Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavrepalanchowk

A Lab Report
On
“COMP 342”

Submitted by: Aaditya K.C
Roll No:05
Level:UNG CS(III/I)

Submitted to: Dhiraj Shrestha
Department of Computer Science and Engineering
Submission Date: 01/08/2026

1. **Implement Cohen Sutherland Line Clipping algorithm**

ALGORITHM

Step 1: Define the clipping window

Specify the rectangular window using:

- xmin, ymin
- xmax, ymax

Step 2: Assign region codes (outcodes)

Each endpoint of the line is given a 4-bit region code based on its position relative to the window.

The bits represent:

- Top = 1000
- Bottom = 0100
- Right = 0010
- Left = 0001
  If a point is inside the window, its code is 0000.

Step 3: Check for acceptance or rejection

- If both endpoints have region code 0000 → The line is completely inside (accept it).
- If the logical AND of both region codes is not 0000 → The line is completely outside (reject it).
- Otherwise → The line is partially inside and needs clipping.

Step 4: Find the intersection point

- Select the endpoint that lies outside the window (region code ≠ 0000).

- Find where the line intersects the window boundary indicated by the region code (top, bottom, left, or right).

- Replace the outside point with the intersection point.

Step 5: Repeat the process

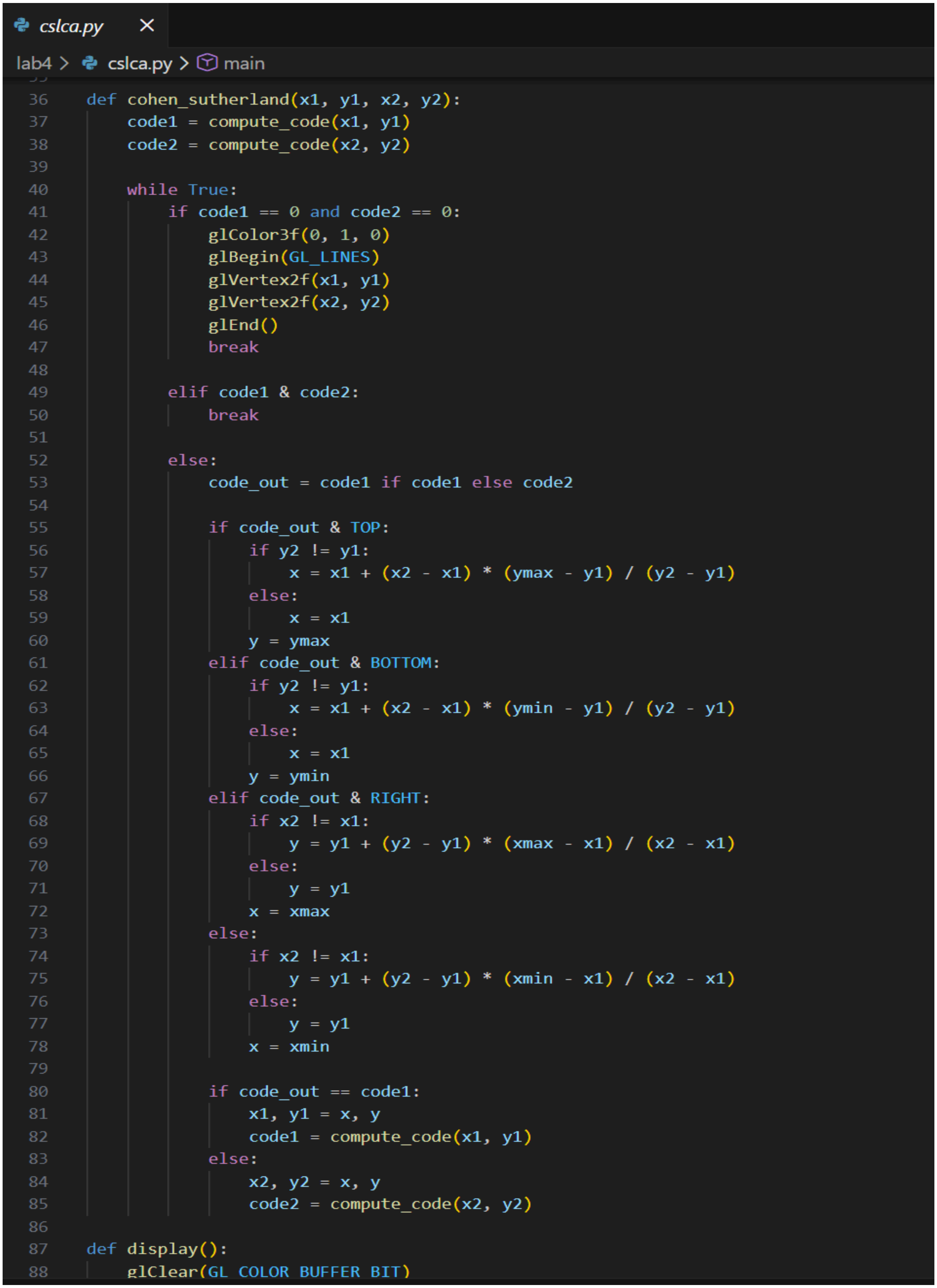
Repeat Steps 2 to 4 until:

- The line is accepted, or

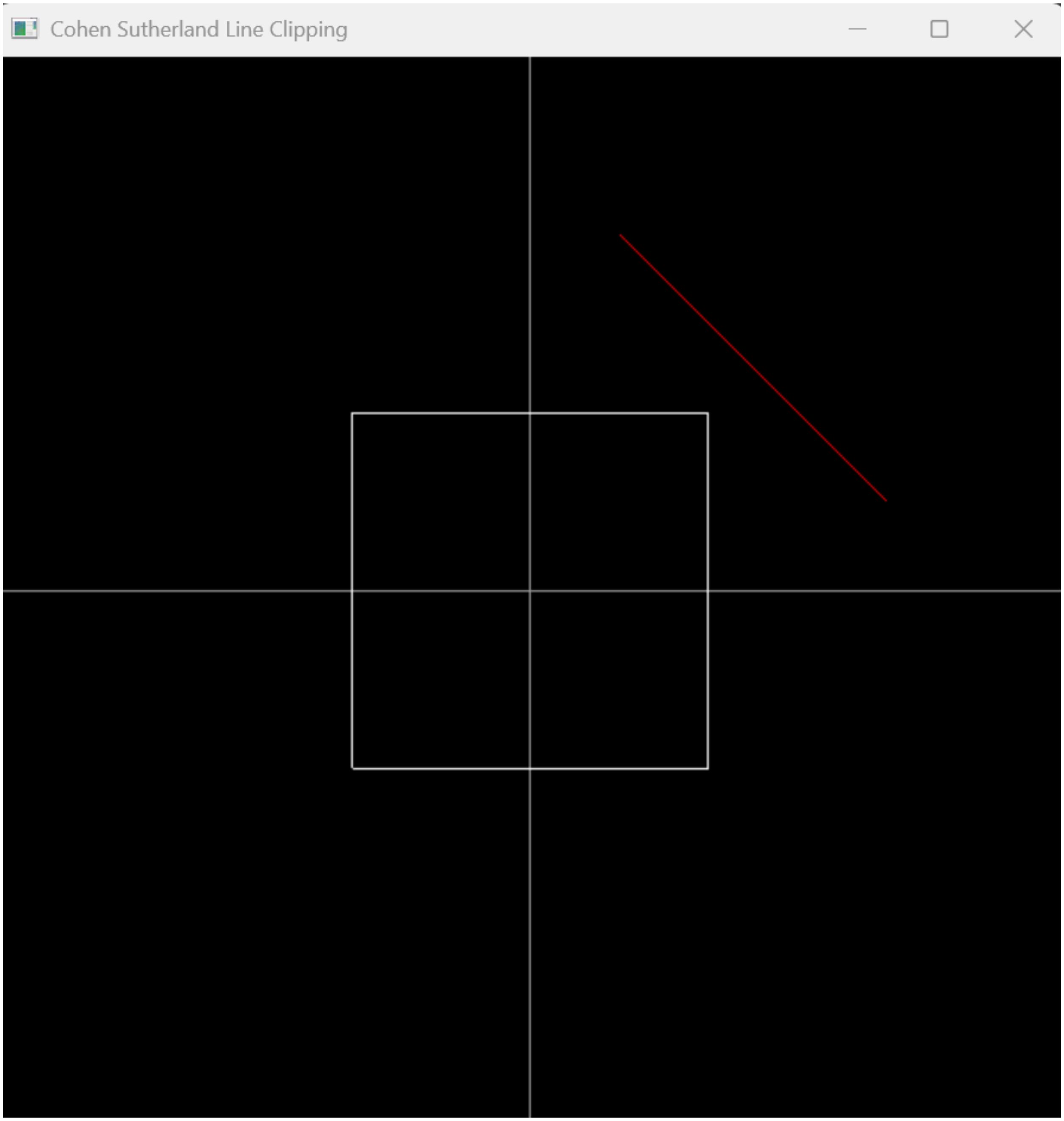
- The line is rejected.

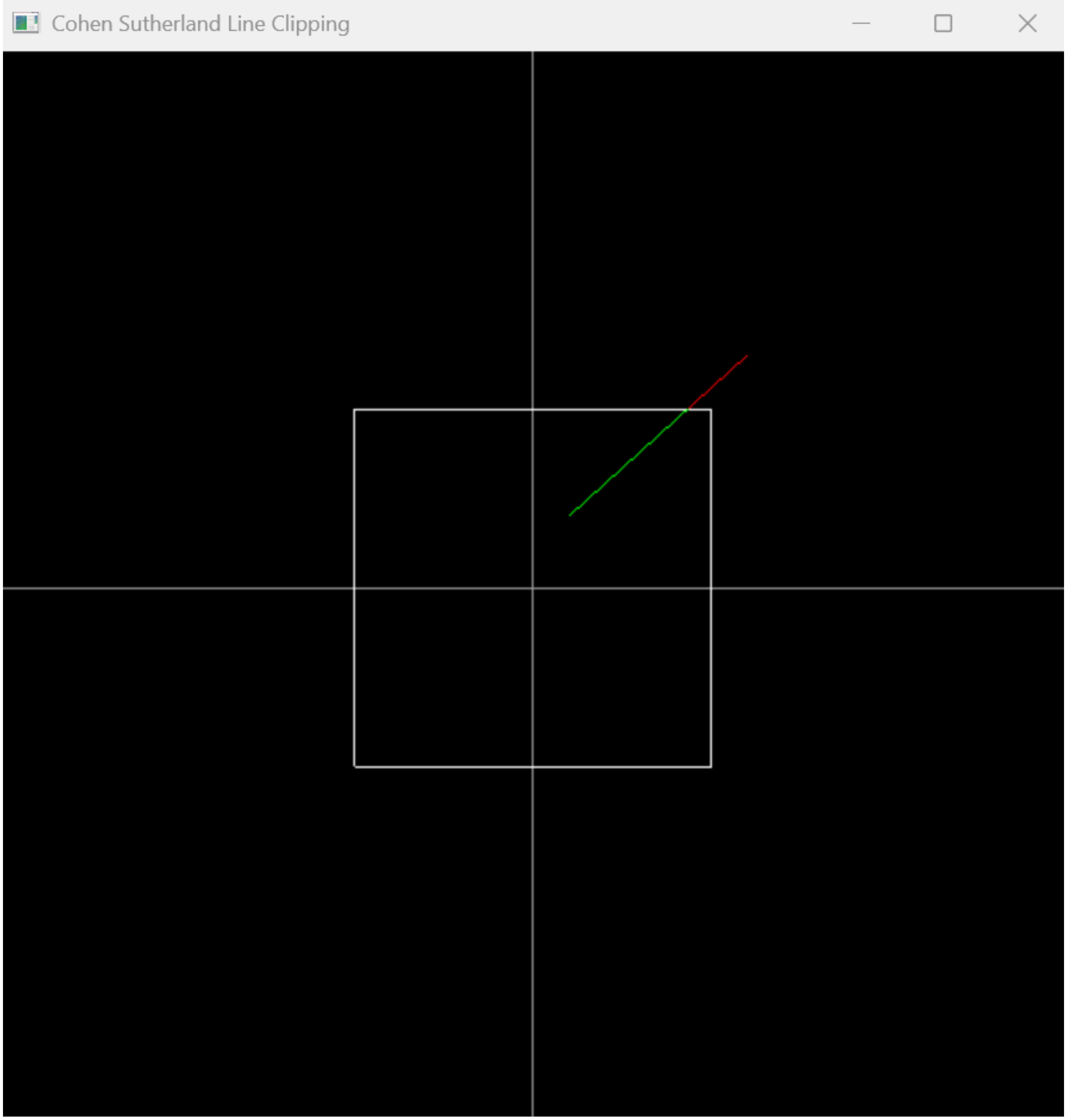
Result

The remaining part of the line inside the window is the clipped line.



```python
def cohen_sutherland(x1, y1, x2, y2):
    code1 = compute_code(x1, y1)
    code2 = compute_code(x2, y2)

    while True:
        if code1 == 0 and code2 == 0:
            glColor3f(0, 1, 0)
            glBegin(GL_LINES)
            glVertex2f(x1, y1)
            glVertex2f(x2, y2)
            glEnd()
            break

        elif code1 & code2:
            break

        else:
            code_out = code1 if code1 else code2

            if code_out & TOP:
                if y2 != y1:
                    x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1)
                else:
                    x = x1
                y = ymax
            elif code_out & BOTTOM:
                if y2 != y1:
                    x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1)
                else:
                    x = x1
                y = ymin
            elif code_out & RIGHT:
                if x2 != x1:
                    y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1)
                else:
                    y = y1
                x = xmax
            else:
                if x2 != x1:
                    y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1)
                else:
                    y = y1
                x = xmin

            if code_out == code1:
                x1, y1 = x, y
                code1 = compute_code(x1, y1)
            else:
                x2, y2 = x, y
                code2 = compute_code(x2, y2)

def display():
    glClear(GL_COLOR_BUFFER_BIT)
```


Cohen Sutherland Line Clipping


Cohen Sutherland Line Clipping

2. Implement Liang Barsky Line Clipping algorithm

ALGORITHM

Step 1: Line equation

A line joining two points (x1,y1)(x1, y1)(x1,y1) and (x2,y2)(x2, y2)(x2,y2) is written as:

- $x = x1 + u(x2 - x1)$
- $y = y1 + u(y2 - y1)$

Here, u varies from 0 to 1.

Step 2: Find direction values

- $dx = x2 - x1$
- $dy = y2 - y1$

Step 3: Calculate p and q values

For a clipping window with limits xmin,xmax,ymin,ymaxxmin, xmax, ymin, ymax, xmin,xmax,ymin,ymax:

- $p1 = -dx, \quad q1 = x1 - xmin$
- $p2 = dx, \quad q2 = xmax - x1$
- $p3 = -dy, \quad q3 = y1 - ymin$
- $p4 = dy, \quad q4 = ymax - y1$

Then calculate:

- $r = q / p$

Step 4: Check the conditions

- If $p = 0$ and $q < 0$ → The line is parallel and completely outside the window.
- If $p = 0$ and $q \geq 0$ → The line is parallel and inside the window.
- If $p < 0$ → The line is entering the window.
- If $p > 0$ → The line is exiting the window.

Step 5: Find u values

- u1 = maximum value of r for entering points (where $p < 0$)
- u2 = minimum value of r for exiting points (where $p > 0$)

If u1 > u2, the line lies completely outside the window.

Step 6: Find the clipped line

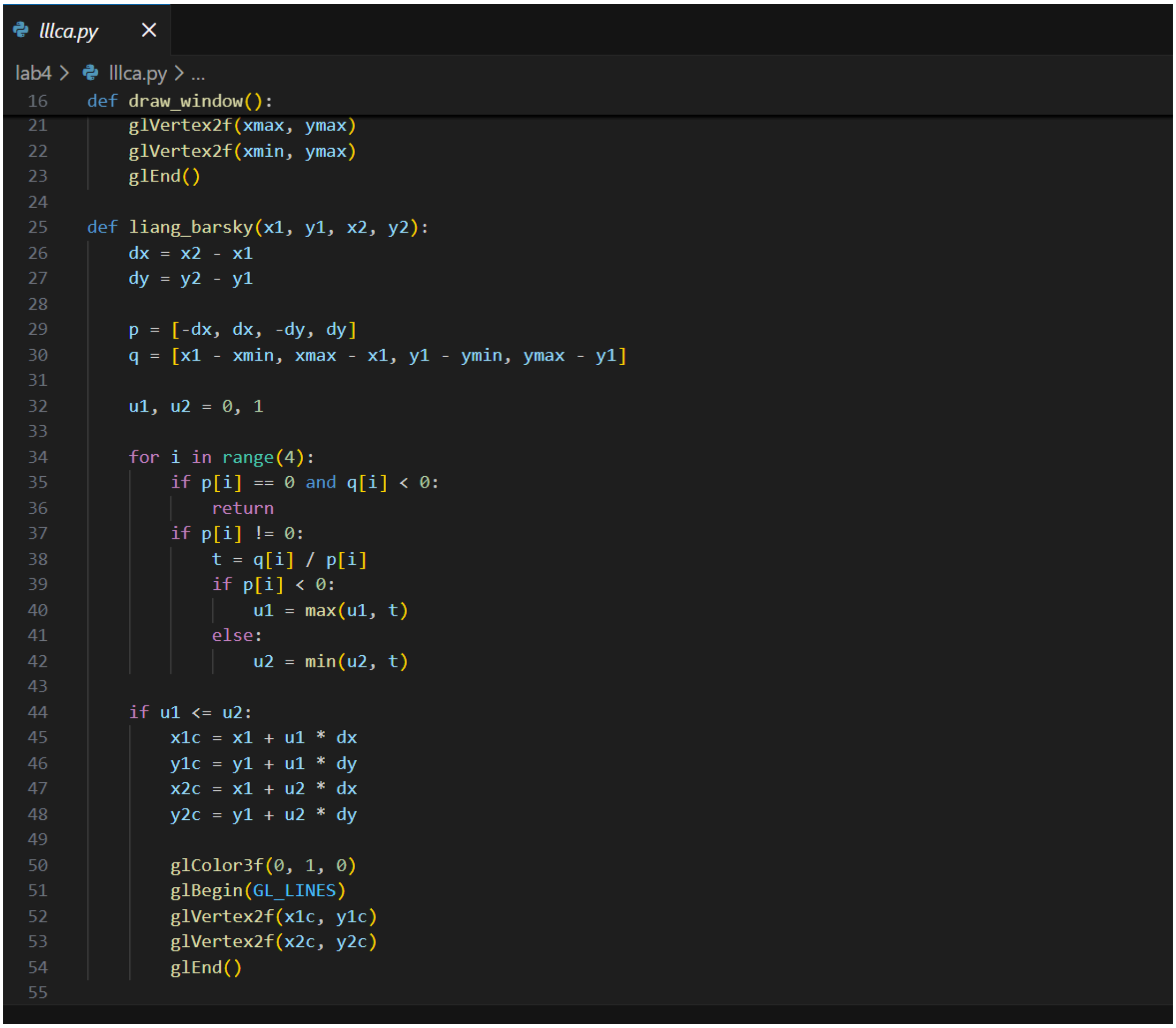
Entering point:

- $x' = x1 + u1 \times dx$
- $y' = y1 + u1 \times dy$

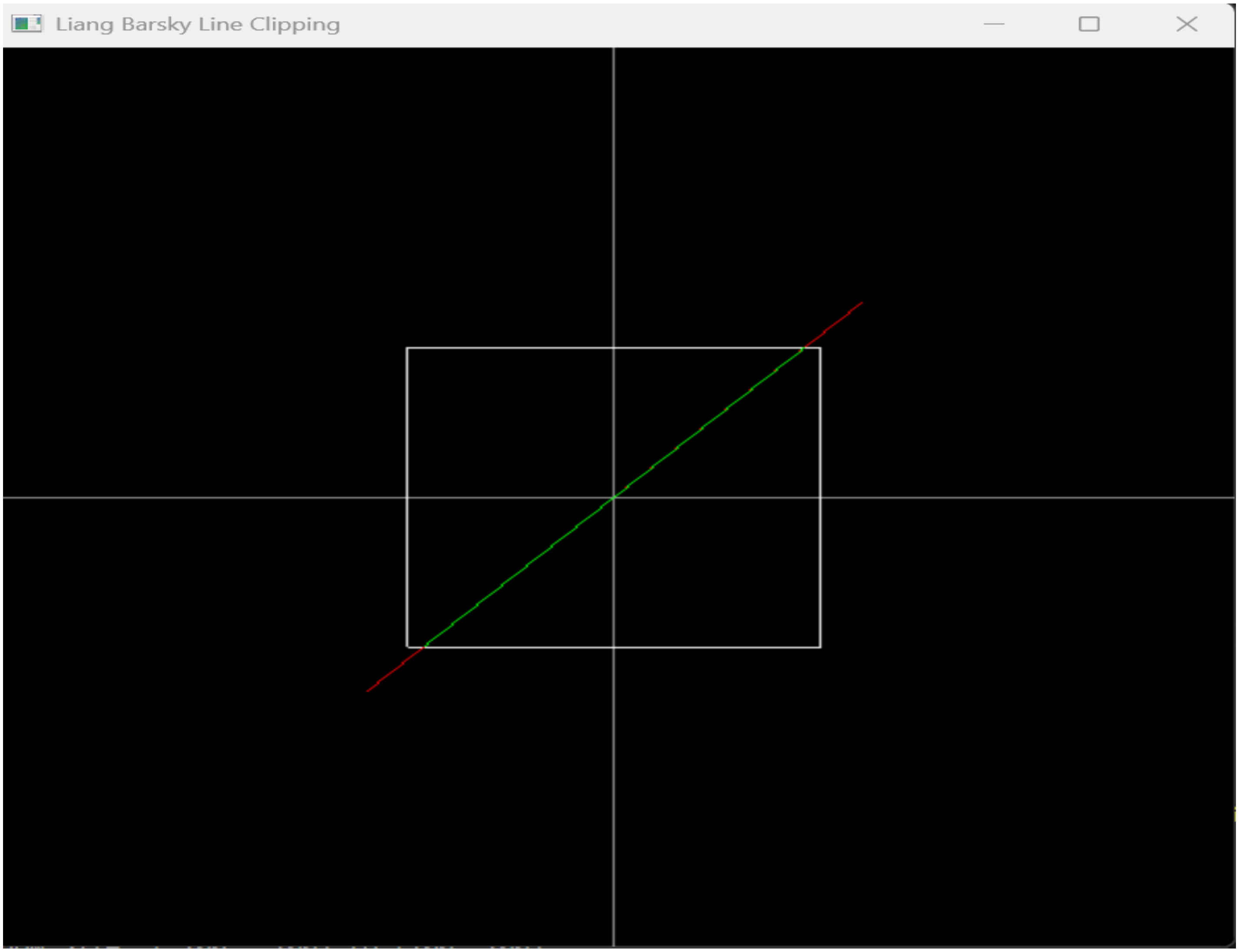
Exiting point:

- $x'' = x1 + u2 \times dx$
- $y'' = y1 + u2 \times dy$

Result   The part of the line between the entering and exiting points is the clipped line inside the window.



```python
def draw_window():
    glVertex2f(xmax, ymax)
    glVertex2f(xmin, ymax)
    glEnd()

def liang_barsky(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1

    p = [-dx, dx, -dy, dy]
    q = [x1 - xmin, xmax - x1, y1 - ymin, ymax - y1]

    u1, u2 = 0, 1

    for i in range(4):
        if p[i] == 0 and q[i] < 0:
            return
        if p[i] != 0:
            t = q[i] / p[i]
            if p[i] < 0:
                u1 = max(u1, t)
            else:
                u2 = min(u2, t)

    if u1 <= u2:
        x1c = x1 + u1 * dx
        y1c = y1 + u1 * dy
        x2c = x1 + u2 * dx
        y2c = y1 + u2 * dy

        glColor3f(0, 1, 0)
        glBegin(GL_LINES)
        glVertex2f(x1c, y1c)
        glVertex2f(x2c, y2c)
        glEnd()

```


Liang Barsky Line Clipping

3. Implement Sutherland Hodgemann polygon clipping algorithm

ALGORITHM

Step 1: Input

- A polygon defined by its vertices (x,y)(x, y)(x,y) in order
- A rectangular clipping window with boundaries:
    - Left
    - Right
    - Bottom
    - Top

Step 2: Clip against one window edge at a time

The polygon is clipped sequentially against the four edges of the window in this order:

1. Left edge
2. Right edge
3. Bottom edge
4. Top edge

The output polygon from one edge becomes the input for the next edge.

Step 3: Process each polygon edge

For each clipping boundary, take two consecutive vertices of the polygon:

- First point (S)
- Second point (E)

Check their position relative to the clipping edge.

Step 4: Apply the following cases

For each pair of points (S → E):

1. Both points inside
    - Add point E to the output list.
2. S inside, E outside
    - Find the intersection point with the clipping edge.
    - Add the intersection point to the output list.
3. S outside, E inside
    - Find the intersection point.
    - Add the intersection point and then point E.
4. Both points outside
    - Do not add anything.

Step 5: Repeat for all edges

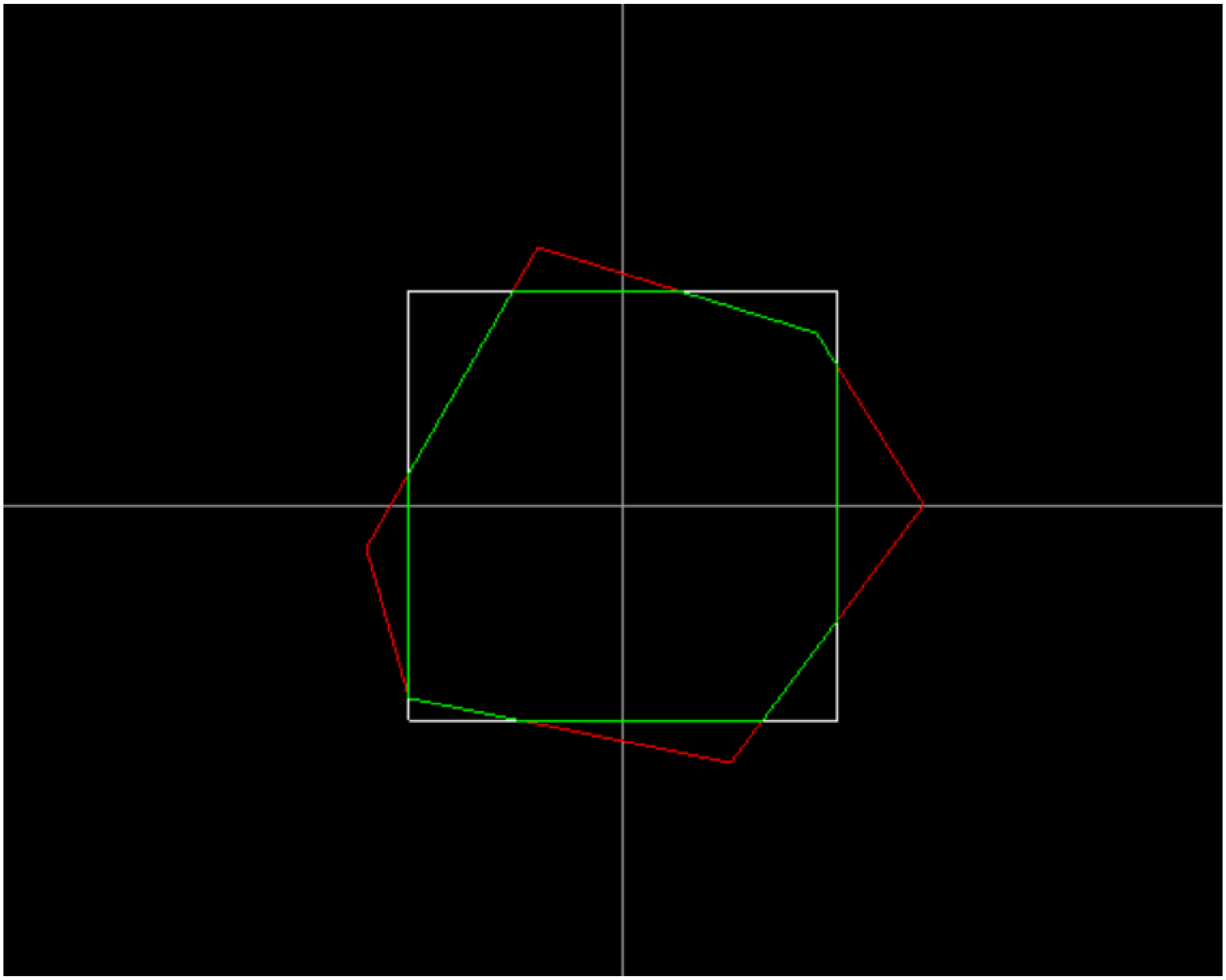
- After clipping against one boundary, use the new polygon.
- Repeat Steps 3 and 4 for the next boundary.
- Continue until all four boundaries are processed.

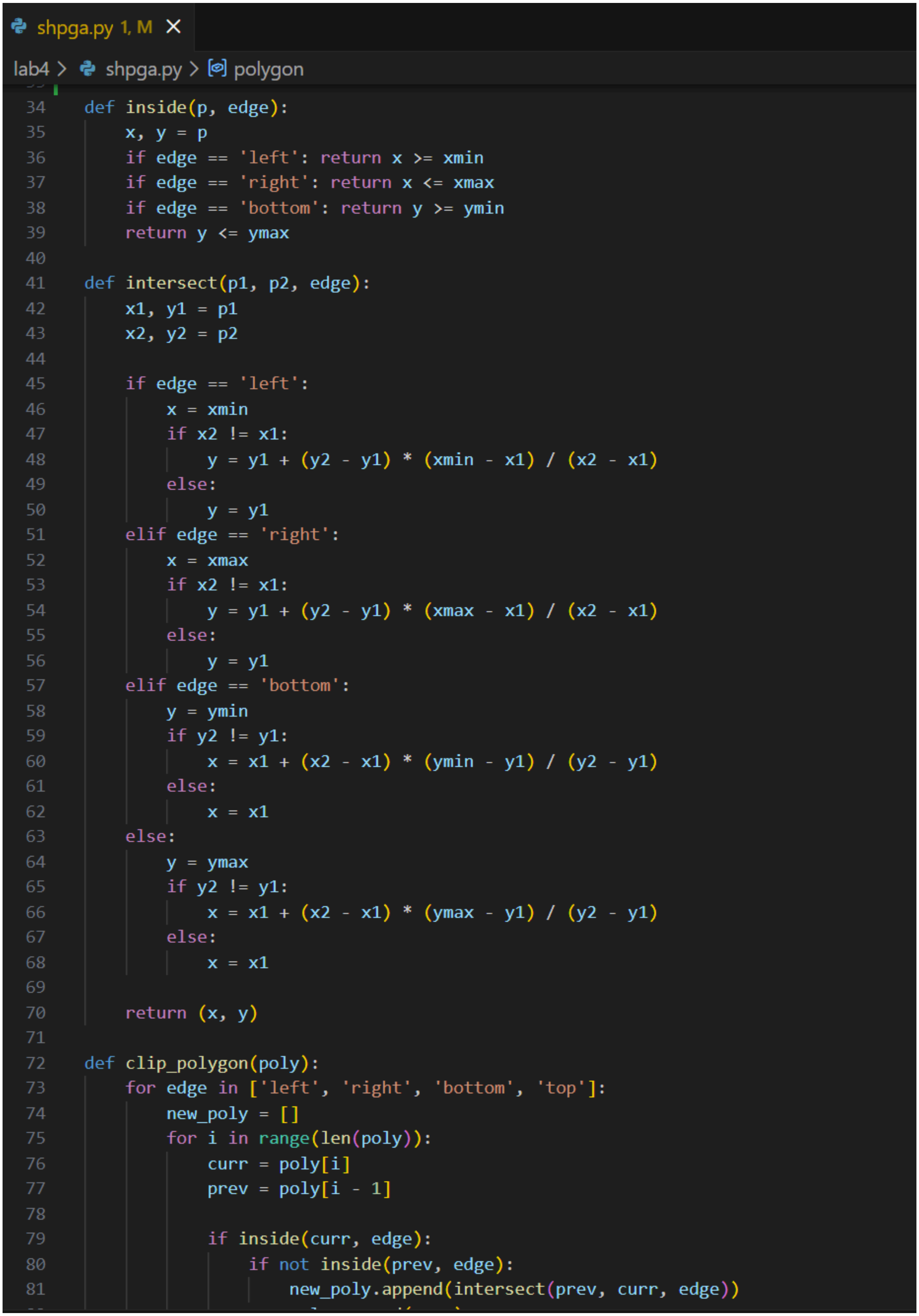
Step 6: Final Output

The remaining vertices form the clipped polygon, which lies completely inside the clipping window.

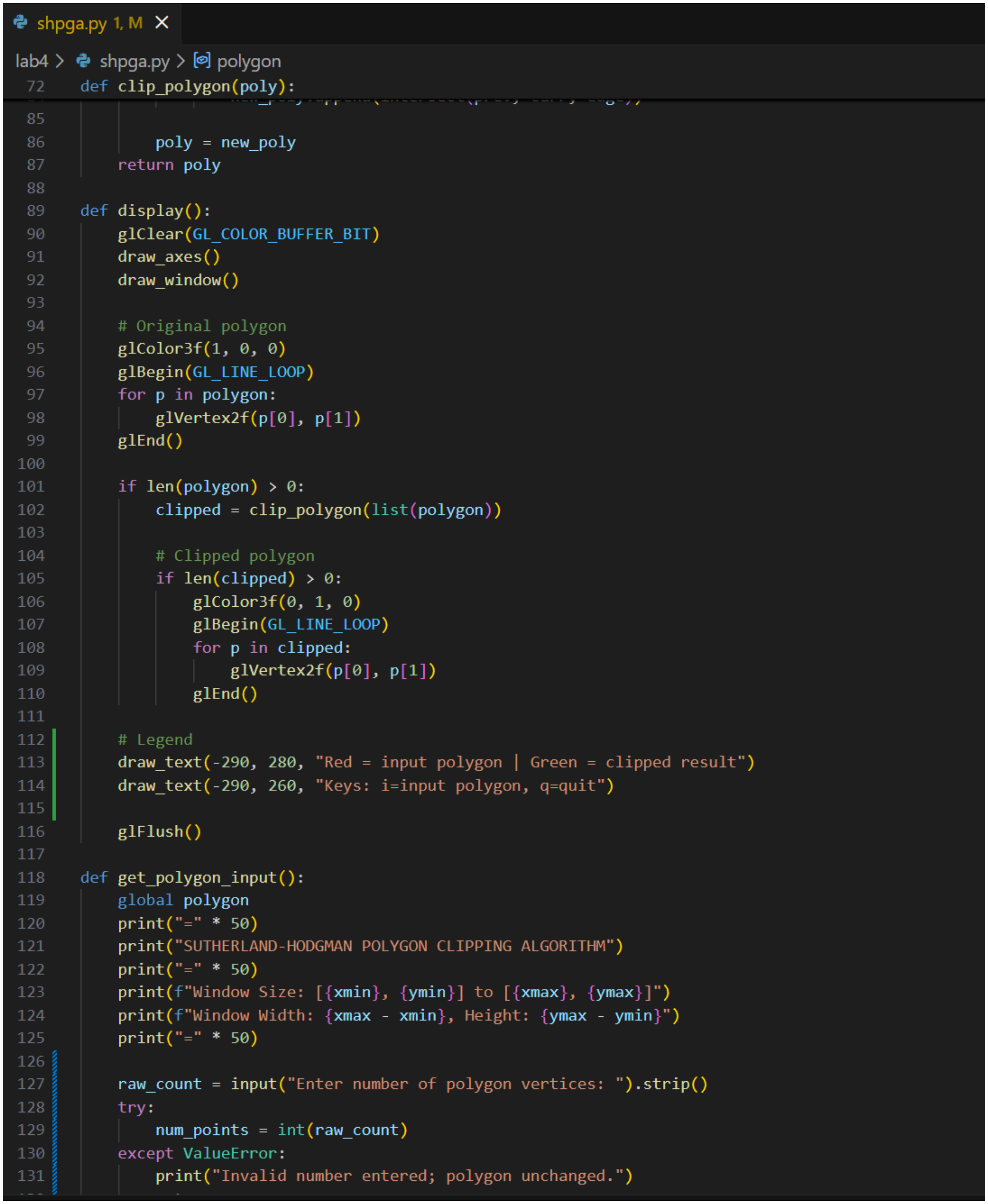
Result

The final polygon obtained after clipping against all window edges is the visible polygon.





```python
def inside(p, edge):
    x, y = p
    if edge == 'left': return x >= xmin
    if edge == 'right': return x <= xmax
    if edge == 'bottom': return y >= ymin
    return y <= ymax

def intersect(p1, p2, edge):
    x1, y1 = p1
    x2, y2 = p2

    if edge == 'left':
        x = xmin
        if x2 != x1:
            y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1)
        else:
            y = y1
    elif edge == 'right':
        x = xmax
        if x2 != x1:
            y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1)
        else:
            y = y1
    elif edge == 'bottom':
        y = ymin
        if y2 != y1:
            x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1)
        else:
            x = x1
    else:
        y = ymax
        if y2 != y1:
            x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1)
        else:
            x = x1

    return (x, y)

def clip_polygon(poly):
    for edge in ['left', 'right', 'bottom', 'top']:
        new_poly = []
        for i in range(len(poly)):
            curr = poly[i]
            prev = poly[i - 1]

            if inside(curr, edge):
                if not inside(prev, edge):
                    new_poly.append(intersect(prev, curr, edge))
```



```
def clip_polygon(poly):

            poly = new_poly
    return poly

def display():
    glClear(GL_COLOR_BUFFER_BIT)
    draw_axes()
    draw_window()

    # Original polygon
    glColor3f(1, 0, 0)
    glBegin(GL_LINE_LOOP)
    for p in polygon:
        glVertex2f(p[0], p[1])
    glEnd()

    if len(polygon) > 0:
        clipped = clip_polygon(list(polygon))

        # Clipped polygon
        if len(clipped) > 0:
            glColor3f(0, 1, 0)
            glBegin(GL_LINE_LOOP)
            for p in clipped:
                glVertex2f(p[0], p[1])
            glEnd()

    # Legend
    draw_text(-290, 280, "Red = input polygon | Green = clipped result")
    draw_text(-290, 260, "Keys: i=input polygon, q=quit")

    glFlush()

def get_polygon_input():
    global polygon
    print("=" * 50)
    print("SUTHERLAND-HODGMAN POLYGON CLIPPING ALGORITHM")
    print("=" * 50)
    print(f"Window Size: [{xmin}, {ymin}] to [{xmax}, {ymax}]")
    print(f"Window Width: {xmax - xmin}, Height: {ymax - ymin}")
    print("=" * 50)

    raw_count = input("Enter number of polygon vertices: ").strip()
    try:
        num_points = int(raw_count)
    except ValueError:
        print("Invalid number entered; polygon unchanged.")
```