**Batch: D - 1 Roll No.: 16010122096**

**Experiment No. 07**

|  |
| --- |
| **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:**

CO3: Implement Clipping,3D Geometric Transformations and 3D viewing

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

**Books/ Journals/ Websites referred:**

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

**Psuedocode:**

Function isInside(Point p, Point p1, Point p2) -> Boolean

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

Function getIntersection(Point p1, Point p2, Point clip1, Point clip2) -> Point

A1 = p2.y - p1.y

B1 = p1.x - p2.x

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

A2 = clip2.y - clip1.y

B2 = clip1.x - clip2.x

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

det = A1 \* B2 - A2 \* B1

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

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

Return (x, y)

Function SutherlandHodgman(vector<Point> polygon, vector<Point> clippingWindow) -> vector<Point>

clippedPolygon = polygon

For i from 0 to length(clippingWindow) - 1

     newPolygon = Empty list

     clip1 = clippingWindow[i]

     clip2 = clippingWindow[(i + 1) % length(clippingWindow)]

     For j from 0 to length(clippedPolygon) - 1

         current = clippedPolygon[j]

         previous = clippedPolygon[(j + length(clippedPolygon) - 1) % length(clippedPolygon)]

         If isInside(current, clip1, clip2)

             If not isInside(previous, clip1, clip2)

                 newPolygon.append(getIntersection(previous, current, clip1, clip2))

             newPolygon.append(current)

         Else If isInside(previous, clip1, clip2)

             newPolygon.append(getIntersection(previous, current, clip1, clip2))

     clippedPolygon = newPolygon

Return clippedPolygon

Function display()

Clear color buffer

Set color to green

Begin drawing polygon

For each point in polygon

     Add vertex point

End drawing polygon

clippedPolygon = SutherlandHodgman(polygon, clippingWindow)

Set color to magenta

Begin drawing clipped polygon

For each point in clippedPolygon

     Add vertex point

End drawing clipped polygon

Set color to blue

Begin drawing clipping window as lines

For each point in clippingWindow

     Add vertex point

End drawing clipping window

Flush graphics commands

Function main()

Initialize GLUT

Set display mode to single buffer and RGB color

Set window size to 500x500

Create window with title "Polygon Clipping"

Set orthographic projection from (0,0) to (500,500)

Register display function

Enter GLUT event processing loop

**Implementation details:**

#include <GL/glut.h>

#include <vector>

#include <utility>

using std::vector;

using std::pair;

using Point = pair<float, float>;

vector<Point> polygon = {{50, 50}, {400, 50}, {400, 400}, {50, 400}};

vector<Point> clippingWindow = {{100, 100}, {350, 100}, {350, 350}, {100, 350}};

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

   return (p2.first - p1.first) \* (p.second - p1.second) - (p2.second - p1.second) \* (p.first - p1.first) <= 0;

}

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

   float A1 = p2.second - p1.second;

   float B1 = p1.first - p2.first;

   float C1 = A1 \* p1.first + B1 \* p1.second;

   float A2 = clip2.second - clip1.second;

   float B2 = clip1.first - clip2.first;

   float C2 = A2 \* clip1.first + B2 \* clip1.second;

   float det = A1 \* B2 - A2 \* B1;

   return { (B2 \* C1 - B1 \* C2) / det, (A1 \* C2 - A2 \* C1) / det };

}

vector<Point> SutherlandHodgman(const vector<Point>& polygon, const vector<Point>& clippingWindow) {

   vector<Point> clippedPolygon = polygon;

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

       vector<Point> newPolygon;

       Point clip1 = clippingWindow[i];

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

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

           Point current = clippedPolygon[j];

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

           }

       }

       clippedPolygon = newPolygon;

   }

   return clippedPolygon;

}

void display() {

   glClear(GL\_COLOR\_BUFFER\_BIT);

   glColor3f(0.0, 1.0, 0.0);

   glBegin(GL\_POLYGON);

   for (const auto& point : polygon) {

       glVertex2f(point.first, point.second);

   }

   glEnd();

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

   glColor3f(1.0, 0.0, 1.0);

   glBegin(GL\_POLYGON);

   for (const auto& point : clippedPolygon) {

       glVertex2f(point.first, point.second);

   }

   glEnd();

   glColor3f(0.0, 0.0, 1.0);

   glBegin(GL\_LINE\_LOOP);

   for (const auto& point : clippingWindow) {

       glVertex2f(point.first, point.second);

   }

   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);

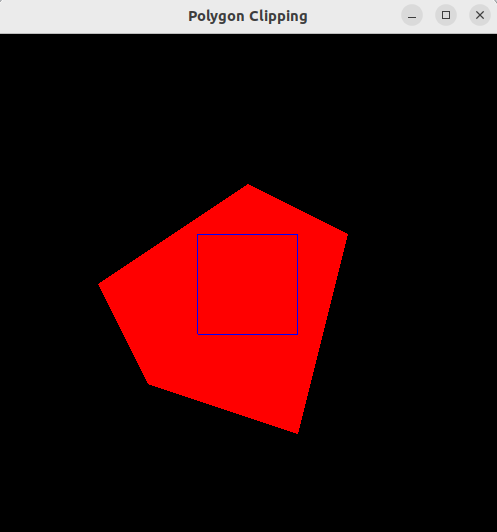
   glutDisplayFunc(display);

   glutMainLoop();

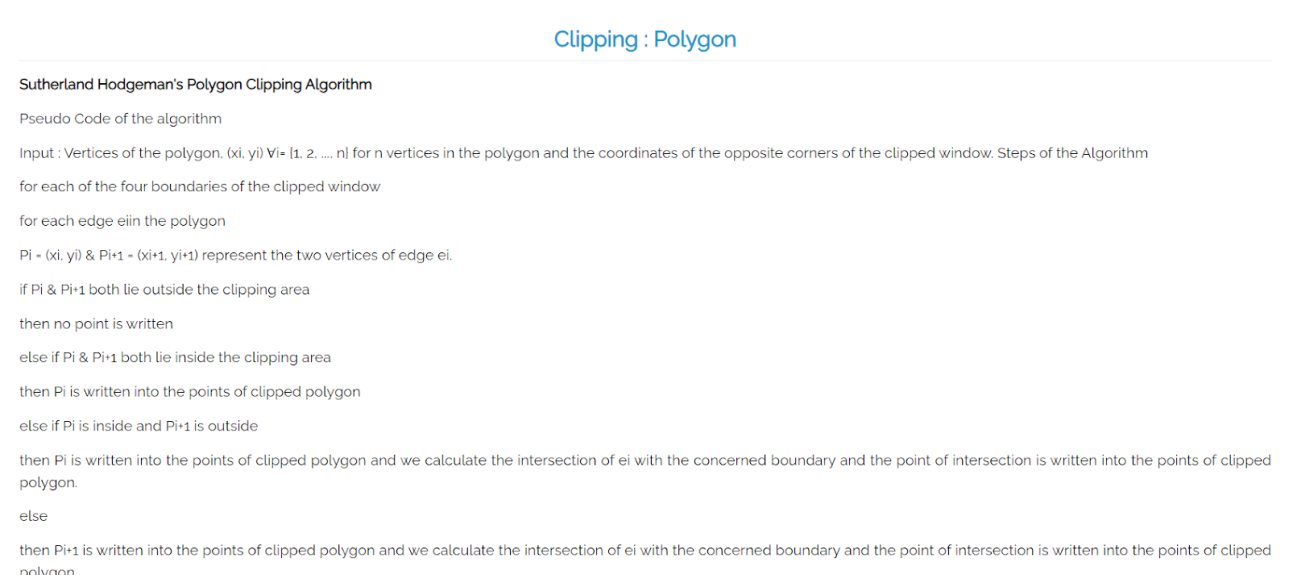
   return 0;

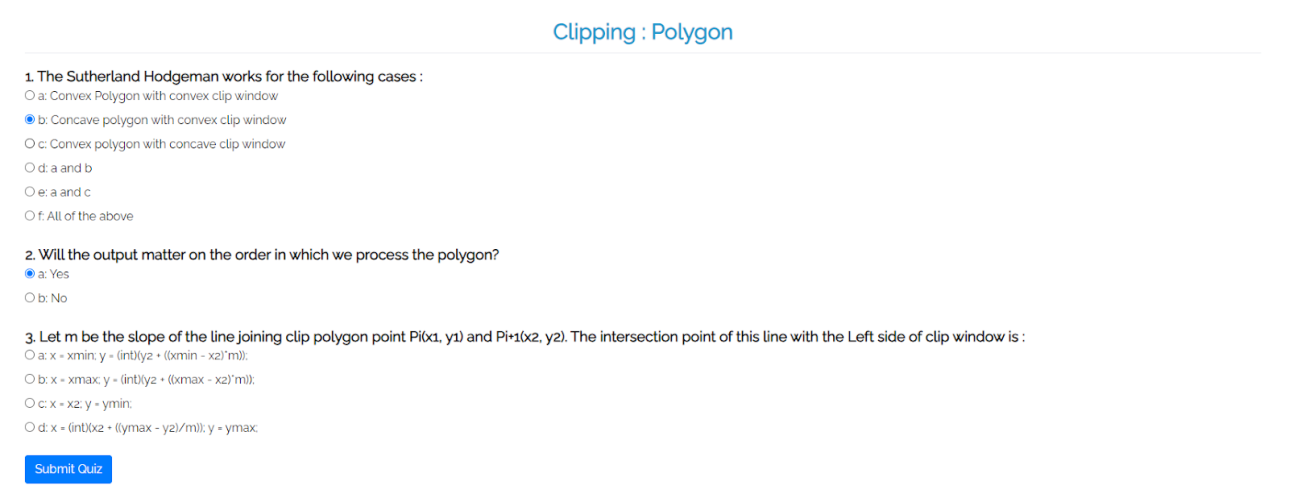
}

Output:



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

****

****

**Conclusion and discussion:**

The Polygon Clipping algorithm demonstrates how portions of a polygon lying outside a defined clipping window can be efficiently removed while preserving the interior sections. This is critical for rendering in computer graphics, ensuring only visible portions are displayed, optimizing performance, and enhancing visualization accuracy in various applications.

**Date: Signature of faculty in-charge**

**Post lab**

Explain Wiler-Atherton algorithm, Implement.

### Weiler-Atherton Algorithm Overview

The Weiler-Atherton algorithm is more versatile and can handle more complex cases than the Sutherland-Hodgman algorithm. It works for clipping polygons against arbitrary clipping windows, including both convex and concave clipping regions.

#### Key Steps of the Weiler-Atherton Algorithm:

1. **Classify the Polygon Edges**: Determine which edges of the polygon are inside or outside the clipping window.
2. **Find Intersection Points**: Identify the intersection points of the polygon edges with the clipping window edges.
3. **Construct New Polygon**: Create a new polygon using the intersection points and the segments of the polygon that lie inside the clipping window.
4. **Handle Multiple Contours**: Manage cases where the polygon might be split into multiple contours due to the clipping operation.

#include <GL/glut.h>

#include <vector>

#include <utility>

using std::vector;

using std::pair;

using Point = pair<float, float>;

vector<Point> polygon = {

   {100, 50}, {200, 20}, {300, 50}, {350, 150}, {200, 250}

};

vector<Point> clippingWindow = {

   {150, 100}, {300, 100}, {300, 200}, {150, 200}

};

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

   return (p2.first - p1.first) \* (p.second - p1.second) - (p2.second - p1.second) \* (p.first - p1.first) <= 0;

}

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

   float A1 = p2.second - p1.second;

   float B1 = p1.first - p2.first;

   float C1 = A1 \* p1.first + B1 \* p1.second;

   float A2 = clip2.second - clip1.second;

   float B2 = clip1.first - clip2.first;

   float C2 = A2 \* clip1.first + B2 \* clip1.second;

   float det = A1 \* B2 - A2 \* B1;

   return { (B2 \* C1 - B1 \* C2) / det, (A1 \* C2 - A2 \* C1) / det };

}

vector<Point> WeilerAtherton(const vector<Point>& polygon, const vector<Point>& clippingWindow) {

   vector<Point> clippedPolygon;

   vector<Point> currentPolygon = polygon;

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

       vector<Point> newPolygon;

       Point clip1 = clippingWindow[i];

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

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

           Point current = currentPolygon[j];

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

           }

       }

       currentPolygon = newPolygon;

   }

   return clippedPolygon;

}

void display() {

   glClear(GL\_COLOR\_BUFFER\_BIT);

   glColor3f(0.0, 1.0, 0.0);  // Green for original polygon

   glBegin(GL\_POLYGON);

   for (const auto& point : polygon) {

       glVertex2f(point.first, point.second);

   }

   glEnd();

   vector<Point> clippedPolygon = WeilerAtherton(polygon, clippingWindow);

   glColor3f(1.0, 0.0, 1.0);  // Magenta for clipped polygon

   glBegin(GL\_POLYGON);

   for (const auto& point : clippedPolygon) {

       glVertex2f(point.first, point.second);

   }

   glEnd();

   glColor3f(0.0, 0.0, 1.0);  // Blue for clipping window

   glBegin(GL\_LINE\_LOOP);

   for (const auto& point : clippingWindow) {

       glVertex2f(point.first, point.second);

   }

   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;

}

**Output:**

