

| **TITLE**: Write a program to demonstrate the Polygon CLIPPING algorithm |
| --- |

**AIM:**

Write a program to demonstrate the Polygon CLIPPING algorithm

(Implement using Sutherland Hodgeman polygon clipping algorithm)

VLab

[**https://cse18-iiith.vlabs.ac.in/exp/clipping-polygon/**](https://cse18-iiith.vlabs.ac.in/exp/clipping-polygon/)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

Implement Clipping,3D Geometric Transformations and 3D viewing

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

Google

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Algorithm/ Pseudocode for each process:**

function SutherlandHodgman(polygon, clippingWindow)

clippedPolygon = polygon

for each edge in clippingWindow

newPolygon = empty list

for i = 0 to size(clippedPolygon) - 1

currentVertex = clippedPolygon[i]

previousVertex = clippedPolygon[(i - 1) % size(clippedPolygon)]

if currentVertex is inside edge

if previousVertex is not inside edge

intersection = findIntersection(previousVertex, currentVertex, edge)

newPolygon.append(intersection)

end if

newPolygon.append(currentVertex)

else if previousVertex is inside edge

intersection = findIntersection(previousVertex, currentVertex, edge)

newPolygon.append(intersection)

end if

end for

clippedPolygon = newPolygon

end for

return clippedPolygon

end function

**Implementation details:**

#include <GL/glut.h>

#include <vector>

struct Point {

float x, y;

};

vector<Point> polygon = {{150, 150}, {300, 100}, {350, 300}, {250, 350}, {100, 250}};

vector<Point> clippingWindow = {{200, 200}, {300, 200}, {300, 300}, {200, 300}};

bool isInside(Point p, Point p1, Point p2) {

return (p2.x - p1.x) \* (p.y - p1.y) - (p2.y - p1.y) \* (p.x - p1.x) <= 0;

}

Point getIntersection(Point p1, Point p2, Point clip1, Point clip2) {

float A1 = p2.y - p1.y;

float B1 = p1.x - p2.x;

float C1 = A1 \* p1.x + B1 \* p1.y;

float A2 = clip2.y - clip1.y;

float B2 = clip1.x - clip2.x;

float C2 = A2 \* clip1.x + B2 \* clip1.y;

float det = A1 \* B2 - A2 \* B1;

Point intersection;

intersection.x = (B2 \* C1 - B1 \* C2) / det;

intersection.y = (A1 \* C2 - A2 \* C1) / det;

return intersection;

}

std::vector<Point> SutherlandHodgman(std::vector<Point> polygon, std::vector<Point> clippingWindow) {

for (size\_t i = 0; i < clippingWindow.size(); i++) {

std::vector<Point> newPolygon;

Point clip1 = clippingWindow[i];

Point clip2 = clippingWindow[(i + 1) % clippingWindow.size()];

for (size\_t j = 0; j < polygon.size(); j++) {

Point current = polygon[j];

Point previous = polygon[(j + polygon.size() - 1) % polygon.size()];

if (isInside(current, clip1, clip2)) {

if (!isInside(previous, clip1, clip2)) {

newPolygon.push\_back(getIntersection(previous, current, clip1, clip2));

}

newPolygon.push\_back(current);

} else if (isInside(previous, clip1, clip2)) {

newPolygon.push\_back(getIntersection(previous, current, clip1, clip2));

}

}

polygon = newPolygon;

}

return polygon;

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

// Draw original polygon in red

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

for (const auto& point : polygon) {

glVertex2f(point.x, point.y);

}

glEnd();

std::vector<Point> clippedPolygon = SutherlandHodgman(polygon, clippingWindow);

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_POLYGON);

for (const auto& point : clippedPolygon) {

glVertex2f(point.x, point.y);

}

glEnd();

// Draw clipping window in blue

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_LOOP);

for (const auto& point : clippingWindow) {

glVertex2f(point.x, point.y);

}

glEnd();

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Polygon Clipping");

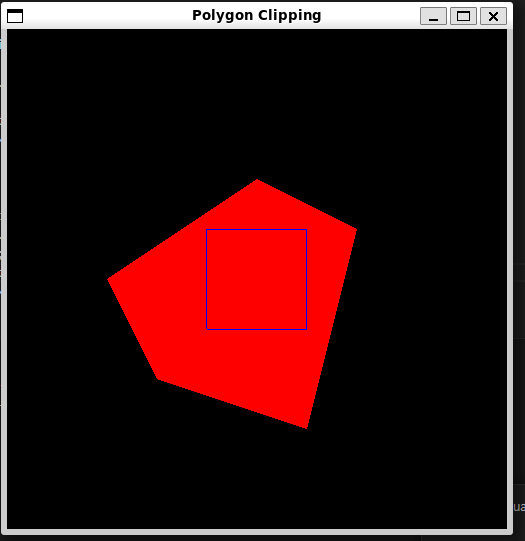
gluOrtho2D(0, 500, 0, 500);

glutDisplayFunc(display);

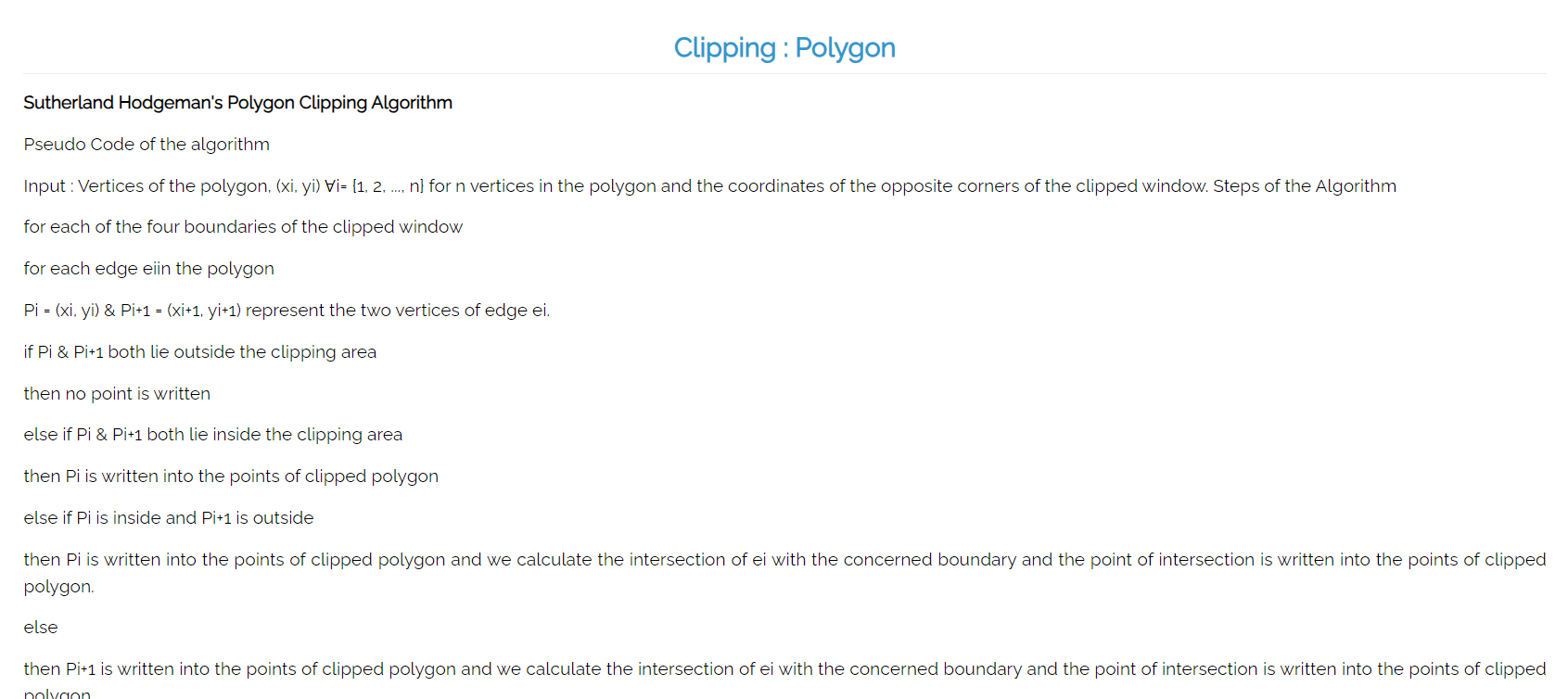
glutMainLoop();

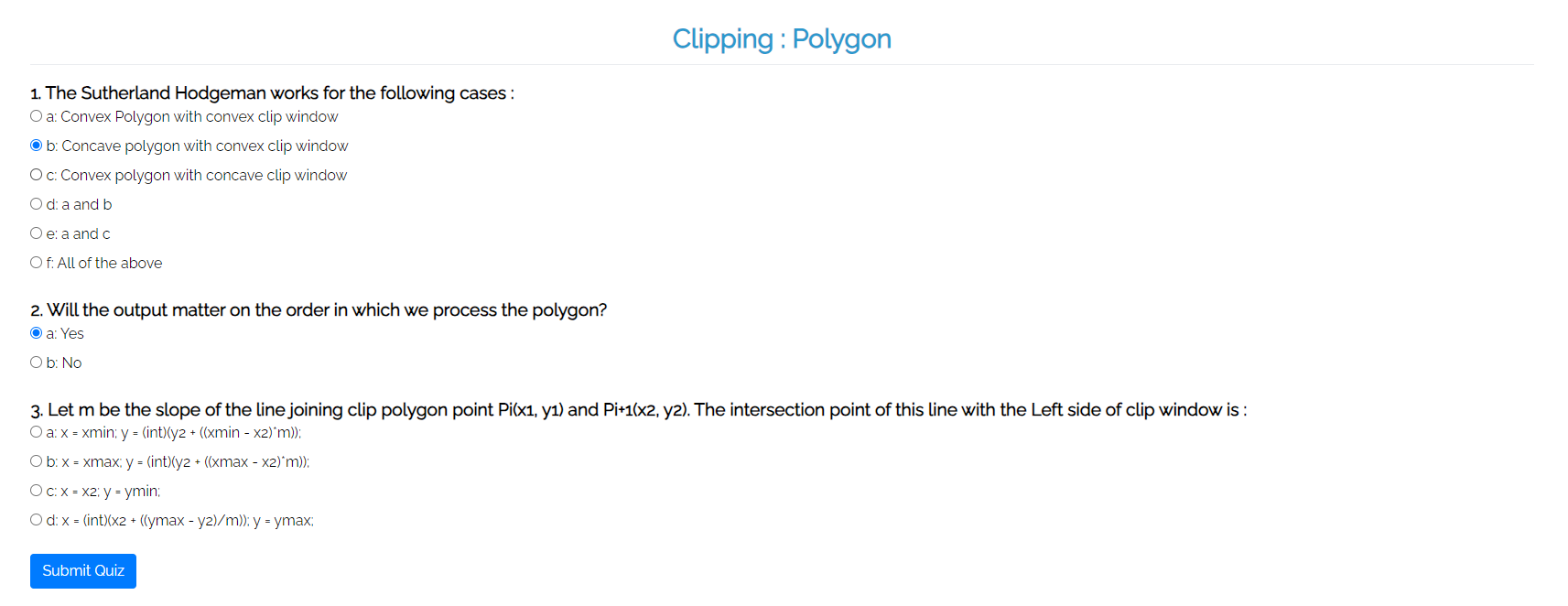
return 0;

}

****

**Screenshots from VLab(if any):**

****

****

**Conclusion and discussion:**

We have learned about polygon clipping algorithms.

**Date: 6/9/24**

**Signature of faculty in-charge**

**Post lab**

Explain Wiler-Atherton algorithm, Implement.

Weiler-Atherton Polygon Clipping Algorithm

The Weiler-Atherton algorithm is an extension of the Sutherland-Hodgman algorithm, particularly useful for clipping complex polygons (polygons with holes or polygons that need to be split into multiple parts). The algorithm is capable of handling both concave and convex polygons by dividing the problem into smaller subproblems. The key idea is to trace the boundaries of both the subject polygon and the clipping polygon, generating a sequence of vertices that define the clipped polygon(s).

Algorithm :

Identify Entry and Exit Points:Traverse the subject polygon and find the intersection points with the clipping polygon's edges.Classify these intersection points as "entry" or "exit" points depending on the direction of traversal relative to the clipping polygon.

Build Output Polygons:Starting from an entry point, trace the polygon by following the edges of the subject polygon.When you encounter an intersection (exit) point, switch to tracing the clipping polygon.Continue this process until you loop back to the starting point, completing one clipped polygon.Repeat the process for all entry points.

Handle Multiple Polygons:This algorithm can generate multiple polygons as a result, depending on the complexity of the subject polygon and the clipping polygon.  
  
Implementation:

#include <GL/glut.h>

#include <vector>

#include <algorithm>

struct Point {

float x, y;

bool isIntersection = false;

bool isEntry = false;

bool operator==(const Point& other) const {

return (x == other.x && y == other.y && isIntersection == other.isIntersection && isEntry == other.isEntry);

}

};

struct Edge {

Point p1, p2;

};

std::vector<Point> subjectPolygon = {{150, 150}, {300, 100}, {350, 300}, {250, 350}, {100, 250}};

std::vector<Point> clipPolygon = {{200, 200}, {300, 200}, {300, 300}, {200, 300}};

// Utility function to check if a point is inside the clipping window

bool isInside(Point p, Edge edge) {

return (edge.p2.x - edge.p1.x) \* (p.y - edge.p1.y) - (edge.p2.y - edge.p1.y) \* (p.x - edge.p1.x) <= 0;

}

// Find the intersection of two lines

Point getIntersection(Edge edge1, Edge edge2) {

float A1 = edge1.p2.y - edge1.p1.y;

float B1 = edge1.p1.x - edge1.p2.x;

float C1 = A1 \* edge1.p1.x + B1 \* edge1.p1.y;

float A2 = edge2.p2.y - edge2.p1.y;

float B2 = edge2.p1.x - edge2.p2.x;

float C2 = A2 \* edge2.p1.x + B2 \* edge2.p1.y;

float det = A1 \* B2 - A2 \* B1;

Point intersection;

intersection.x = (B2 \* C1 - B1 \* C2) / det;

intersection.y = (A1 \* C2 - A2 \* C1) / det;

intersection.isIntersection = true;

return intersection;

}

std::vector<std::vector<Point>> WeilerAtherton(std::vector<Point> subjectPolygon, std::vector<Point> clipPolygon) {

std::vector<std::vector<Point>> outputPolygons;

std::vector<Edge> subjectEdges;

std::vector<Edge> clipEdges;

// Create edges from vertices

for (size\_t i = 0; i < subjectPolygon.size(); i++) {

subjectEdges.push\_back({subjectPolygon[i], subjectPolygon[(i + 1) % subjectPolygon.size()]});

}

for (size\_t i = 0; i < clipPolygon.size(); i++) {

clipEdges.push\_back({clipPolygon[i], clipPolygon[(i + 1) % clipPolygon.size()]});

}

// Step 1: Identify intersections

for (Edge subjectEdge : subjectEdges) {

for (Edge clipEdge : clipEdges) {

if (isInside(subjectEdge.p1, clipEdge) != isInside(subjectEdge.p2, clipEdge)) {

Point intersection = getIntersection(subjectEdge, clipEdge);

if (isInside(subjectEdge.p1, clipEdge)) {

intersection.isEntry = true;

}

subjectPolygon.insert(

std::find(subjectPolygon.begin(), subjectPolygon.end(), subjectEdge.p2),

intersection

);

clipPolygon.insert(

find(clipPolygon.begin(), clipPolygon.end(), clipEdge.p2),

intersection

);

}

}

}

// Step 2: Traverse polygons and build clipped polygons

for (auto& point : subjectPolygon) {

if (point.isIntersection && point.isEntry) {

std::vector<Point> clippedPolygon;

clippedPolygon.push\_back(point);

Point current = point;

bool followSubject = true;

do {

auto& polygon = followSubject ? subjectPolygon : clipPolygon;

auto it = std::find(polygon.begin(), polygon.end(), current);

current = \*(++it == polygon.end() ? polygon.begin() : it);

clippedPolygon.push\_back(current);

if (current.isIntersection) {

followSubject = !followSubject;

}

} while (current.x != point.x || current.y != point.y);

outputPolygons.push\_back(clippedPolygon);

}

}

return outputPolygons;

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

// Draw original subject polygon in red

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

for (const auto& point : subjectPolygon) {

glVertex2f(point.x, point.y);

}

glEnd();

// Perform Weiler-Atherton Clipping

std::vector<std::vector<Point>> clippedPolygons = WeilerAtherton(subjectPolygon, clipPolygon);

// Draw clipped polygons in green

glColor3f(0.0, 1.0, 0.0);

for (const auto& poly : clippedPolygons) {

glBegin(GL\_POLYGON);

for (const auto& point : poly) {

glVertex2f(point.x, point.y);

}

glEnd();

}

// Draw clipping window in blue

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_LOOP);

for (const auto& point : clipPolygon) {

glVertex2f(point.x, point.y);

}

glEnd();

glFlush();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutCreateWindow("Weiler-Atherton Polygon Clipping");

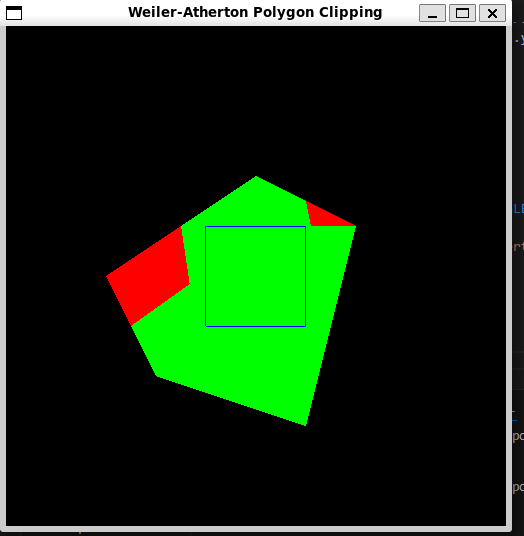
gluOrtho2D(0, 500, 0, 500);

glutDisplayFunc(display);

glutMainLoop();

return 0;

}

****