**SCHOOL OF COMPUTER SCIENCE**

**UNIVERSITY OF PETROLEUM AND ENERGY STUDIES**

**DEHRADUN, UTTARAKHAND**



**COMPUTER GRAPHICS**

**LABORATORY FILE**

**(2024-2025)**

**For**

**Vth Semester**

**Submitted To: Submitted By:**

Mr. Dinesh Akshat Negi

Assistant Professor 500106533(SAP ID)

[Vth Semester] R2142220414(Roll No.)

School of Computer Science B.Tech. CSF (Batch-1)

**LAB EXPERIMENT – 5**

**Viewing and Clipping**

**[Geographical Animation for demonstration]**

***# Take the window coordinates as input from the user, also take polygon coordinates as input.***

1. Write an interactive program for line clipping using Cohen Sutherland line clipping algorithm.

#include <GL/freeglut.h>

#include <iostream>

using namespace std;

// Defining region codes

const int INSIDE = 0; // 0000

const int LEFT = 1; // 0001

const int RIGHT = 2; // 0010

const int BOTTOM = 4; // 0100

const int TOP = 8; // 1000

// Defining x\_max, y\_max, x\_min, y\_min for clipping rectangle

const int x\_max = 250;

const int y\_max = 250;

const int x\_min = 150;

const int y\_min = 150;

// Function to compute region code for a point (x, y)

int computeCode(double x, double y) {

int code = INSIDE;

if (x < x\_min) code |= LEFT;

else if (x > x\_max) code |= RIGHT;

if (y < y\_min) code |= BOTTOM;

else if (y > y\_max) code |= TOP;

return code;

}

// Function to draw the clipping boundary

void drawBoundary() {

glColor3f(1.0, 0.0, 0.0); // Red color for the boundary

glBegin(GL\_LINE\_LOOP);

glVertex2f(x\_min, y\_min);

glVertex2f(x\_max, y\_min);

glVertex2f(x\_max, y\_max);

glVertex2f(x\_min, y\_max);

glEnd();

}

// Function to draw a line with specific color

void drawLine(float x1, float y1, float x2, float y2, float r, float g, float b) {

glColor3f(r, g, b); // Set the color for the line

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

// Cohen-Sutherland Line Clipping Algorithm

void cohenSutherlandClip(double x1, double y1, double x2, double y2) {

int code1 = computeCode(x1, y1);

int code2 = computeCode(x2, y2);

bool accept = false;

while (true) {

if ((code1 == 0) && (code2 == 0)) {

// If both endpoints lie within the rectangle

accept = true;

break;

}

else if (code1 & code2) {

// If both endpoints are outside the rectangle in the same region

break;

}

else {

// Some segment of the line lies within the rectangle

int code\_out;

double x, y;

if (code1 != 0) code\_out = code1;

else code\_out = code2;

// Find intersection point

if (code\_out & TOP) {

x = x1 + (x2 - x1) \* (y\_max - y1) / (y2 - y1);

y = y\_max;

}

else if (code\_out & BOTTOM) {

x = x1 + (x2 - x1) \* (y\_min - y1) / (y2 - y1);

y = y\_min;

}

else if (code\_out & RIGHT) {

y = y1 + (y2 - y1) \* (x\_max - x1) / (x2 - x1);

x = x\_max;

}

else if (code\_out & LEFT) {

y = y1 + (y2 - y1) \* (x\_min - x1) / (x2 - x1);

x = x\_min;

}

// Replace the point outside the rectangle with the intersection point

if (code\_out == code1) {

x1 = x;

y1 = y;

code1 = computeCode(x1, y1);

}

else {

x2 = x;

y2 = y;

code2 = computeCode(x2, y2);

}

}

}

if (accept) {

cout << "Line accepted from (" << x1 << ", " << y1 << ") to (" << x2 << ", " << y2 << ")\n";

drawLine(x1, y1, x2, y2, 1.0, 0.0, 0.0); // Draw the clipped line in red

}

else {

cout << "Line rejected\n";

}

}

// Function to display the content

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT); // Clear the screen

drawBoundary(); // Draw the clipping boundary

// Prompt the user for input coordinates

double x1, y1, x2, y2;

cout << "Enter coordinates for the line (x1, y1, x2, y2): ";

cin >> x1 >> y1 >> x2 >> y2;

drawLine(x1, y1, x2, y2, 0.0, 1.0, 0.0); // Draw the original line in green

cout << "Press any key to clip the line.\n";

cohenSutherlandClip(x1, y1, x2, y2); // Clip the line

glFlush(); // Render now

}

// Function to set up OpenGL projection and modelview matrices

void init() {

glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 400, 0, 400); // Define the 2D orthographic projection

}

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

// Initialize GLUT

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 800); // Set the window size

glutCreateWindow("Cohen Sutherland Line Clipping - Akshat Negi"); // Create the window

init(); // Initialize OpenGL state

// Register display callback function

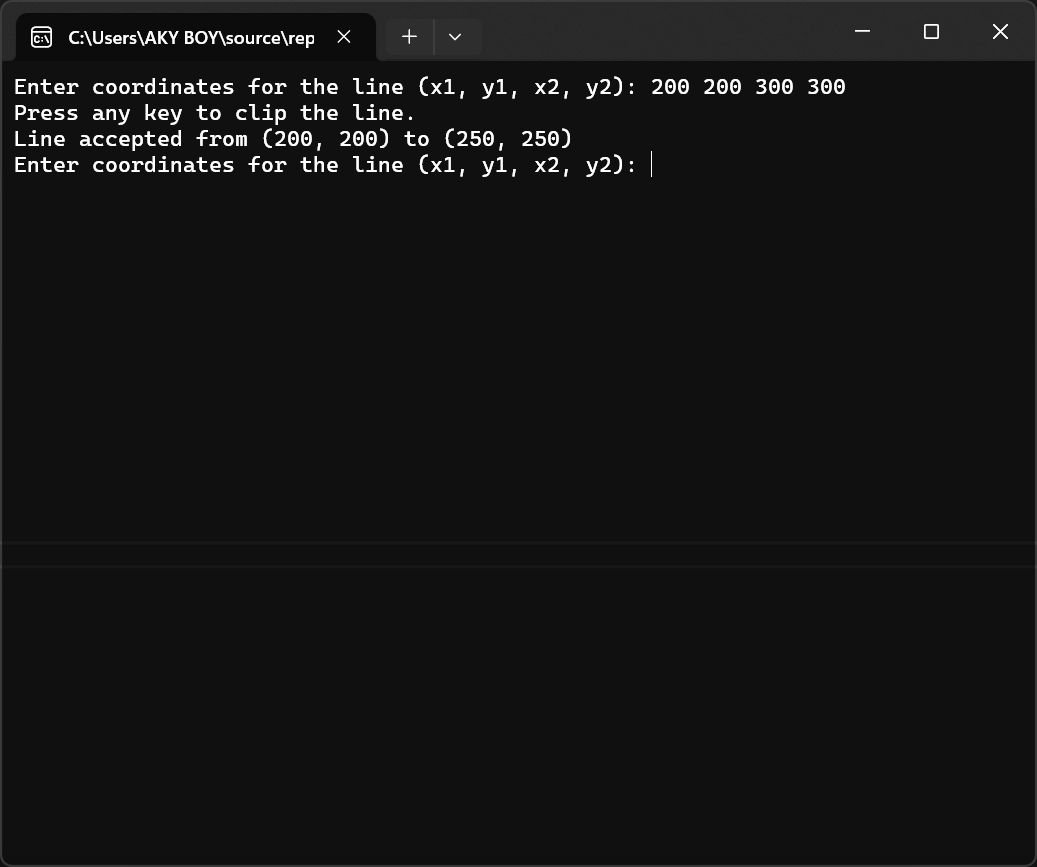
glutDisplayFunc(display);

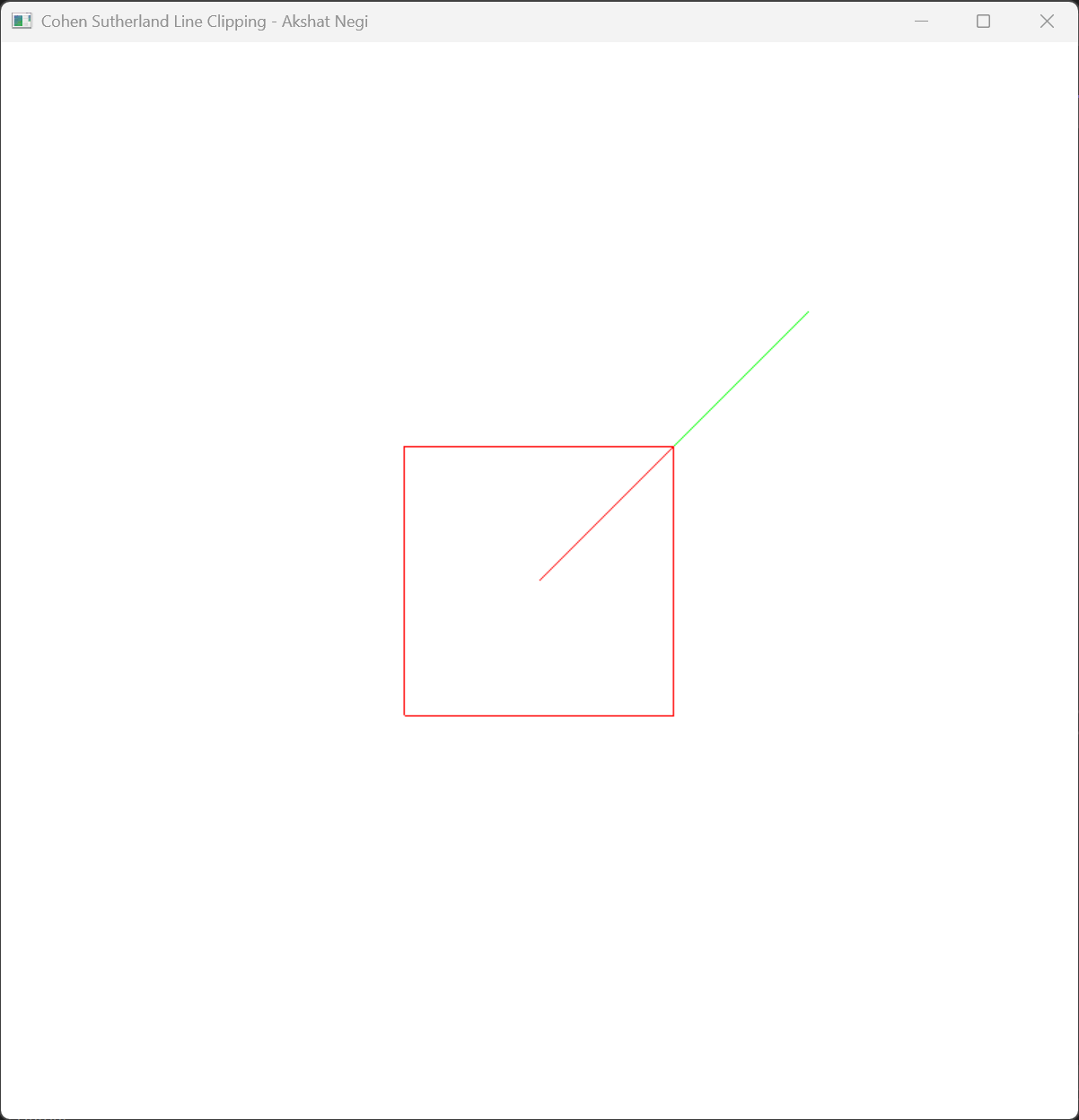
// Enter the GLUT event processing loop

glutMainLoop();

return 0;

}





1. Write an interactive program for line clipping using Liang-Barsky line clipping algorithm.

#include <GL/freeglut.h>

#include <iostream>

using namespace std;

// Defining the clipping window boundaries

const int x\_min = 10;

const int x\_max = 200;

const int y\_min = 10;

const int y\_max = 200;

// Function to draw a line with specified color

void drawLine(float x1, float y1, float x2, float y2, float r, float g, float b) {

glColor3f(r, g, b); // Set color for the line

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

// Liang-Barsky Line Clipping Algorithm

bool liangBarskyClip(double& x1, double& y1, double& x2, double& y2) {

double t0 = 0.0, t1 = 1.0;

double dx = x2 - x1;

double dy = y2 - y1;

auto clipTest = [&](double p, double q) {

if (p == 0) {

if (q < 0) return false;

}

else {

double t = q / p;

if (p < 0) {

if (t > t1) return false;

if (t > t0) t0 = t;

}

else {

if (t < t0) return false;

if (t < t1) t1 = t;

}

}

return true;

};

// Clip against each boundary

if (!clipTest(-dx, x1 - x\_min) || !clipTest(dx, x\_max - x1) ||

!clipTest(-dy, y1 - y\_min) || !clipTest(dy, y\_max - y1)) {

return false;

}

// Update the points based on t0 and t1

if (t1 < 1.0) {

x2 = x1 + t1 \* dx;

y2 = y1 + t1 \* dy;

}

if (t0 > 0.0) {

x1 = x1 + t0 \* dx;

y1 = y1 + t0 \* dy;

}

return true;

}

// Function to display the content

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT); // Clear the screen

// Draw the clipping boundary (rectangle)

glColor3f(1.0, 0.0, 0.0); // Red color for the boundary

glBegin(GL\_LINE\_LOOP);

glVertex2f(x\_min, y\_min);

glVertex2f(x\_max, y\_min);

glVertex2f(x\_max, y\_max);

glVertex2f(x\_min, y\_max);

glEnd();

// Prompt the user for input coordinates

double x1, y1, x2, y2;

cout << "Enter coordinates for the line (x1, y1, x2, y2): ";

cin >> x1 >> y1 >> x2 >> y2;

// Draw the original line in green

drawLine(x1, y1, x2, y2, 0.0, 1.0, 0.0);

// Apply Liang-Barsky clipping

bool isClipped = liangBarskyClip(x1, y1, x2, y2);

if (isClipped) {

// Draw the clipped line in blue and print clipped coordinates

cout << "Clipped line from (" << x1 << ", " << y1 << ") to (" << x2 << ", " << y2 << ")\n";

drawLine(x1, y1, x2, y2, 0.0, 0.0, 1.0); // Draw the clipped line in blue

}

else {

// Line is outside the window

cout << "Line is outside the clipping window.\n";

}

glFlush(); // Render now

}

// Function to set up OpenGL projection and modelview matrices

void init() {

glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 400, 0, 400); // Define the 2D orthographic projection

}

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

// Initialize GLUT

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 800); // Set the window size

glutCreateWindow("Liang-Barsky Line Clipping - Akshat Negi"); // Create the window

init(); // Initialize OpenGL state

// Register display callback function

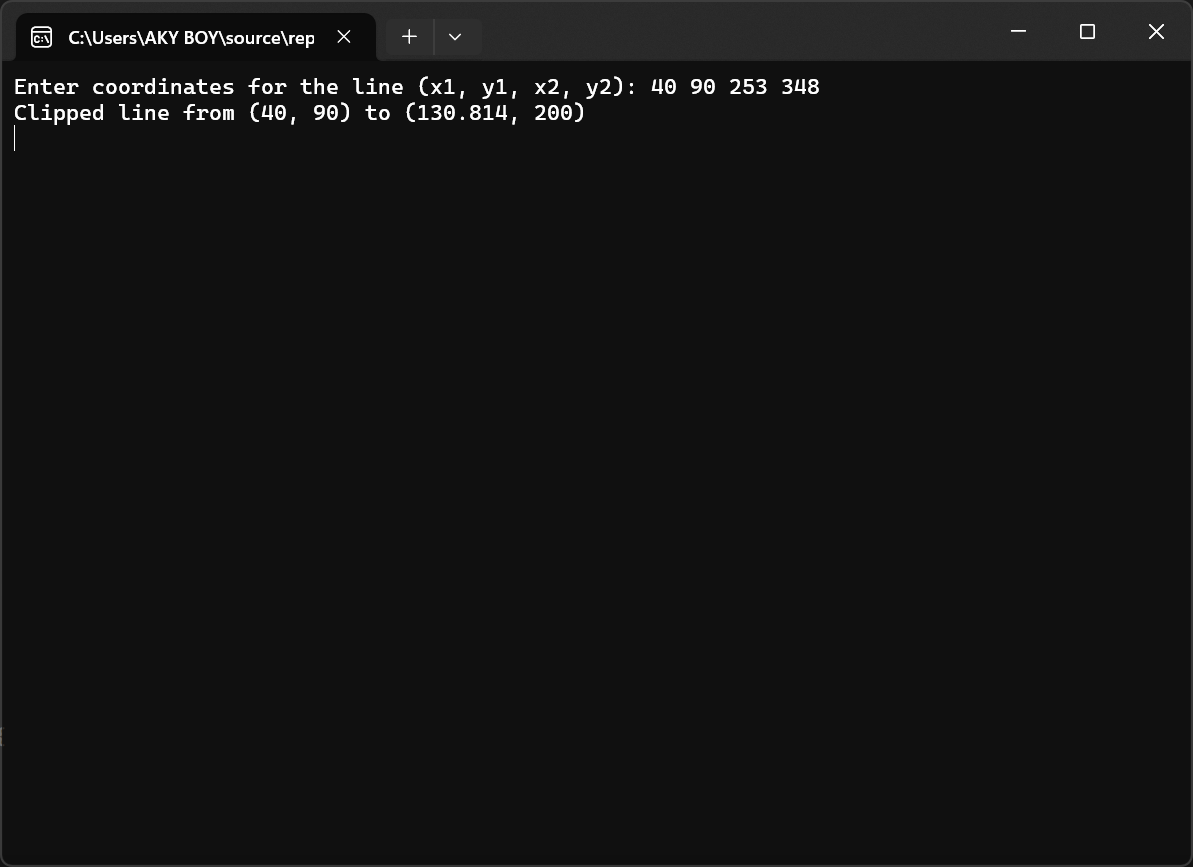
glutDisplayFunc(display);

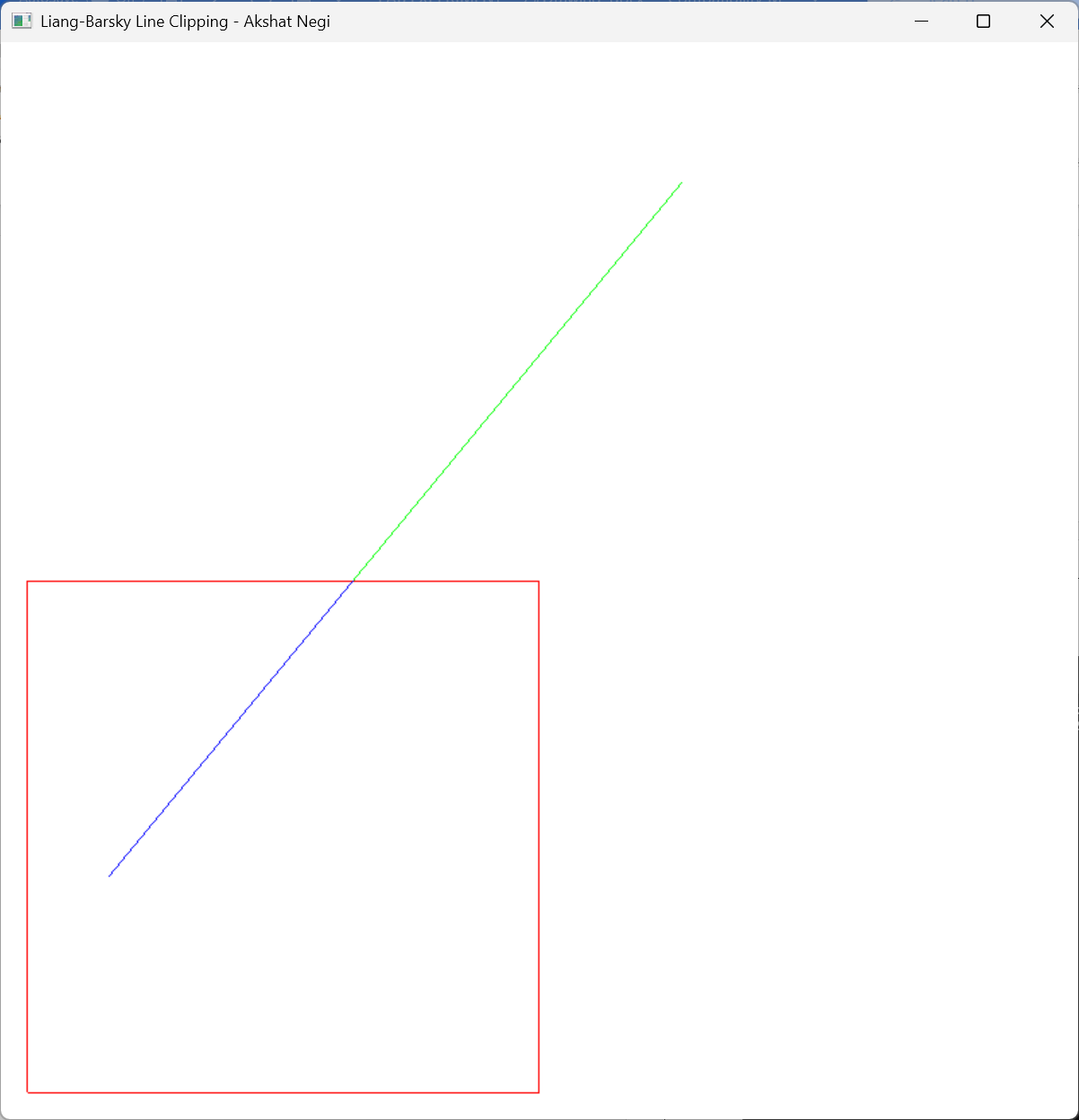
// Enter the GLUT event processing loop

glutMainLoop();

return 0;

}





1. Write an interactive program for polygon clipping using Sutherland – Hodgeman polygon clipping algorithm.

#include <GL/freeglut.h>

#include <iostream>

#include <vector>

using namespace std;

struct Point {

float x, y;

};

// Global variables for the clipping window boundaries

int x\_min, y\_min, x\_max, y\_max;

vector<Point> polygon;

vector<Point> clippedPolygon;

// Function to draw a polygon

void drawPolygon(const vector<Point>& poly, float r, float g, float b) {

glColor3f(r, g, b);

glBegin(GL\_LINE\_LOOP);

for (const auto& p : poly) {

glVertex2f(p.x, p.y);

}

glEnd();

}

// Function to check if a point is inside the clipping boundary

bool inside(const Point& p, int edge) {

switch (edge) {

case 0: return p.x >= x\_min; // Left

case 1: return p.x <= x\_max; // Right

case 2: return p.y >= y\_min; // Bottom

case 3: return p.y <= y\_max; // Top

}

return true;

}

// Function to compute the intersection point with a clipping edge

Point intersect(const Point& p1, const Point& p2, int edge) {

Point intersection;

float m;

if (p2.x != p1.x)

m = (p2.y - p1.y) / (p2.x - p1.x); // Slope of the line

switch (edge) {

case 0: // Left edge

intersection.x = x\_min;

intersection.y = p1.y + m \* (x\_min - p1.x);

break;

case 1: // Right edge

intersection.x = x\_max;

intersection.y = p1.y + m \* (x\_max - p1.x);

break;

case 2: // Bottom edge

intersection.y = y\_min;

if (p2.x != p1.x)

intersection.x = p1.x + (y\_min - p1.y) / m;

else

intersection.x = p1.x;

break;

case 3: // Top edge

intersection.y = y\_max;

if (p2.x != p1.x)

intersection.x = p1.x + (y\_max - p1.y) / m;

else

intersection.x = p1.x;

break;

}

return intersection;

}

// Sutherland-Hodgman Polygon Clipping Algorithm

vector<Point> sutherlandHodgmanClip(const vector<Point>& input, int edge) {

vector<Point> output;

Point s = input.back(); // Start with the last point

for (const auto& e : input) {

if (inside(e, edge)) { // Case 1: End point is inside

if (!inside(s, edge)) // Case 1.1: Start point is outside

output.push\_back(intersect(s, e, edge)); // Add intersection point

output.push\_back(e); // Add end point

}

else if (inside(s, edge)) { // Case 2: End point is outside, start is inside

output.push\_back(intersect(s, e, edge)); // Add intersection point

}

s = e;

}

return output;

}

// Clipping function to clip the polygon against all four edges

void clipPolygon() {

clippedPolygon = polygon;

for (int edge = 0; edge < 4; edge++) {

clippedPolygon = sutherlandHodgmanClip(clippedPolygon, edge);

}

// Print the clipped polygon points to the console

cout << "Clipped Polygon Points:\n";

for (const auto& point : clippedPolygon) {

cout << "(" << point.x << ", " << point.y << ")\n";

}

}

// Display function to render the polygons

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

// Draw clipping window (rectangle)

glColor3f(1.0, 0.0, 0.0); // Red color for the boundary

glBegin(GL\_LINE\_LOOP);

glVertex2f(x\_min, y\_min);

glVertex2f(x\_max, y\_min);

glVertex2f(x\_max, y\_max);

glVertex2f(x\_min, y\_max);

glEnd();

// Draw original polygon

drawPolygon(polygon, 0.0f, 1.0f, 0.0f); // Green color

// Draw clipped polygon

drawPolygon(clippedPolygon, 0.0f, 0.0f, 1.0f); // Blue color

glFlush();

}

// Keyboard callback function

void handleKeypress(unsigned char key, int x, int y) {

if (key == 'c') {

clipPolygon();

glutPostRedisplay(); // Request redisplay

}

else if (key == 27) { // ESC key

exit(0);

}

}

// Setup OpenGL

void initGL() {

glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 400, 0, 400); // Define 2D orthographic projection

}

void inputPolygon() {

int numVertices;

cout << "Enter number of vertices for the polygon: ";

cin >> numVertices;

polygon.clear();

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

Point p;

cout << "Enter vertex " << i + 1 << " (x, y): ";

cin >> p.x >> p.y;

polygon.push\_back(p);

}

}

void inputClippingWindow() {

cout << "Enter the clipping window coordinates:\n";

cout << "x\_min, y\_min: ";

cin >> x\_min >> y\_min;

cout << "x\_max, y\_max: ";

cin >> x\_max >> y\_max;

}

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

// User inputs

inputClippingWindow();

inputPolygon();

// Initialize GLUT and display

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(600, 600);

glutCreateWindow("Sutherland-Hodgman Polygon Clipping - Akshat Negi");

initGL();

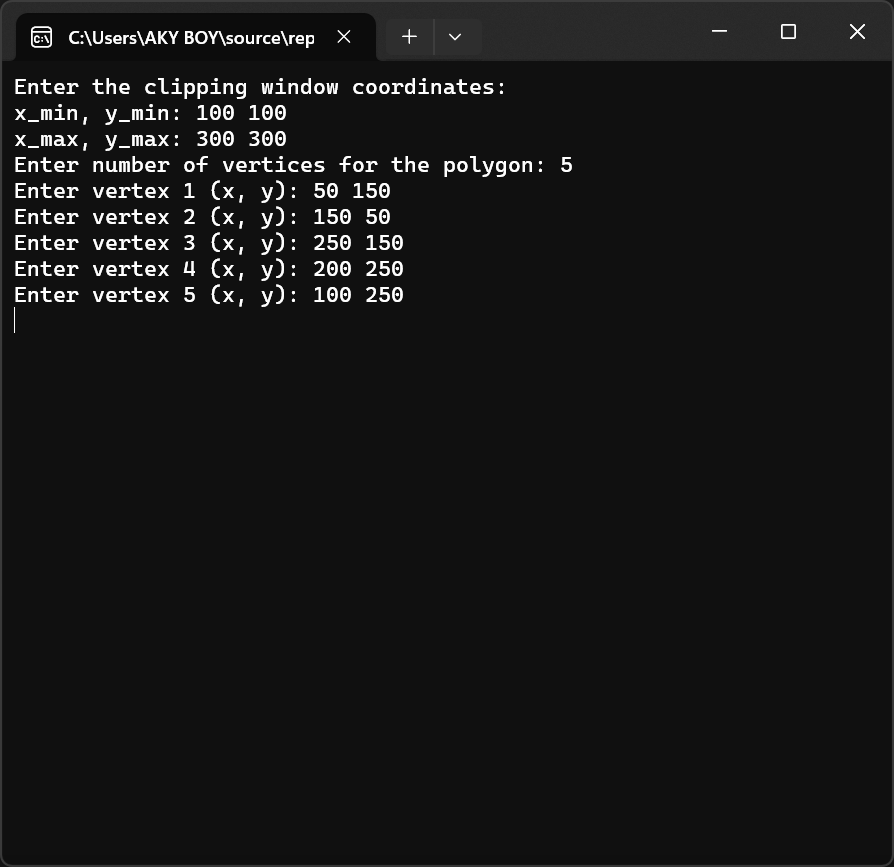
glutDisplayFunc(display);

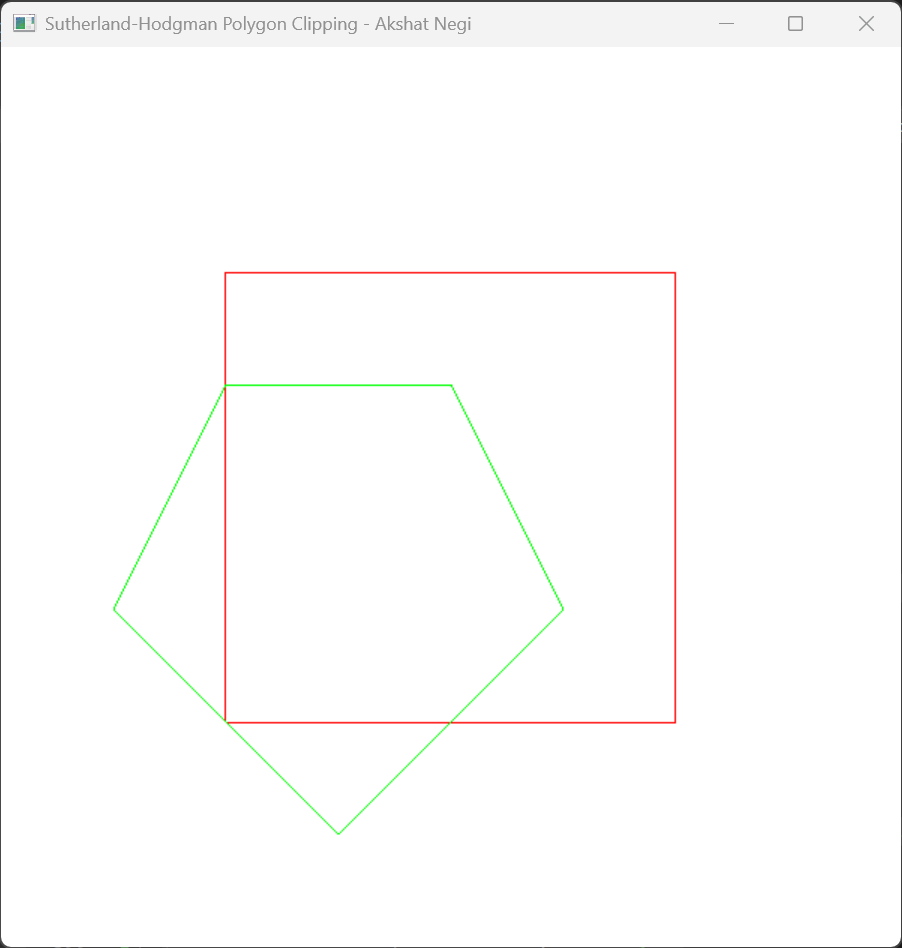
glutKeyboardFunc(handleKeypress);

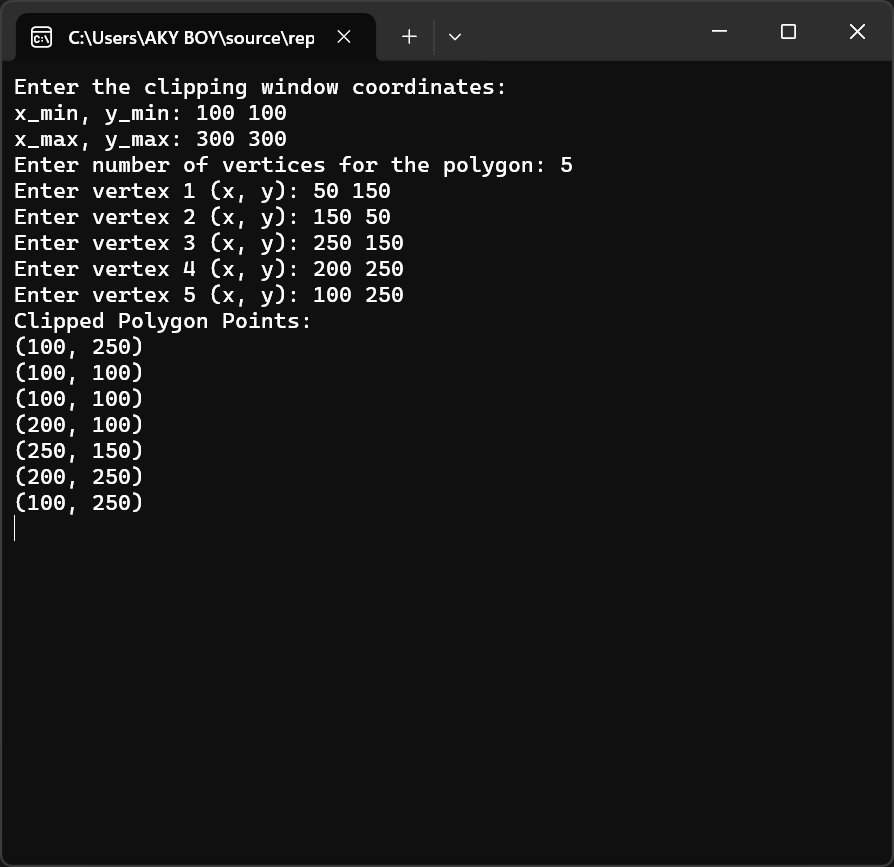
glutMainLoop();

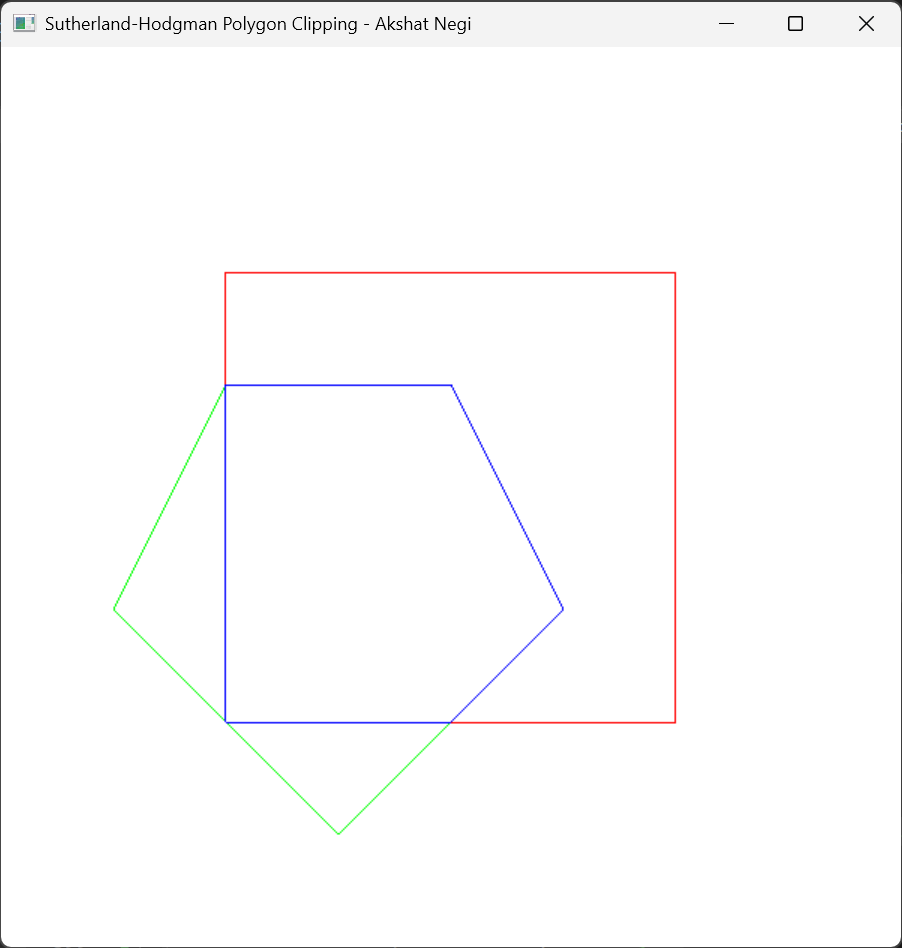
return 0;

}

****

****

****

****