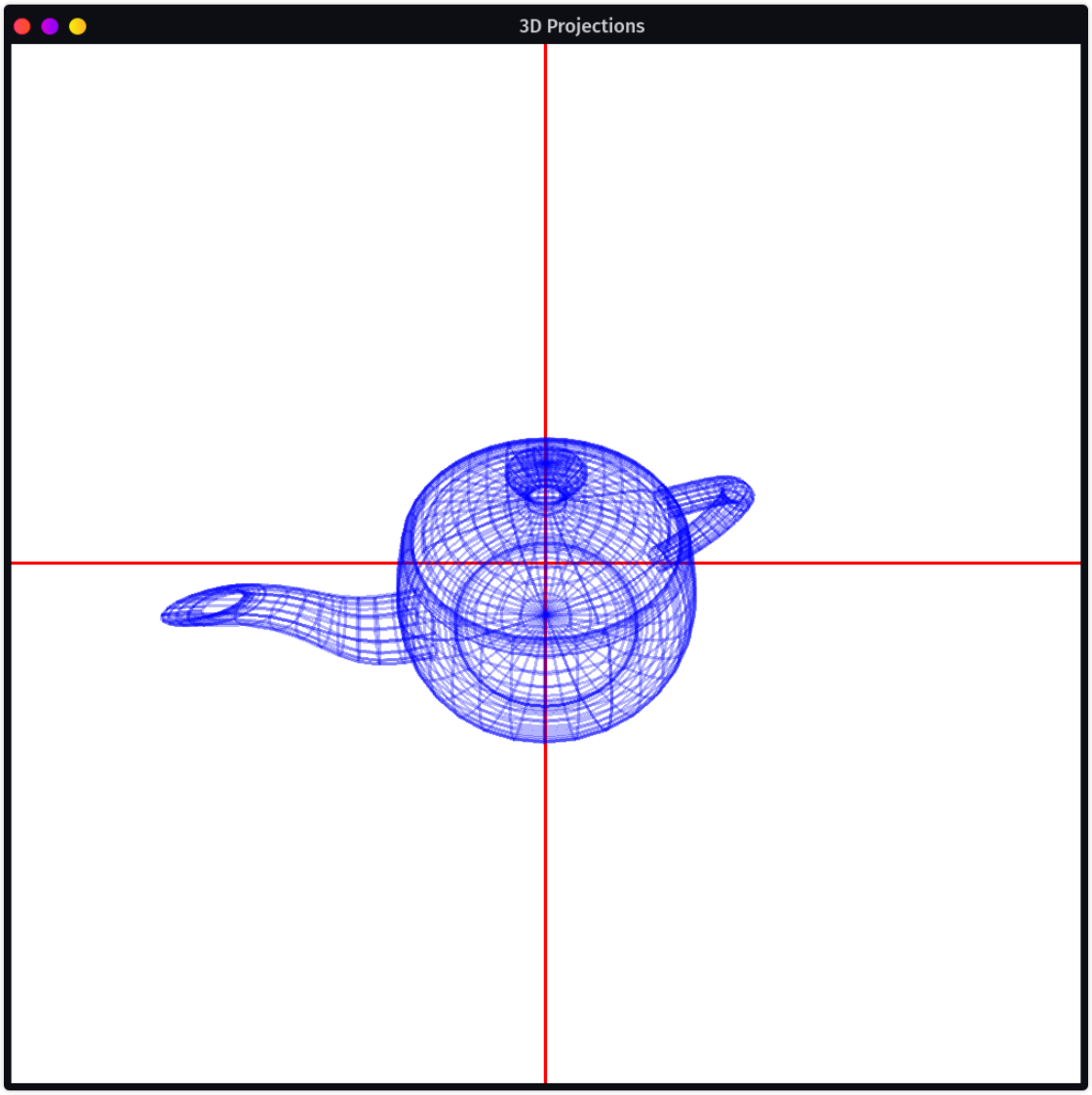
Output: 3D Object - Ortho (Rotated)

Figure 3: 3D Object - Ortho (Rotated).



Output: 3D Object - Perspective (Rotated)

Figure 4: 3D Object - Perspective (Rotated).



Learning Outcome:

- I learnt how to set-up **keyboard functions** for handling different user inputs for rotation & projections using **glutKeyboardFunc()** callback method.
- I learnt how to use inbuilt 3D transformation methods like **glRotatef() and glTranslatef()**.
- I understood the working of **glOrtho() and gluPerspective()** methods.
- I learnt how to set-up camera position using **gluLookAt()** method.
- I understood the usage of **glPushMatrix() & glPopMatrix()**.
- I understood the difference between the matrix modes of **GL_MODELVIEW & GL_PROJECTION**, and when to use which.
- I was able to use the inbuilt **glutWireTeapot()** method to display a 3-D Teapot object and performed parallel and perspective projections upon them, apart from rotating the object in the X and Y directions.
- I learnt about **field of view**, **aspect ratio** and how to properly configure **zNear & zFar** in gluPerspective().
- I learnt how orthographic and parallel projections differ from each other.
- I understood how the **camera, center & up vectors** are configured in gluLookAt().

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

27 October 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 10: Creating a 3D Scene in C++ using OpenGL

Aim:

Write a C++ program using OpenGl to draw atleast four 3D objects. Apply lighting and texture and render the scene. Apply transformations to create a simple 3D animation.

OpenGL Functions to use:

- glShadeModel()
- glMaterialfv()
- glLightfv()
- glEnable()
- glGenTextures()
- glTexEnvf()
- glBindTexture()
- glTexParameteri()
- glTexCoord2f()

Note: Use built-in transformation functions.

Code: 3D Scene:

```
 /*
Write a C++ program using Opengl to draw atleast four 3D objects. Apply lighting and texture and
render the scene. Apply transformations to create a simple 3D animation.
*/

#include <iostream>
#include <cstring>
#include <GL/glut.h>
#include <math.h>

const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 800;
const int FPS = 60;

//Global variables for handling animation
float translate_x = 0;
int frame = 0;
int direction = 1;

using namespace std;

void initializeDisplay();
void renderAnimation(int val);
void setLights();
void setMaterialParams(float aR, float aG, float aB, float dR, float dG, float dB, float sR, float sG, float sB, float shiny);

int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("3D Animation");

    glutDisplayFunc(initializeDisplay);
    glutTimerFunc(1000/FPS, renderAnimation, 0);

    glEnable(GL_DEPTH_TEST);

    setLights();

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(40, 1, 4, 20);

    glMatrixMode(GL_MODELVIEW);
```

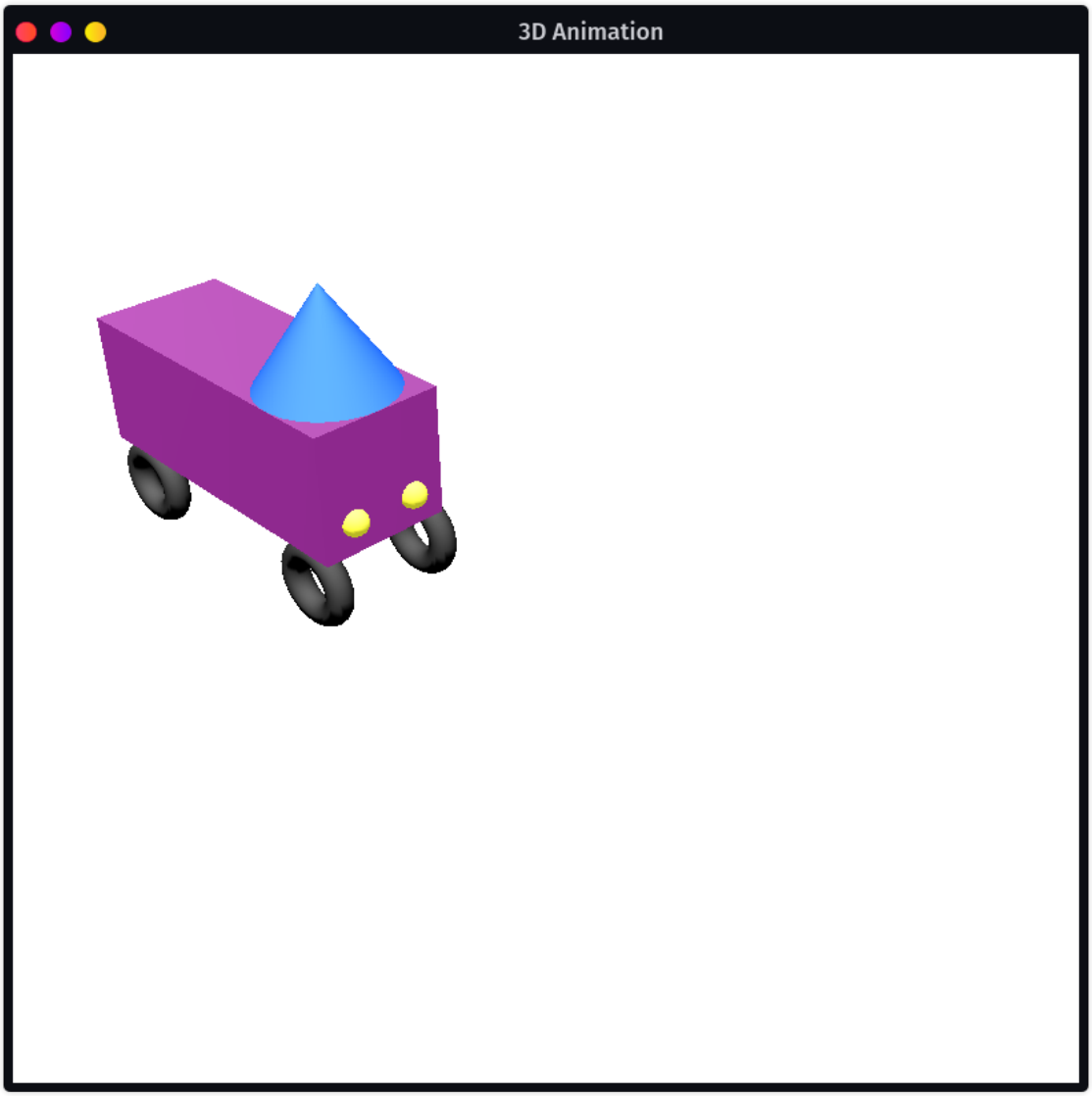
```
46      glLoadIdentity();
47      gluLookAt(5, 5, 5, 0, 0, 0, 0, 1, 0);
48
49      glutMainLoop();
50  }
51
52  void initializeDisplay(){
53      glClearColor(1, 1, 1, 1);
54      glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
55
56      glMatrixMode(GL_MODELVIEW);
57
58      glColor4f(0, 0, 0, 0.3);
59
60      //Draw a car
61
62      glPushMatrix();
63      glTranslatef(translate_x, 0, 0);
64
65      //Body
66      glPushMatrix();
67      glScalef(5, 2, 2);
68      setMaterialParams(0.75, 0.2, 0.75, 1, 1, 1, 1, 1, 1, 100);
69      glutSolidCube(0.5);
70      glPopMatrix();
71
72      //Wheels
73      setMaterialParams(0, 0, 0, 0, 0, 0, 1, 1, 1, 1);
74
75      //Back Left
76      glPushMatrix();
77      glTranslatef(-1.25, -1, 0.25);
78      glutSolidTorus(0.1, 0.25, 30, 30);
79      glPopMatrix();
80
81      //Front Left
82      glPushMatrix();
83      glTranslatef(0.75, -1, 0.25);
84      glutSolidTorus(0.1, 0.25, 30, 30);
85      glPopMatrix();
86
87      //Back Right
88      glPushMatrix();
89      glTranslatef(-1.25, -1, -0.7);
90      glutSolidTorus(0.1, 0.25, 30, 30);
91      glPopMatrix();
92
93      //Front Right
94      glPushMatrix();
95      glTranslatef(0.75, -1, -0.7);
96      glutSolidTorus(0.1, 0.25, 30, 30);
```

```
97      glPopMatrix();
98
99      //Headlights
100     setMaterialParams(1, 1, 0.2, 1, 1, 1, 1, 1, 0.2, 300);
101
102     //Left
103     glPushMatrix();
104     glTranslatef(1.25, -0.25, 0.25);
105     glutSolidSphere(0.1, 30, 30);
106     glPopMatrix();
107
108     //Right
109     glPushMatrix();
110     glTranslatef(1.25, -0.25, -0.25);
111     glutSolidSphere(0.1, 30, 30);
112     glPopMatrix();
113
114     //Roof Hat
115     glPushMatrix();
116
117     glRotatef(270, 1, 0, 0);
118     glTranslatef(0.75, 0, 0.5); //Order important. Rotate->Translate
119     //Last command acted on first in OpenGL, thus rotate about fixed point
        here
120
121     setMaterialParams(0, 0.25, 1, 0, 0.5, 1, 1, 1, 1, 1);
122     //setMaterialParams(0, 0.5, 1, 0, 0.5, 1, 1, 1, 1, 50);
123     glutSolidCone(0.5, 0.75, 30, 30);
124
125     glPopMatrix();
126
127     glPopMatrix();
128
129     glFlush();
130     glutSwapBuffers();  //Swap the offscreen buffer to screen
131 }
132
133 void renderAnimation(int val){
134     //Render an animation frame by frame
135
136     frame = (frame % FPS) + 1;
137
138     if(frame % 5 == 0){
139         translate_x += (0.04 * direction);
140         glutPostRedisplay();
141     }
142
143     if(translate_x >= 1.40 || translate_x <= -3.40){
144         direction *= -1;
145     }
146
```

```
147     //Call the timer function again to keep animating
148     glutTimerFunc(1000/FPS, renderAnimation, 0);
149 
150 }
151 
152 void setMaterialParams(float aR, float aG, float aB, float dR, float dG,
        float dB, float sR, float sG, float sB, float shiny){
153     //Set material's ambient, diffuse and specular component colors, along
        with
154     //the shininess of the material
155 
156     float ambient[3] = {aR, aG, aB};
157     float diffuse[3] = {dR, dG, dB};
158     float specular[3] = {sR, sG, sB};
159 
160     glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
161     glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuse);
162     glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);
163     glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, shiny);
164 }
165 
166 void setLights(){
167     glShadeModel(GL_SMOOTH);    //Enable smooth shading of objects
168 
169     //Set modelview matrix for the lighting
170     glMatrixMode(GL_MODELVIEW);
171     glLoadIdentity();
172 
173     float lightPosition[] = {0.0, 10.0, 5.0};
174     float lightColor[] = {0.5, 0.5, 0.5};
175     float ambientColor[] = {0.3, 0.3, 0.3};
176     float spotDirection[] = {-1.0, -1.0, -1.0};
177 
178     glEnable(GL_LIGHTING);
179     glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientColor);
180 
181     glEnable(GL_LIGHT0);
182     glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
183     glLightfv(GL_LIGHT0, GL_AMBIENT, lightColor);
184     glLightfv(GL_LIGHT0, GL_DIFFUSE, lightColor);
185     glLightfv(GL_LIGHT0, GL_SPECULAR, lightColor);
186     glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 37.0);
187     glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotDirection);
188     glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 1);
189 }
```

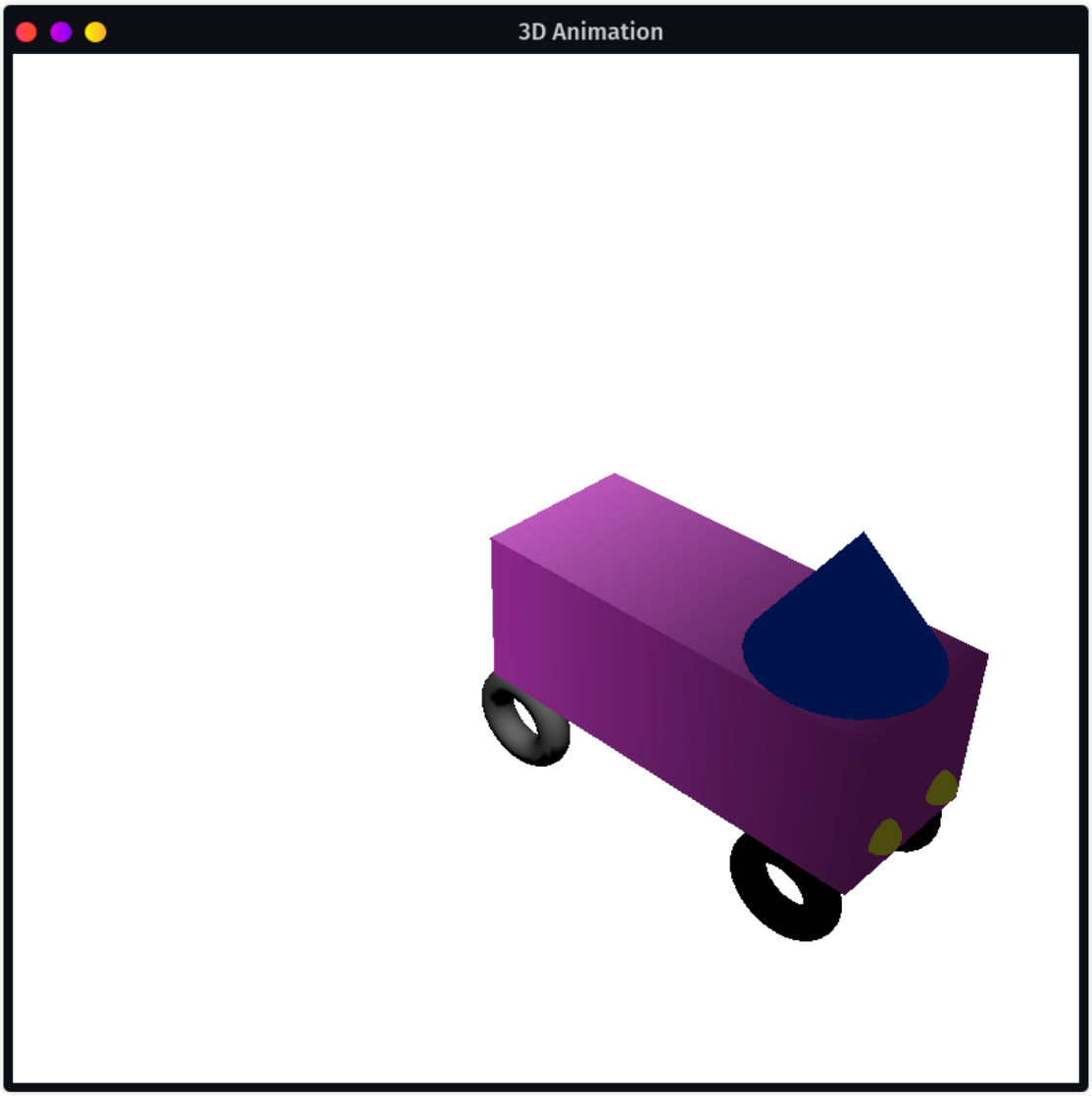
Output: Scene - 1

Figure 1: Scene - 1.



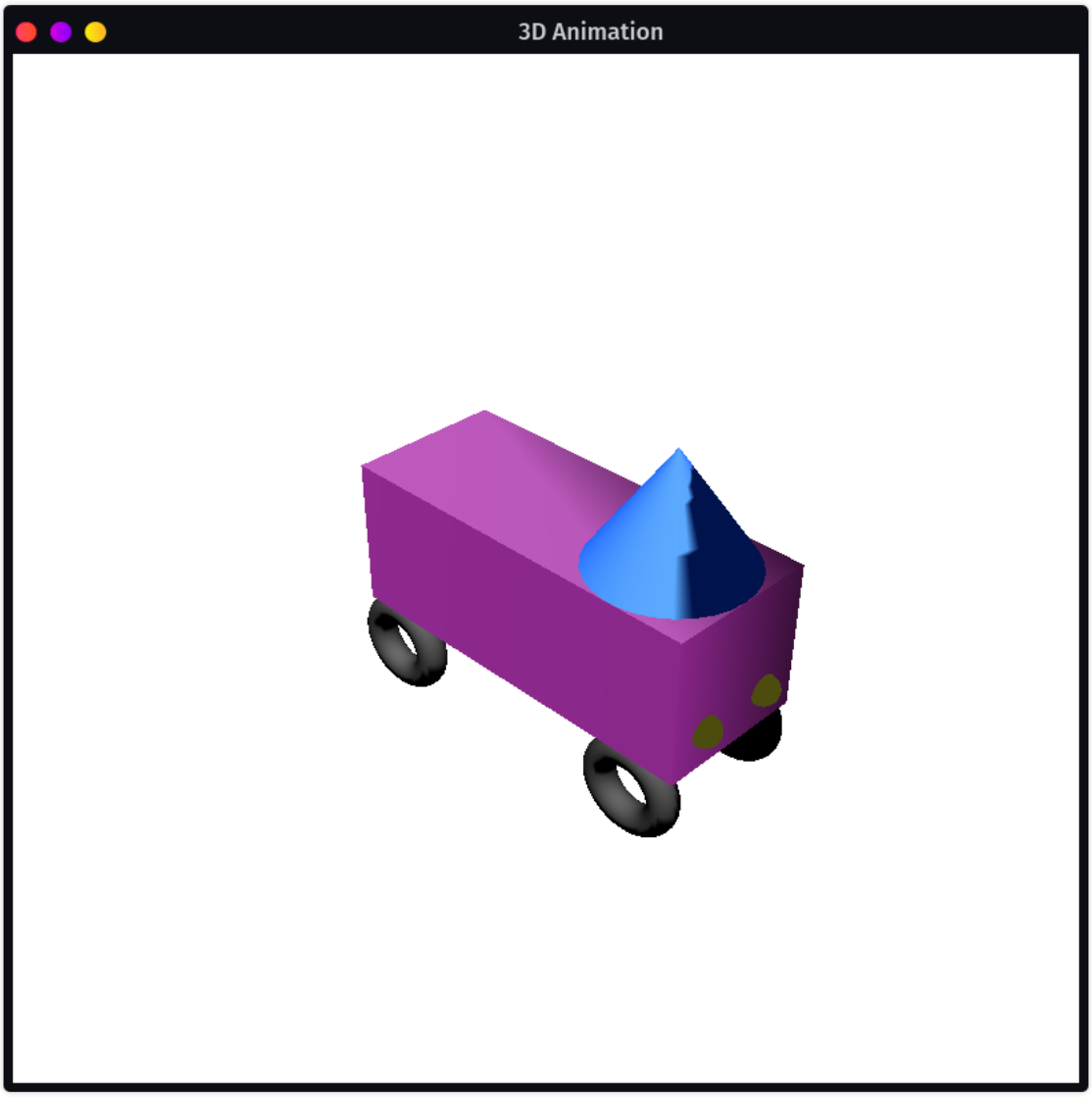
Output: Scene - 2

Figure 2: Scene - 2.



Output: Scene - 3

Figure 3: Scene - 3.



Learning Outcome:

- I learnt how to use in-built 3-D object drawing methods for generating a **Torus, Cube, Cone and Sphere**.
- I was able to apply in-built 3-D transformations to modify the objects & position them appropriately.
- I used these functions to draw a primitive 3D Car.
- I understood how to set different material parameters and transformations for different objects using the **glPushMatrix() and glPopMatrix()** methods.
- I understood how to animate the car object (perform translation) using an **FPS counter & translation variables along the X-axis** and **glutPostRedisplay()** to redraw the scene.
- I learnt how to define **material parameters** for **Ambient, Diffuse and Specular** components.
- I learnt how to set-up a basic lighting model & added **Ambient, Diffuse and Specular** components to it.
- I implemented **spot lighting and spot directionality** for the light using **GL_SPOT_DIRECTION & GL_SPOT_CUTOFF**.
- I was able to animate the car in and out of the lighting area.

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

20 October 2021

UCS 1712 - Graphics And Multimedia Lab

Exercise 11: Image Editing and Manipulation

Aim:

1. Using GIMP, include an image and apply filters, noise and masks.
2. Using GIMP, create a GIF animated image.

Input: Base Image

Figure 1: Base Image.

Output: Filter - Vignette

Figure 2: Filter: Vignette

Output: Filter - Lighting

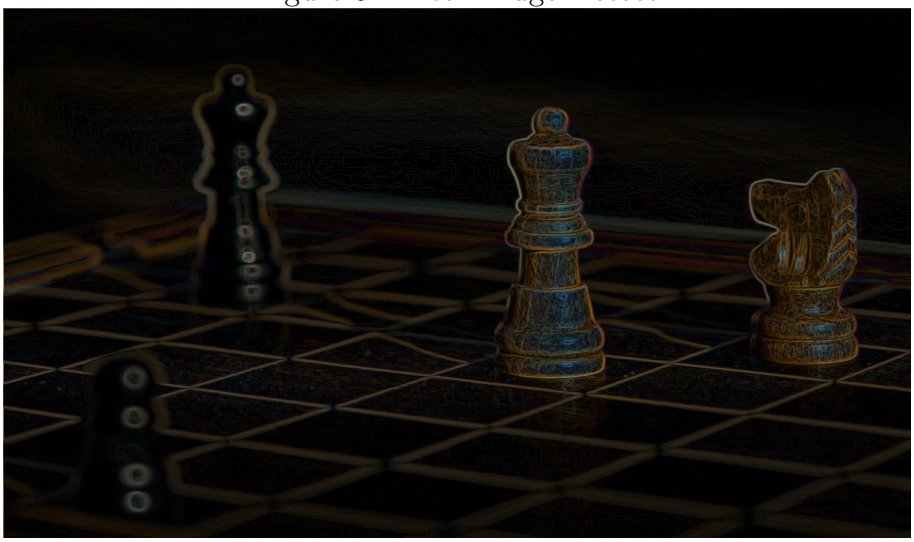
Figure 3: Filter: Lighting

Output: Filter - Gaussian Blur

Figure 4: Filter: Gaussian Blur

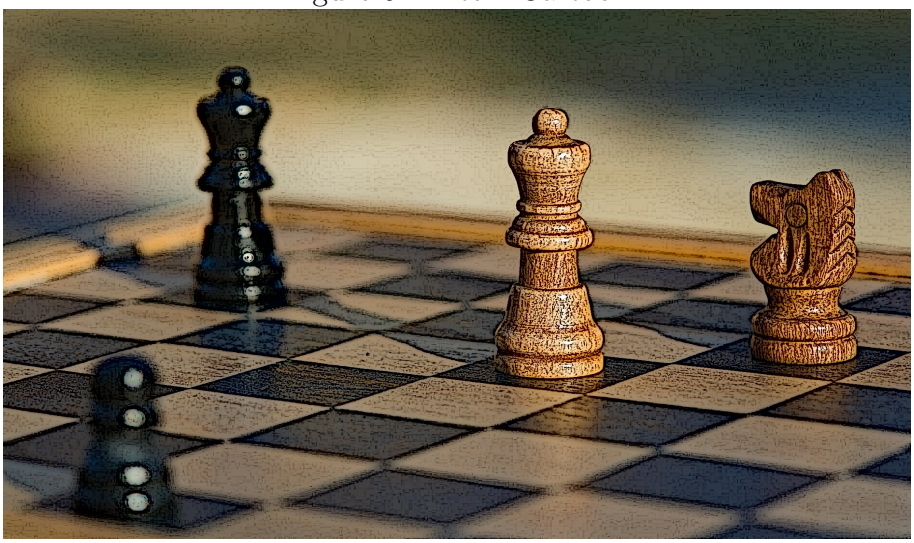
Output: Filter - Edge Detect

Figure 5: Filter: Edge Detect



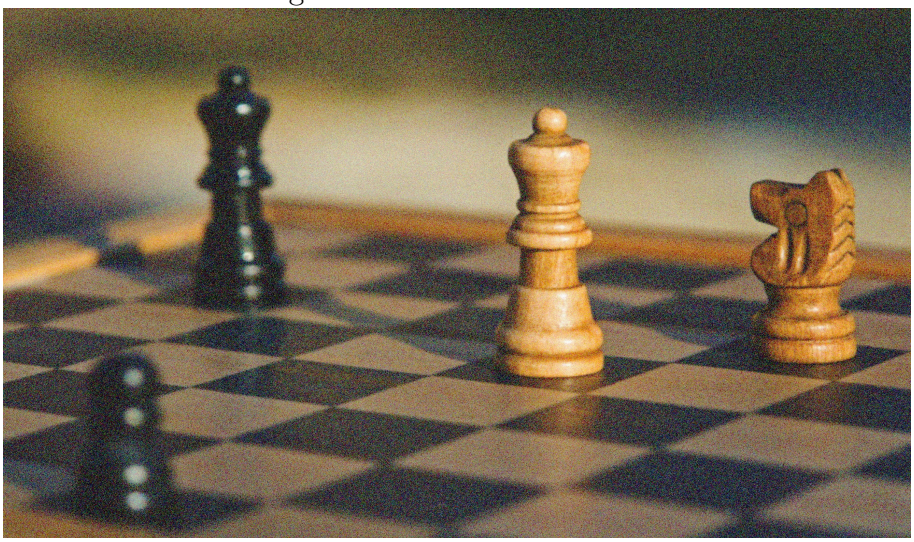
Output: Filter - Cartoon

Figure 6: Filter: Cartoon



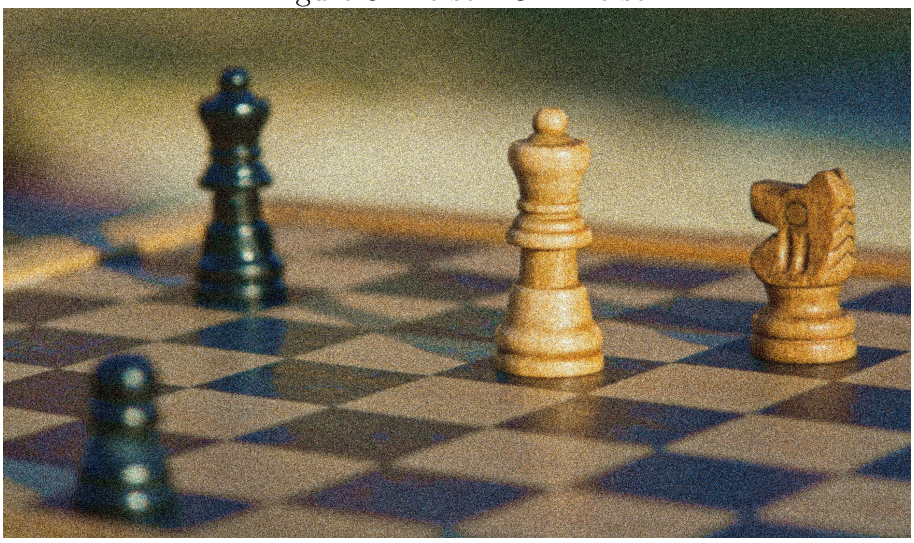
Output: Noise - RGB Noise

Figure 7: Noise - RGB Noise



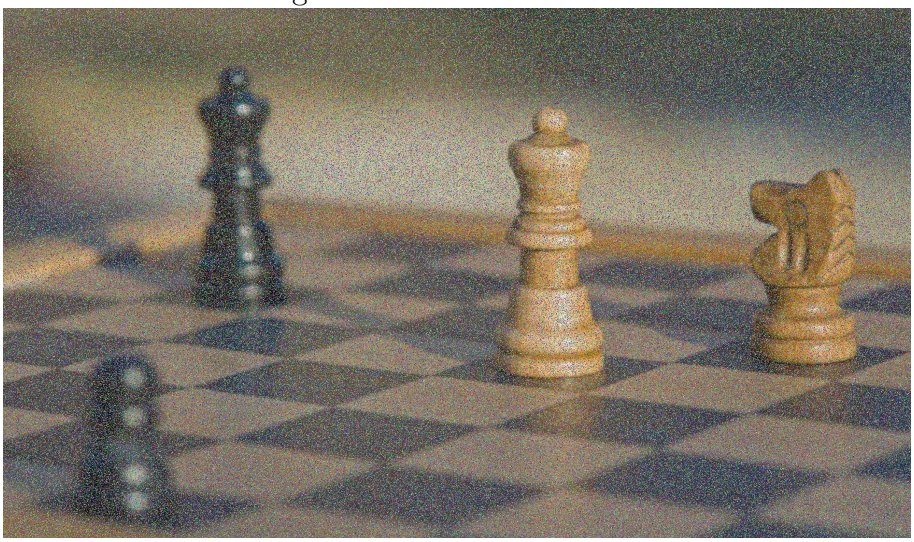
Output: Noise - CIE Noise

Figure 8: Noise - CIE Noise



Output: Noise - Hurl Noise

Figure 9: Noise - Hurl Noise



Input: Base Image

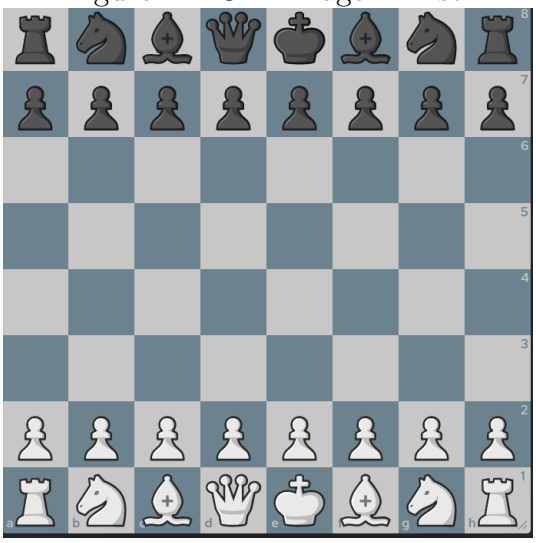
Figure 10: Base Image.

Output: Layer Mask

Figure 11: Layer Mask.

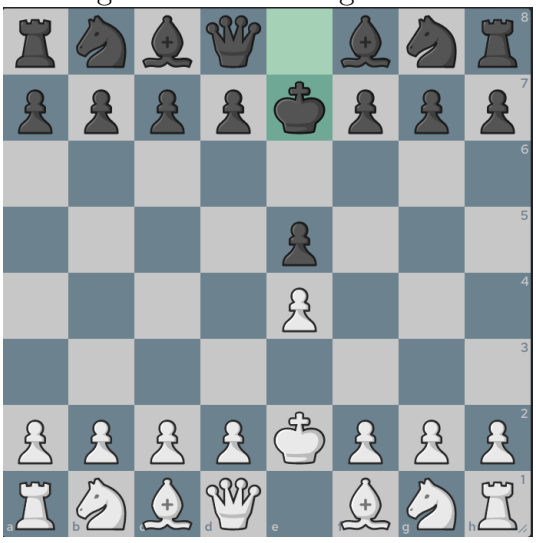
Input: GIF Image - First

Figure 12: GIF Image - First.



Input: GIF Image - Last

Figure 13: GIF Image - Last.



Learning Outcome:

- I learnt how to use the basic tools of **GIMP** like the **Paintbrush, Free Select, Rectangular Select, Eraser, Fill** etc.
- I understood how to **apply filters** to an image using GIMP.
- I learnt how to **apply noise** to an image using GIMP.
- I understood how to use **layer masks** to highlight specific areas of an image.
- I learnt how to create a **GIF Image** from a set of static images using GIMP.
- I understood how to control the **playback speed** of each image in the GIF.
- I learnt how to export the edited images/GIF from the GIMP editor to my computer.
- I learnt about different filters like **Lighting, Gaussian Blur, Vignette** etc.
- I also learnt about different types of noise like **RGB Noise, CIE Noise, Hurl Noise** etc.
- I learnt how to convert images to **grayscale** in GIMP.

Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VII

20 October 2021

UCS 1712 - Graphics And Multimedia Lab

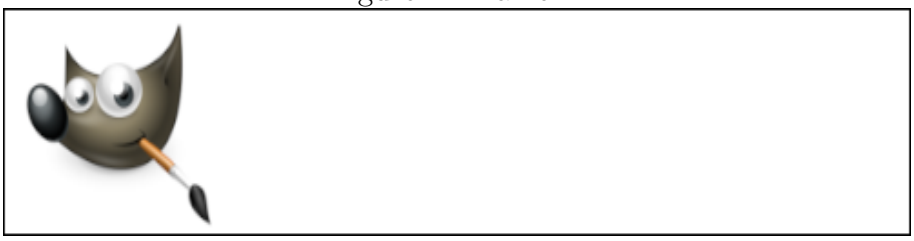
Exercise 12: Creating A 2D Animation

Aim:

1. Using GIMP, include layers and create a simple animation of your choice.

Input: Frame-1

Figure 1: Frame-1.

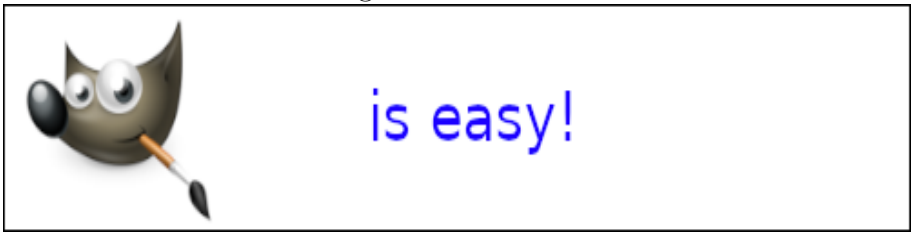


Input: Frame-2

Figure 2: Frame-2.

Input: Frame-3

Figure 3: Frame-3.



Output:

The frames were animated sequentially and exported as a **GIF** image, with each frame having a delay of 1000ms. The animation was viewed through a GIF Viewer.

Learning Outcome:

- I learnt how to create an empty banner image using GIMP.
- I learnt how to implement a border around the image using another layer.
- I learnt how to type in custom text and color it using the Text tool.
- I learnt how to import an external image as a layer using GIMP.
- I understood how to calculate delays for each frame and implement it in GIMP.
- I learnt how to scale an object while maintaining its aspect ratio using the Scale tool.
- I learnt how to animate the frames using the Filters $\rightarrow$ Animate filter.
- I learnt how to loop the animation forever and optimize it for a GIF using GIMP.