SSN COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UCS 1712 – GRAPHICS AND MULTIMEDIA LAB
MODEL PRACTICAL EXAMINATIONS

Name: S. Vishakan
Class: CSE – C
Register Number: 18 5001 196

Date: 29 Oct 2021
Session: Forenoon

AIM:

29) a. Construct a C++ program using OpenGL to draw a line and clip it with respect to bottom edge using Cohen-Sutherland Line Clipping Algorithm.

b. Draw a 3-D Wireframe Cone and animate it.

CODE:

a. Cohen-Sutherland Line Clipping Algorithm

```
/* To perform Cohen-Sutherland Line Clipping Algorithm*/

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

using namespace std;

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

//Region codes
const int LEFT = 0x1;
const int RIGHT = 0x2;
const int BOTTOM = 0x4;
const int TOP = 0x8;

class Point{   //Wrapper class for a 2D point
private:
          float x, y;

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

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

     //Getters & Setters
     void setCoords(float X, float Y){
          x = X; y = Y;
     }

     float getX(){
          return x;
     }

     float getY(){
          return y;
```

```
    }

    void setX(float X){
        x = X;
    }

    void setY(float Y){
        y = Y;
    }

    int getRC(Point windowMin, Point windowMax){
        //Returns region code for a given point
        int RC = 0;

        if(x < windowMin.getX()){
            RC = RC | LEFT;
        }

        if(x > windowMax.getX()){
            RC = RC | RIGHT;
        }

        if(y < windowMin.getY()){
            RC = RC | BOTTOM;
        }

        if(y > windowMax.getY()){
            RC = RC | TOP;
        }

        return RC;
    }
};

class Line{         //Wrapper class for a 2D Line
private:
    Point p, q;
    int RC1, RC2;

public:
    //Constructors
    Line(){
        p = Point(0, 0);
        q = Point(0, 0);
    }

    Line(Point P, Point Q){
        p = P;
        q = Q;
```

```
}

void getRCs(Point windowMin, Point windowMax){
     //Obtain region codes for the line endpoints
     RC1 = p.getRC(windowMin, windowMax);
     RC2 = q.getRC(windowMin, windowMax);
}

int trivialAccept(){
     //Check trivial accept case
     return (!(RC1 | RC2));
}

int trivialReject(){
     //Check trivial reject case
     return (RC1 & RC2);
}

int isPInside(){
     //Return true if P is inside clipping window
     return !RC1;
}

int isQInside(){
     //Return true if Q is inside clipping window
     return !RC2;
}

void swapPoints(){
     //Swap the line endpoints P & Q
     Point x = p;
     p = q;
     q = x;
}

void drawLine(bool clip=false){
     //Draw the line

     if(clip){
          glColor3d(0, 1, 0); //green
     } else{
          glColor3d(1, 0, 0); //red
     }

     glLineWidth(3);

     glBegin(GL_LINES);
     glVertex2f(p.getX(), p.getY());
     glVertex2f(q.getX(), q.getY());
```

```
            glEnd();

            glFlush();
      }

      bool cohenSutherlandLineClipping(Point windowMin, Point
windowMax){
            bool clip = false, done = false;


            while(!done){
                  getRCs(windowMin, windowMax);

                  //cout << "\nRC1: " << RC1 << "RC2: " << RC2;

                  if(trivialAccept()){
                        done = true;
                        continue;
                  }

                  if(trivialReject()){
                        done = true;
                        continue;
                  }

                  //Clipping will occur
                  clip = true;

                  if(isPInside()){
                        swapPoints();
                        getRCs(windowMin, windowMax);
                  }

                  //Slope Calculation
                  float dx = q.getX() - p.getX();
                  float dy = q.getY() - p.getY();

                  float m = dy / dx;

                  if(RC1 & LEFT){
                        p.setY(q.getY() + (windowMin.getX() - q.getX()) *
m);
                        p.setX(windowMin.getX());

                  }

                  else if(RC1 & RIGHT){
                        p.setY(q.getY() + (windowMax.getX() - q.getX()) *
m);
```

```
                    p.setX(windowMax.getX());

                }

                else if(RC1 & BOTTOM){
                    p.setX(q.getX() + (windowMin.getY() - q.getY()) *
(1 / m));
                    p.setY(windowMin.getY());
                }

                else if(RC1 & TOP){
                    p.setX(q.getX() + (windowMax.getY() - q.getY()) *
(1 / m));
                    p.setY(windowMax.getY());
                }
            }

            return clip;
        }
};


void initializeDisplay();
void renderContents();
void drawClippingWindow();

//Objects for Line and Clipping Window
Line l;
Point windowMin, windowMax;

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

    //Initialize the GLUT Primitives for Output
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
    glutCreateWindow("Cohen-Sutherland Line Clipping Algorithm");

    initializeDisplay();

    glutDisplayFunc(renderContents);
    glutMainLoop();

    return 1;
}
```

```
void initializeDisplay(){

     //To clear the display and set the matrix mode & projection
     glClearColor(1, 1, 1, 1);
     glClear(GL_COLOR_BUFFER_BIT);

     glMatrixMode(GL_PROJECTION);
     gluOrtho2D(0, WINDOW_WIDTH, 0, WINDOW_HEIGHT);
}

void renderContents(){
     //To render the contents to be shown in the output window

     //Clipping Window Coordinates
     windowMin = Point(100, 100);
     windowMax = Point(500, 500);

     //Initial Line Coordinates
     l = Line(Point(110, 40), Point(570, 550));

     drawClippingWindow();
     l.drawLine();
     bool clip = l.cohenSutherlandLineClipping(windowMin, windowMax);
     l.drawLine(clip);
}

void drawClippingWindow(){
     //To draw a rectangular clipping window

     glColor3d(0, 0, 1); //blue
     glLineWidth(3);

     glBegin(GL_LINE_LOOP);
     glVertex2f(windowMin.getX(), windowMin.getY());
     glVertex2f(windowMin.getX(), windowMax.getY());
     glVertex2f(windowMax.getX(), windowMax.getY());
     glVertex2f(windowMax.getX(), windowMin.getY());
     glEnd();

     glFlush();
}
```

OUTPUT:

a. Cohen-Sutherland Line Clipping Algorithm

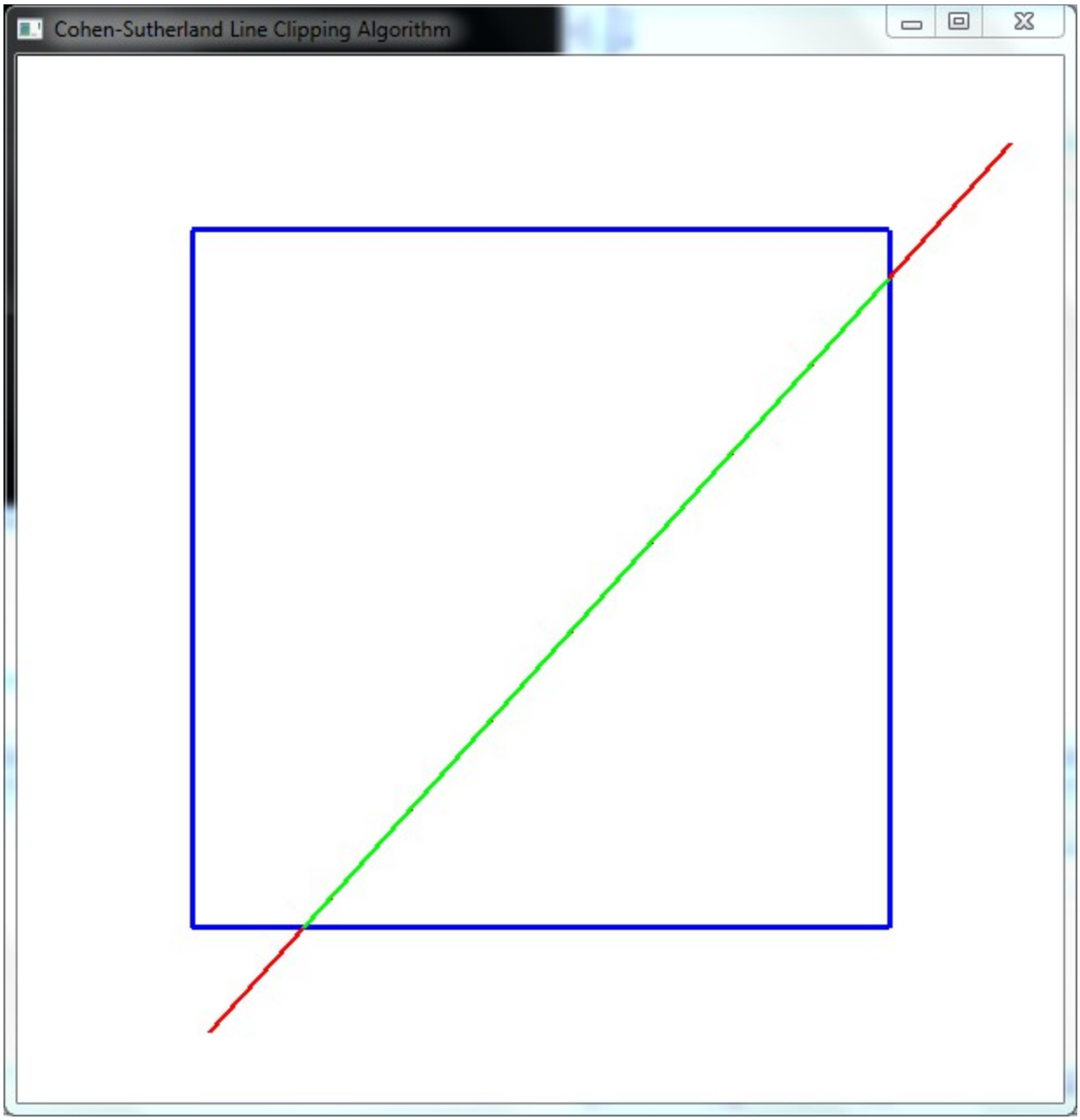

Figure 1: Initial Line – In Red, Clipped Line – In Green

Clipping Window: **(100, 100) to (500, 500)**
Line Coordinates: **P(110, 40) to Q(570, 550)**

CODE:

b. 3-D Wireframe Cone & Animation

```
/* To draw a wireframe cone and animate it*/

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

using namespace std;

const int WINDOW_WIDTH = 600;
const int WINDOW_HEIGHT = 600;
const int FPS = 30;

int iteration = 0;  //Variable to keep track of animation

void initializeDisplay();
void drawCone();
void mainLoop(int val);

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

     //Initialize the GLUT Primitives for Output
     glutInit(&argc, argv);
     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
     glutInitWindowPosition(100, 100);
     glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
     glutCreateWindow("Wireframe Cone");
     glEnable(GL_DEPTH_TEST);

     initializeDisplay();

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

     return 1;
}
```

```
void initializeDisplay(){

	//To clear the display and set the matrix mode & projection
	glClearColor(0, 1, 1, 1);
	glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

	glMatrixMode(GL_PROJECTION);
	//glOrtho(-2, 2, -2, 2, -2, 2);

	//Perspective Projection with 100 deg FoVy
	gluPerspective(100, 1, 0.01, 3);

	//Camera, Centre, Up Vector
	gluLookAt(1, 1, 1, 0, 0, 0, 0, 1, 0);

	glMatrixMode(GL_MODELVIEW);
}

void drawCone(){
	//To draw a cone and animate it

	glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
	glColor3d(1, 0, 1);

	//Animate the cone based on iteration value
	glPushMatrix();
	glRotatef(iteration, 1, 0, 0);		//About X axis
	glutWireCone(0.75, 1.25, 20, 20);
	glPopMatrix();

	glFlush();
}

void mainLoop(int val){

	//Callback function for the timer function
	drawCone();
	iteration = (iteration + 5) % 100000;
	glutPostRedisplay();
	glutTimerFunc(1000/FPS, mainLoop, 0);
	//Keep iterating the animation
}
```

OUTPUT:

b. 3-D Wireframe Cone & Animation

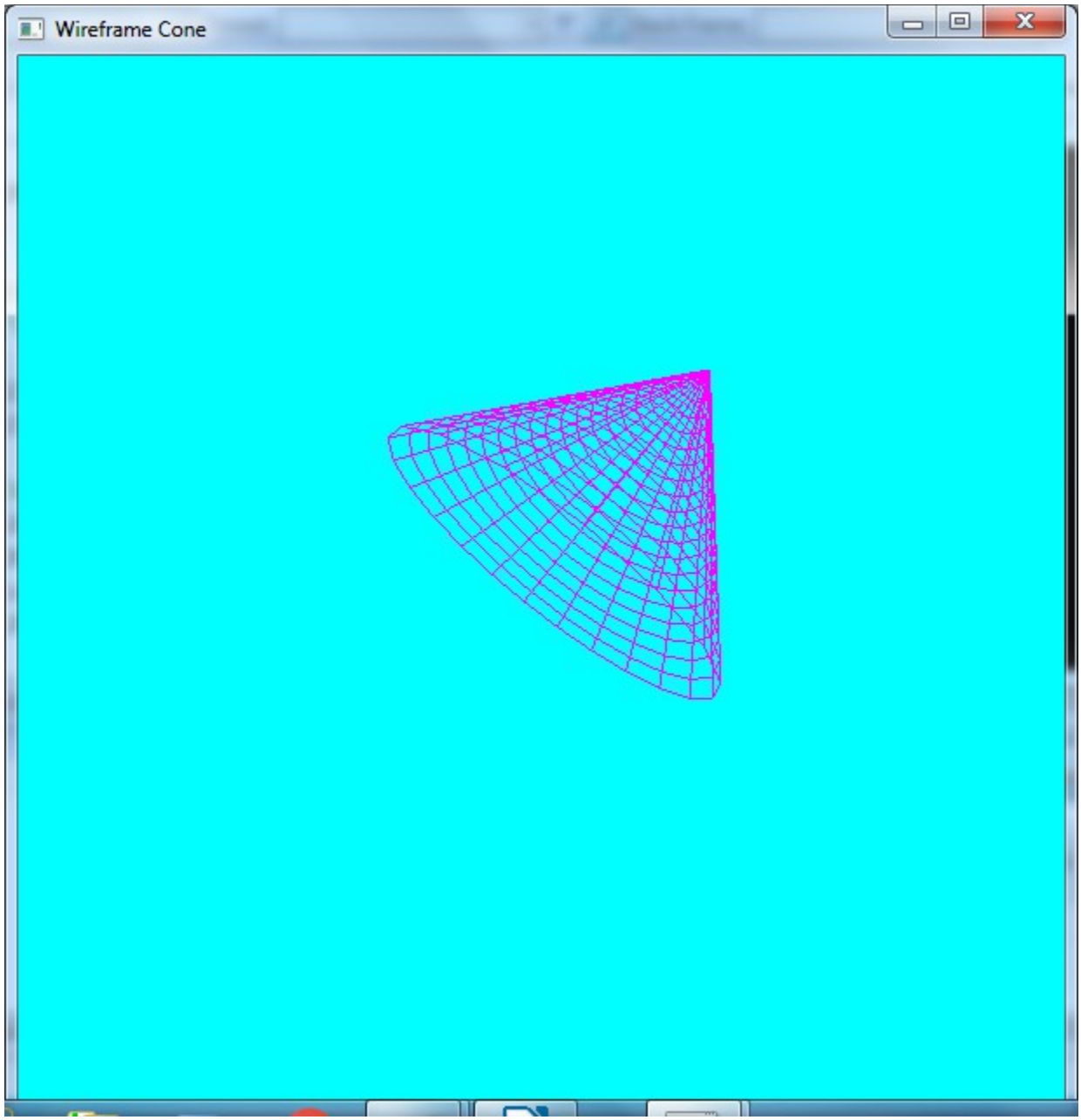

Figure 1: Initial Position

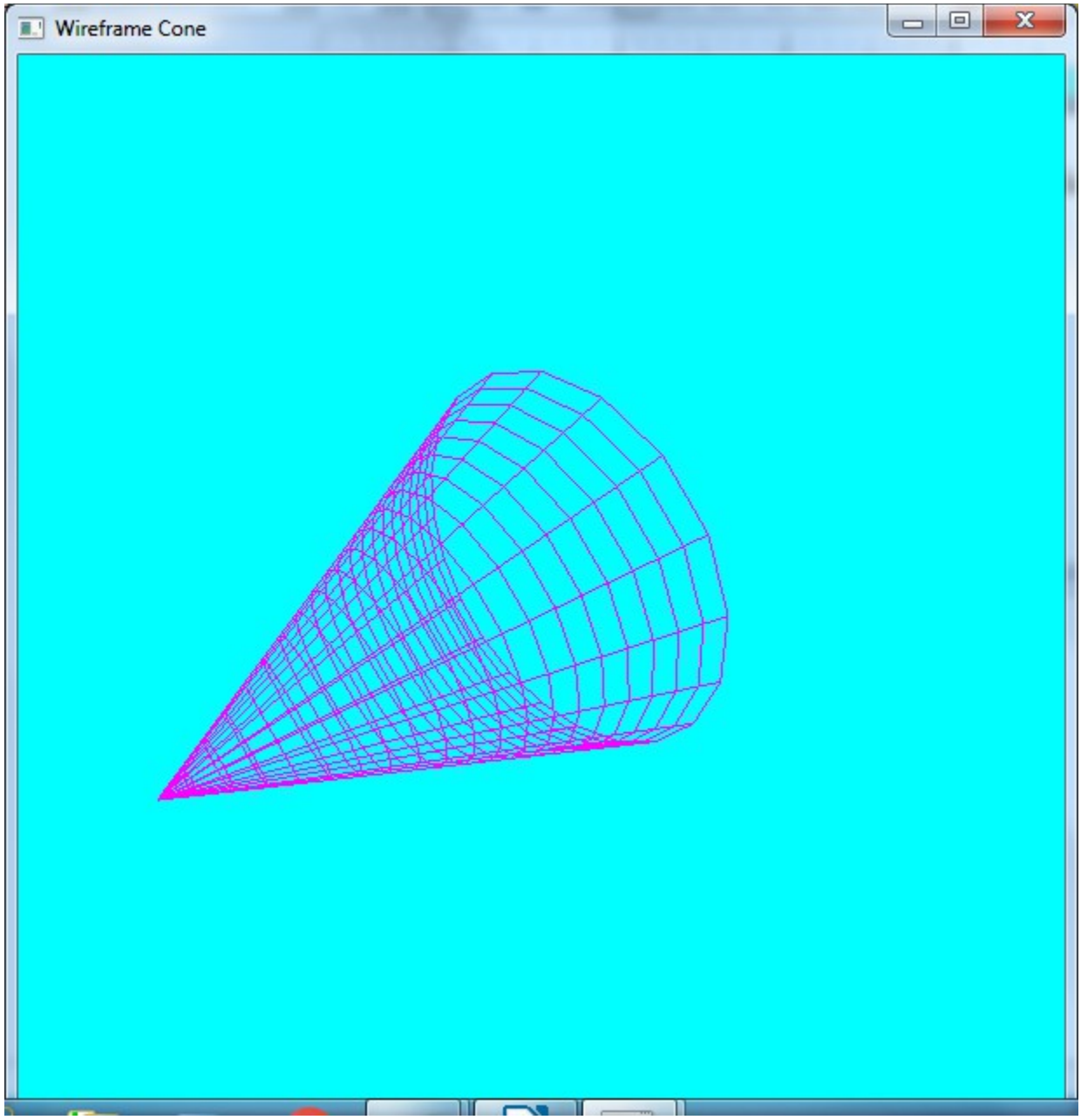

Figure 2: Rotated Position

RESULT:

The programs were constructed using OpenGL in C++ and the outputs were verified.