Experiment No-9

***Aim:*** To implement polygon clipping algorithms

# Theory:

***Sutherland-Hodgeman polygon clipping algorithm-***

The Sutherland-Hodgeman polygon clipping algorithm is a computer graphics algorithm used to clip (i.e., cut or trim) a polygon against a clipping window or another polygon. This algorithm is commonly employed in computer graphics for operations like rendering and image processing. It works by finding the intersection points of the polygon edges with the clipping window and then creating a new clipped polygon from these intersection points.

Here's a step-by-step explanation of the Sutherland-Hodgeman polygon clipping algorithm using an example:

# Algorithm-

1. Input Data:

Define a subject polygon (the one to be clipped). Define a clipping window (typically a rectangle).

1. Initialization: Create an empty list to store the vertices of the clipped polygon.
2. Clipping Loop:

For each edge of the clipping window (usually a rectangle), perform the following steps:

* 1. Inside-Outside Test: Check if the first vertex of the edge is inside the clipping window.
  2. If the first vertex is inside and the second vertex is outside the window, calculate the intersection point between the edge and the clipping window.
  3. If the first vertex is inside and the second vertex is inside, add the second vertex to the list of clipped polygon vertices.
  4. If the first vertex is outside and the second vertex is inside, calculate the intersection point and add it to the list of clipped polygon vertices, along with the second vertex.
  5. If both vertices are outside, skip the edge.

1. Finalize: After all edges of the clipping window have been processed, the list of vertices will contain the vertices of the clipped polygon.

# Example-

Let's illustrate this algorithm with a simple example:

Consider a subject polygon with vertices A(3, 2), B(6, 7), C(8, 3), and D(4, 1), and a rectangular clipping window with vertices P(2, 4), Q(2, 8), R(6, 8), and S(6, 4).

1. Initialise an empty list to store the vertices of the clipped polygon.
2. Start with the first edge of the clipping window: P(2, 4) to Q(2, 8).
3. Check the intersection of this edge with the subject polygon:

The intersection point is I(2, 6). Add I to the list of clipped polygon vertices.

1. Move to the next edge: Q(2, 8) to R(6, 8).
2. The intersection point is J(4, 8). Add J to the list of clipped polygon vertices.
3. Continue with the remaining edges of the clipping window.
4. Once all edges have been processed, you'll have a list of vertices that form the clipped polygon: I(2, 6), J(4, 8), and K(6, 6).

These three points form the clipped polygon, which is the result of applying the Sutherland- Hodgeman algorithm to clip the subject polygon against the given rectangular clipping window.

# Weiler-Atherton polygon clipping algorithm-

The Weiler-Atherton polygon clipping algorithm is a more advanced polygon clipping algorithm than the Sutherland-Hodgeman algorithm. It can handle complex cases where the subject polygon intersects the clipping polygon multiple times, creating multiple resulting polygons. The algorithm was developed by Roger N. Weiler and Peter Atherton.

Here's an explanation of the Weiler-Atherton polygon clipping algorithm using an example:

# Algorithm-

* 1. Input Data:

Define a subject polygon (the one to be clipped). Define a clipping polygon.

Ensure that both polygons are closed and oriented in the same direction (either clockwise or counter-clockwise).

* 1. Initialization:

Initialize lists to store intersection points (entry and exit points) and clipped polygons. Define a starting point (seed point) on either polygon. This can be any point of intersection between the two polygons, or a point known to be outside the other polygon.

* 1. Intersection Calculation:

Find the first intersection point between the subject and clipping polygons. This can be done by checking the subject polygon's edges for intersections with the clipping polygon.

* 1. Clipping Loop:

Start at the first intersection point found.

Trace the subject polygon's boundary while adding points to the entry and exit lists. When you reach an intersection point, switch to the other polygon and continue tracing along its boundary, adding points to the entry and exit lists.

Repeat this process until you return to the seed point.

* 1. Clipped Polygon Creation:

From the entry and exit lists, you can create multiple clipped polygons.

Each entry point in the list corresponds to an exit point, and they define the clipped polygon.

Connect the entry points to their corresponding exit points to form the clipped polygon.

# Example-

Consider a subject polygon with vertices A(2, 2), B(6, 8), C(8, 4), D(4, 1), and a clipping polygon with vertices P(1, 6), Q(1, 9), R(7, 9), S(7, 6). Both polygons are oriented counter- clockwise.

1. Initialize lists for entry and exit points and clipped polygons.
2. Find the first intersection point, which could be A(2, 6).
3. Start tracing the boundary of the subject polygon. Suppose we encounter entry points E1(2, 6) and E2(4, 8), and exit points X1(6, 8) and X2(4, 2).
4. Now switch to the clipping polygon. Continue tracing and find entry points E3(7, 6) and E4(7, 9), and exit points X3(1, 9) and X4(1, 6).
5. Connect entry points to their corresponding exit points: Clipped polygon 1 is A-B-C-D, and clipped polygon 2 is E1-E2-X1-X2-X3-X4.

The Weiler-Atherton algorithm can handle cases where multiple polygons are formed due to intersections and is more versatile for complex scenarios.

# Pseudocode of Sutherland-Hodgeman algorithm-

Pseudocode for the Sutherland-Hodgeman polygon clipping algorithm: #include <stdio.h>

// Define a point structure typedef struct {

double x, y;

} Point;

// Function to calculate the intersection point between two line segments

Point calculateIntersection(Point S, Point E, Point edgeStart, Point edgeEnd) { Point intersection;

// Calculate intersection point and assign it to 'intersection'

// Implement your intersection calculation logic here return intersection;

}

// Function to perform the Sutherland-Hodgeman polygon clipping

void sutherlandHodgeman(Point subjectPolygon[], int subjectVertices, Point clipPolygon[], int clipVertices) {

Point outputPolygon[MAX\_VERTICES]; int outputVertices = 0;

for (int i = 0; i < clipVertices; i++) { Point edgeStart = clipPolygon[i];

Point edgeEnd = clipPolygon[(i + 1) % clipVertices];

Point S = subjectPolygon[subjectVertices - 1]; // Last point in the subject polygon for (int j = 0; j < subjectVertices; j++) {

Point E = subjectPolygon[j];

// Check if E is inside the clipping edge if (E is inside edgeStart and edgeEnd) {

if (S is outside edgeStart and edgeEnd) {

Point intersection = calculateIntersection(S, E, edgeStart, edgeEnd); outputPolygon[outputVertices++] = intersection;

}

outputPolygon[outputVertices++] = E;

}

// Check if S is inside the clipping edge

else if (S is inside edgeStart and edgeEnd) {

Point intersection = calculateIntersection(S, E, edgeStart, edgeEnd); outputPolygon[outputVertices++] = intersection;

}

S = E; // Update S

}

}

// Output the clipped polygon

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

printf("Vertex %d: (%lf, %lf)\n", i, outputPolygon[i].x, outputPolygon[i].y);

}

}

int main() {

// Define subjectPolygon and clipPolygon here

// Call sutherlandHodgeman function to clip subjectPolygon against clipPolygon return 0;

}

In this C pseudocode, you would need to define your subjectPolygon and clipPolygon by providing their vertices. You should also implement the calculateIntersection function to calculate the intersection between two line segments. The result is stored in outputPolygon. Finally, the clipped polygon vertices are printed in the main function.

# Source Code:

#include <stdio.h> #include <graphics.h> #include <conio.h> #include <stdlib.h> *int* main()

{

*int* gd, gm, n, \*x, i, k = 0;

// window coordinates int wx1=220,wy1=140,wx2=420,wy2=140,wx3=420,wy3=340,wx4=220,wy4=340;

*int* w[] = {220, 140, 420, 140, 420, 340, 220, 340, 220, 140}; //

array for drawing window detectgraph(&gd, &gm);

initgraph(&gd, &gm, "c:\\turboc3\\bgi"); // initializing graphics

printf("Window:-");

setcolor(RED); // red colored window drawpoly(5, w); // window drawn

printf("Enter the no. of vertices of polygon: "); scanf("%d", &n);

x = malloc(n \* 2 + 1);

printf("Enter the coordinates of points:\n"); k = 0;

for (i = 0; i < n \* 2; i += 2) // reading vertices of polygon

{

printf("(x%d,y%d): ", k, k);

scanf("%d,%d", &x[i], &x[i + 1]); k++;

}

x[n \* 2] = x[0]; // assigning the coordinates of first vertex to last additional vertex for drawpoly method.

x[n \* 2 + 1] = x[1];

setcolor(WHITE); drawpoly(n + 1, x);

printf("\nPress a button to clip a polygon.."); getch();

setcolor(RED); drawpoly(5, w);

setfillstyle(SOLID\_FILL, BLACK); floodfill(2, 2, RED);

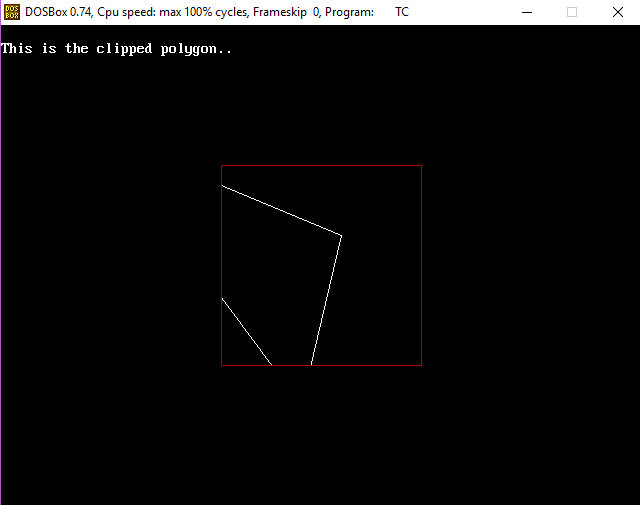
gotoxy(1, 1); // bringing cursor at starting position printf("\nThis is the clipped polygon..");

getch(); cleardevice();

closegraph(); return 0;

}

# Output:



***Conclusion:***

I have understood to implement polygon clipping algorithms using CG.