K. J. Somaiya College of Engineering, Mumbai-77
Somaiya Vidyavihar University

Batch: D-2 **Roll No.:** 16010122151

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/

Expected OUTCOME of Experiment:

Implement Clipping, 3D Geometric Transformations and 3D viewing.

Books/ Journals/ Websites referred:

Algorithm/ Pseudocode for each process:

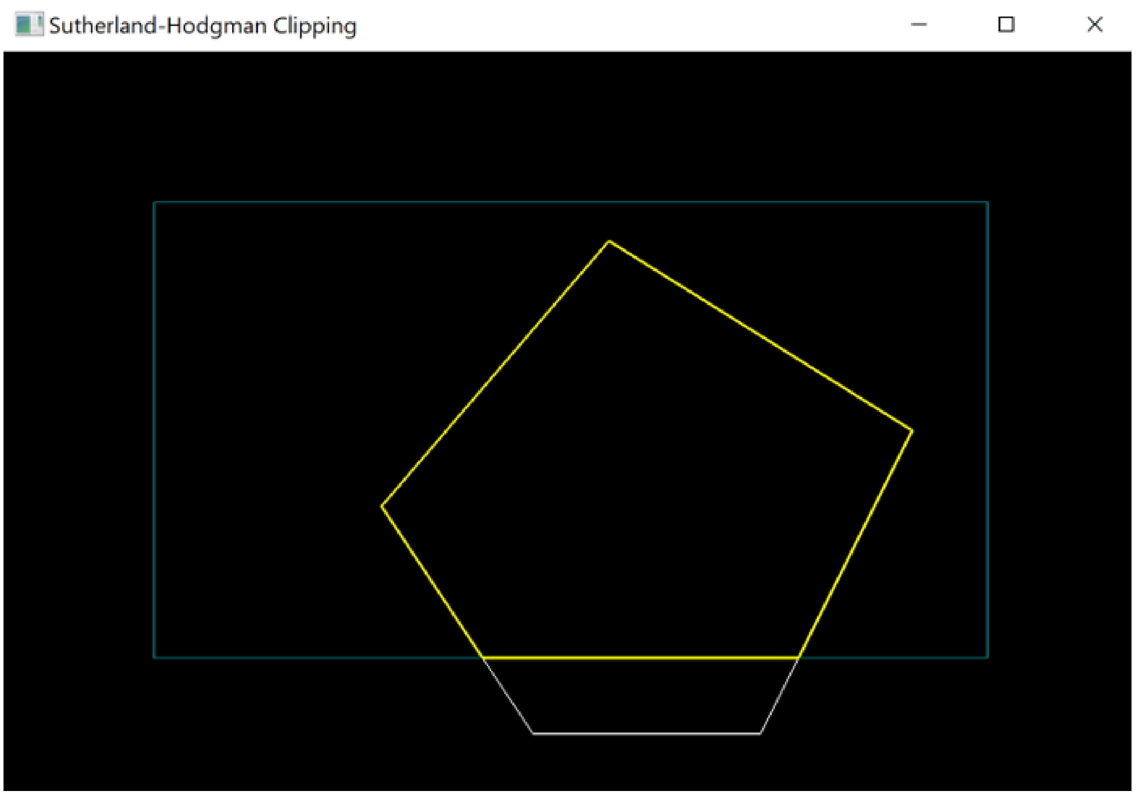
```
for each clipping edge do
    for (i = 0; i < Polygon.length -1; i++)
        Pi = Polygon.vertex[i];
        Pi+1 = Polygon.vertex[i+1];
        if (Pi is inside clipping region)
            if (Pi+1 is inside clipping region)
                clippedPolygon.add(Pi+1)
            else
                clippedPolygon.add(intersectionPoint(Pi, Pi+1, currentEdge)
        else
            if (Pi+1 is inside clipping region)
                clippedPolygon.add(intersectionPoint(Pi, Pi+1, currentEdge)
                clippedPolygon.add(Pi+1)
    end for
    Polygon = clippedPolygon     // keep on working with the new polygon
end for
```

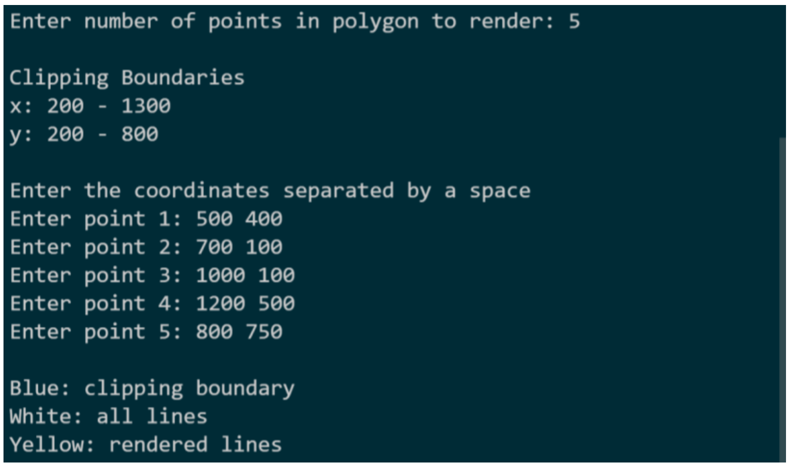
Implementation details:

```
#include <GL/glut.h>
#include <cmath>
#include <vector>
#include <iostream>

int width = 1500;
int height = 1000;
std::vector<std::pair<float, float>> linePointsCopy;
std::vector<std::pair<float, float>> renderPoints;
std::vector<std::pair<float, float>> polyPoints;
std::vector<int> masks = {1, 2, 4, 8};
std::vector<float> clippingBounds = {200, width - 200, 200, height -
200};

void iterate() {
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, width, 0.0, height, 0.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void draw() {
    glColor3f(0.0, 1.0, 1.0);
    glLineWidth(1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(200, 200);
    glVertex2f(200, height - 200);
    glVertex2f(width - 200, height - 200);
    glVertex2f(width - 200, 200);
    glEnd();

    glColor3f(1.0, 1.0, 1.0);
    glLineWidth(2.0);
    glBegin(GL_LINE_LOOP);
    for (const auto& point : linePointsCopy) {
        glVertex2f(point.first, point.second);
    }
    glEnd();

    glColor3f(1.0, 1.0, 0.0);
    glLineWidth(4.0);
    glBegin(GL_LINE_STRIP);
    for (const auto& point : renderPoints) {
        glVertex2f(point.first, point.second);
    }
    glEnd();
}

void showScreen() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    iterate();
    draw();
    glutSwapBuffers();
}

bool isInside(const std::pair<float, float>& point) {
    if (point.first < clippingBounds[0] || point.first >
clippingBounds[1] ||
        point.second < clippingBounds[2] || point.second >
clippingBounds[3]) {
        return false;
    }
    return true;
}

float cartesianDistance(const std::pair<float, float>& point1, const
std::pair<float, float>& point2) {
    float x1 = point1.first;
    float y1 = point1.second;
    float x2 = point2.first;
    float y2 = point2.second;
```

```cpp
    return std::sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
}

std::pair<float, float> boundIntersect(const std::pair<float, float>&
pointInside, const std::pair<float, float>& pointOutside) {
    float x1 = pointInside.first;
    float y1 = pointInside.second;
    float x2 = pointOutside.first;
    float y2 = pointOutside.second;
    float newX = 0, newY = 0;
    float newX1 = 0, newY1 = 0;
    float distFromVert = std::numeric_limits<float>::max();
    float distFromHori = std::numeric_limits<float>::max();

    if (pointOutside.first < clippingBounds[0]) {
        newX = clippingBounds[0];
        newY = (((newX - x1) / (x2 - x1)) * (y2 - y1)) + y1;
        distFromVert = cartesianDistance({newX, newY}, pointInside);
    } else if (pointOutside.first > clippingBounds[1]) {
        newX = clippingBounds[1];
        newY = (((newX - x1) / (x2 - x1)) * (y2 - y1)) + y1;
        distFromVert = cartesianDistance({newX, newY}, pointInside);
    }
    if (pointOutside.second < clippingBounds[2]) {
        newY1 = clippingBounds[2];
        newX1 = (((newY1 - y1) / (y2 - y1)) * (x2 - x1)) + x1;
        distFromHori = cartesianDistance({newX1, newY1}, pointInside);
    } else if (pointOutside.second > clippingBounds[3]) {
        newY1 = clippingBounds[3];
        newX1 = (((newY1 - y1) / (y2 - y1)) * (x2 - x1)) + x1;
        distFromHori = cartesianDistance({newX1, newY1}, pointInside);
    }

    if (distFromHori > distFromVert) {
        return {newX, newY};
    }
    return {newX1, newY1};
}

int main(int argc, char** argv) {
    int n;
    std::cout << "Enter number of points in polygon to render: ";
    std::cin >> n;

    std::cout << "\nClipping Boundaries\nx: 200 - 1300\ny: 200 - 800\n";
    std::cout << "\nEnter the coordinates separated by a space\n";
    for (int i = 0; i < n; ++i) {
        float tempX, tempY;
        std::cout << "Enter point " << i + 1 << ": ";
        std::cin >> tempX >> tempY;
        polyPoints.push_back({tempX, tempY});
    }

    linePointsCopy = polyPoints;

    std::cout << "\nBlue: clipping boundary\nWhite: all lines\nYellow: 
rendered lines\n";

    bool prevPoint = isInside(polyPoints.back());
    if (prevPoint) {
        renderPoints.push_back(polyPoints.back());
    }

    for (size_t i = 0; i < polyPoints.size(); ++i) {
        bool currentPoint = isInside(polyPoints[i]);
        if (currentPoint && prevPoint) {
            renderPoints.push_back(polyPoints[i]);
        } else if (currentPoint && !prevPoint) {
            auto pointIntersect = boundIntersect(polyPoints[i], 
polyPoints[i - 1]);
            renderPoints.push_back(pointIntersect);
            renderPoints.push_back(polyPoints[i]);
        } else if (!currentPoint && prevPoint) {
            auto pointIntersect = boundIntersect(polyPoints[i - 1], 
polyPoints[i]);
            renderPoints.push_back(pointIntersect);
        }
        prevPoint = currentPoint;
    }

    std::cout << renderPoints.size() << " points rendered\n";

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Sutherland-Hodgman Clipping");
    glutDisplayFunc(showScreen);
    glutIdleFunc(showScreen);
    glutMainLoop();

    return 0;
}
```

Output(s) (Screen Shot):





```
Enter number of points in polygon to render: 5

Clipping Boundaries
x: 200 - 1300
y: 200 - 800

Enter the coordinates separated by a space
Enter point 1: 500 400
Enter point 2: 700 100
Enter point 3: 1000 100
Enter point 4: 1200 500
Enter point 5: 800 750

Blue: clipping boundary
White: all lines
Yellow: rendered lines
```

Screenshots from VLab(if any):

Conclusion and discussion:

This experiment has helped to understand the Sutherland-Hodgman Clipping algorithm for polygons.

This algorithm is used to decide what points need to be rendered for a polygon which may or may not be outside the viewing window of the graphics application. This is done by checking the position of each point and its subsequent point. Based on the positions of the points relative to the clipping boundary, one of four cases following is chosen:

- When both points are inside the clipping region, add the second one to the result polygon.
- When both points are outside the clipping region, add none of them to the result polygon.
- When the current point is inside the clipping region but the next point lies outside, add the point of intersection of the line made by the polygon points and clipping boundary, to the resulting polygon.
- When the current point is outside the clipping region but the next one lies inside, add the point of intersection of the line made by the polygon points and clipping boundary, and the next point to the resulting polygon.

Date: 18-10-2024

Signature of faculty in-charge

Post lab
Explain Wiler-Atherton algorithm, Implement.

Step 1: First,create a list of intersection points that are in the starting or ending state.
(I1, I2……. In).

Step 2: Now, create twomore lists; one for subject polygon and other for clip polygon.Fill both listswithintersection points and vertices of the polygon.
Order of Subject Polygon list- Write down all the vertices of the polygon in the subject polygon column.
Order of Clip Polygon list – Write down the points of the clipping window.

Step 3: Insert the vertices in both lists in such a way that the intersection point exists between the correct vertices.

Step 4: Now, start from the first vertex of the polygon. Select the first intersection point as an entering point and follow the same process until we reach the exiting point.

Step 5: We can move from clip polygon list to subject polygon list and search for the finishing intersection point. Repeat the process until we find the entering point.

Step 6: Now, the polygon is being clipped. Repeat the same process until each point has been visited once.

Step 7: Stop.