Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

LAB-5

[Course : COMP 342]

Submitted by

Sudip Bhattarai

Roll no:11

Submitted to

Mr.Dhiraj Shrestha

Department of Computer Science and Engineering

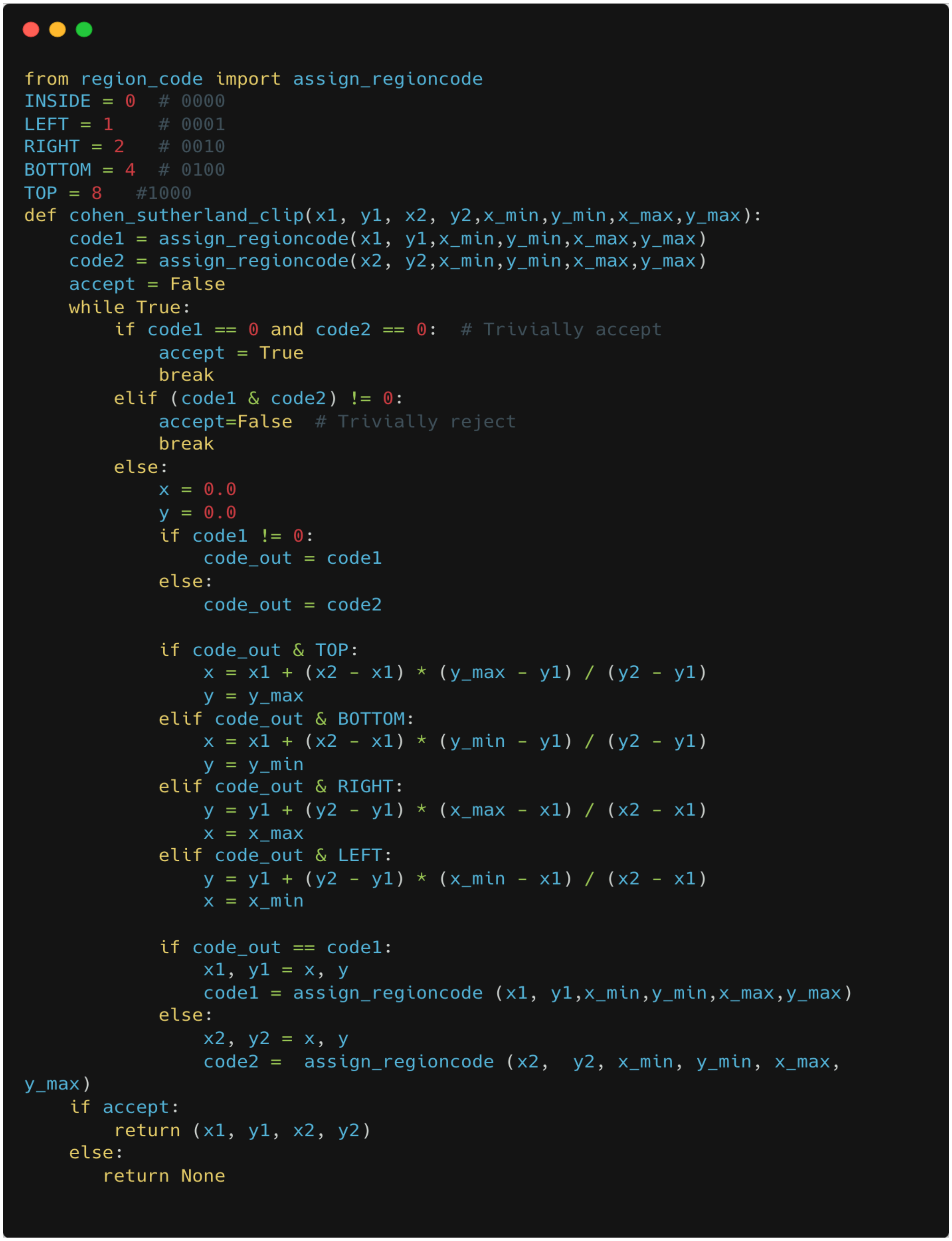
JUN 5 , 2024,

Implement line Clipping Algorithm

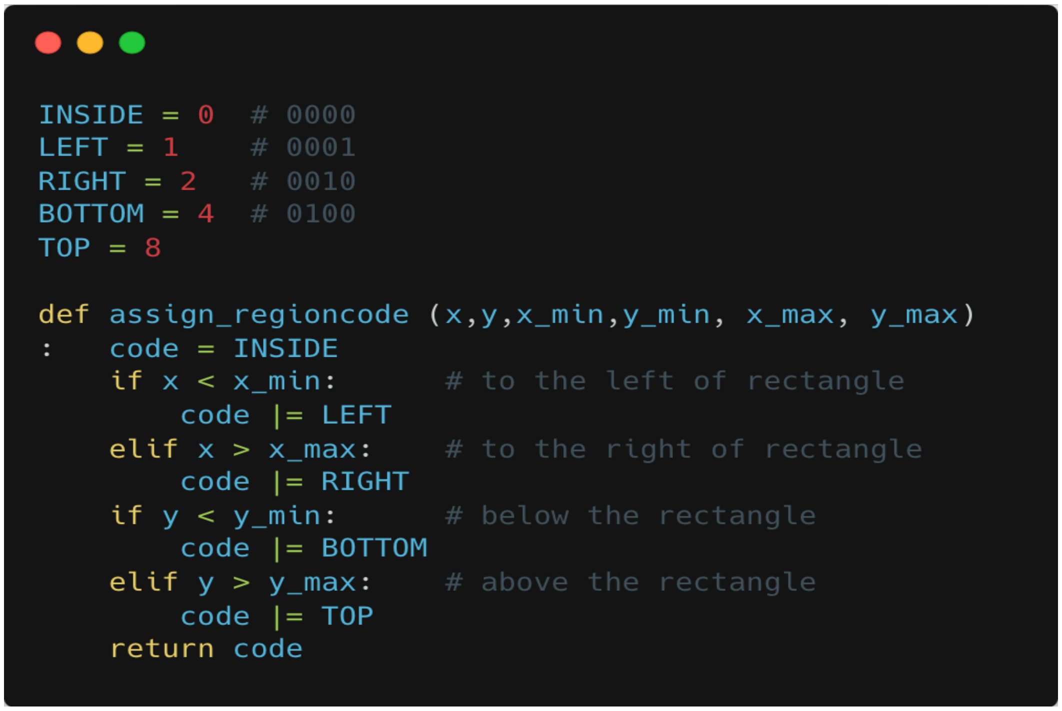
Algorithm

```
1. Assign a region code for each endpoints.
2. If both endpoints have a region code 0000, trivially
accept these line.
3. Else, perform the logical AND operation for both region
    3.1 If the result is NOT 0000: trivially reject the line.
    3.2 else (i.e. result = 0000, need clipping)
        3.2.1. Choose an endpoint of the line that is outside the
        3.2.2. Find the intersection point at the window boundary
            (based on region code).
        3.2.3. Replace the endpoint with the intersection point and
            update the region code.
        3.2.4. Repeat step 2 until we find a clipped line that is either trivially
            accepted or trivially rejected.
4. Repeat step 1 for the other lines.
```

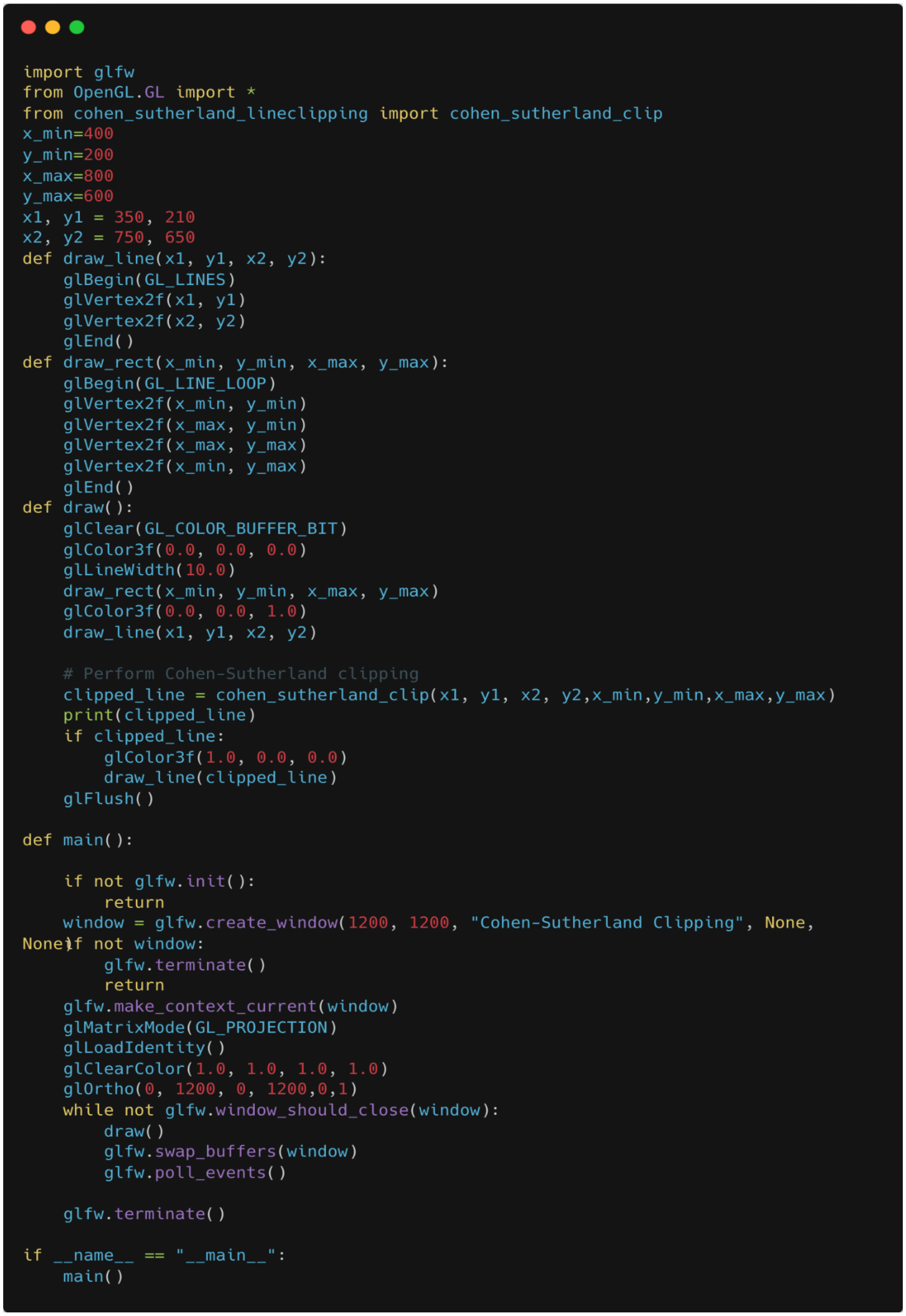
Code Implementation:



```python
from region_code import assign_regioncode
INSIDE = 0  # 0000
LEFT = 1    # 0001
RIGHT = 2   # 0010
BOTTOM = 4  # 0100
TOP = 8   #1000
def cohen_sutherland_clip(x1, y1, x2, y2,x_min,y_min,x_max,y_max):
    code1 = assign_regioncode(x1, y1,x_min,y_min,x_max,y_max)
    code2 = assign_regioncode(x2, y2,x_min,y_min,x_max,y_max)
    accept = False
    while True:
        if code1 == 0 and code2 == 0:  # Trivially accept
            accept = True
            break
        elif (code1 & code2) != 0:
            accept=False  # Trivially reject
            break
        else:
            x = 0.0
            y = 0.0
            if code1 != 0:
                code_out = code1
            else:
                code_out = code2

            if code_out & TOP:
                x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1)
                y = y_max
            elif code_out & BOTTOM:
                x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1)
                y = y_min
            elif code_out & RIGHT:
                y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1)
                x = x_max
            elif code_out & LEFT:
                y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1)
                x = x_min

            if code_out == code1:
                x1, y1 = x, y
                code1 = assign_regioncode (x1, y1,x_min,y_min,x_max,y_max)
            else:
                x2, y2 = x, y
                code2 =  assign_regioncode (x2,  y2, x_min, y_min, x_max,
y_max)
    if accept:
        return (x1, y1, x2, y2)
    else:
       return None
```


```
INSIDE = 0  # 0000
LEFT = 1    # 0001
RIGHT = 2   # 0010
BOTTOM = 4  # 0100
TOP = 8

def assign_regioncode (x,y,x_min,y_min, x_max, y_max)
:   code = INSIDE
    if x < x_min:       # to the left of rectangle
        code |= LEFT
    elif x > x_max:     # to the right of rectangle
        code |= RIGHT
    if y < y_min:       # below the rectangle
        code |= BOTTOM
    elif y > y_max:     # above the rectangle
        code |= TOP
    return code
```

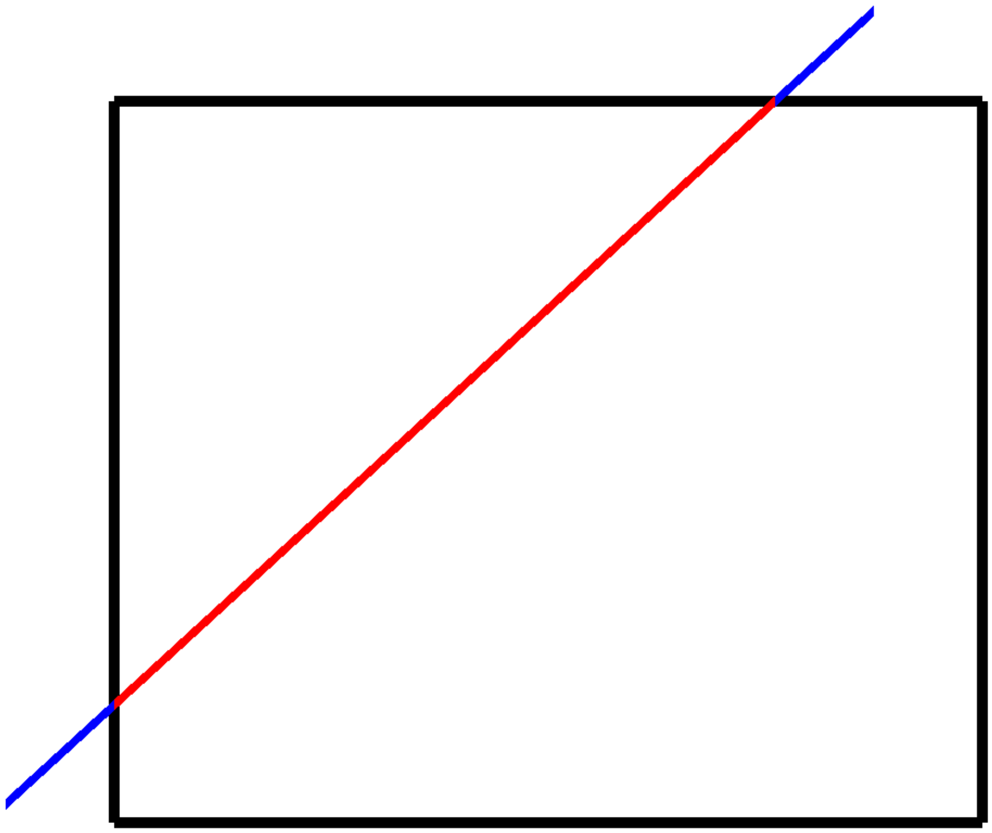
Displaying in OpenGL:



```
import glfw
from OpenGL.GL import *
from cohen_sutherland_lineclipping import cohen_sutherland_clip
x_min=400
y_min=200
x_max=800
y_max=600
x1, y1 = 350, 210
x2, y2 = 750, 650
def draw_line(x1, y1, x2, y2):
    glBegin(GL_LINES)
    glVertex2f(x1, y1)
    glVertex2f(x2, y2)
    glEnd()
def draw_rect(x_min, y_min, x_max, y_max):
    glBegin(GL_LINE_LOOP)
    glVertex2f(x_min, y_min)
    glVertex2f(x_max, y_min)
    glVertex2f(x_max, y_max)
    glVertex2f(x_min, y_max)
    glEnd()
def draw():
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(0.0, 0.0, 0.0)
    glLineWidth(10.0)
    draw_rect(x_min, y_min, x_max, y_max)
    glColor3f(0.0, 0.0, 1.0)
    draw_line(x1, y1, x2, y2)

    # Perform Cohen-Sutherland clipping
    clipped_line = cohen_sutherland_clip(x1, y1, x2, y2,x_min,y_min,x_max,y_max)
    print(clipped_line)
    if clipped_line:
        glColor3f(1.0, 0.0, 0.0)
        draw_line(clipped_line)
    glFlush()

def main():

    if not glfw.init():
        return
    window = glfw.create_window(1200, 1200, "Cohen-Sutherland Clipping", None,
None)if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glClearColor(1.0, 1.0, 1.0, 1.0)
    glOrtho(0, 1200, 0, 1200,0,1)
    while not glfw.window_should_close(window):
        draw()
        glfw.swap_buffers(window)
        glfw.poll_events()

    glfw.terminate()

if __name__ == "__main__":
    main()
```

Output:

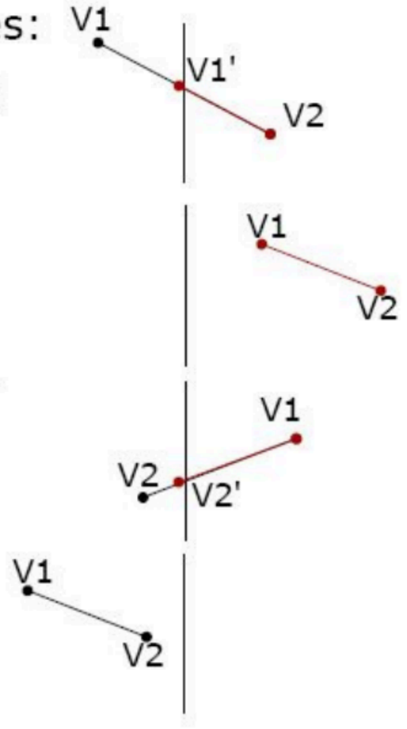
The red line represents the clipped line, and the blue line represents the original line.



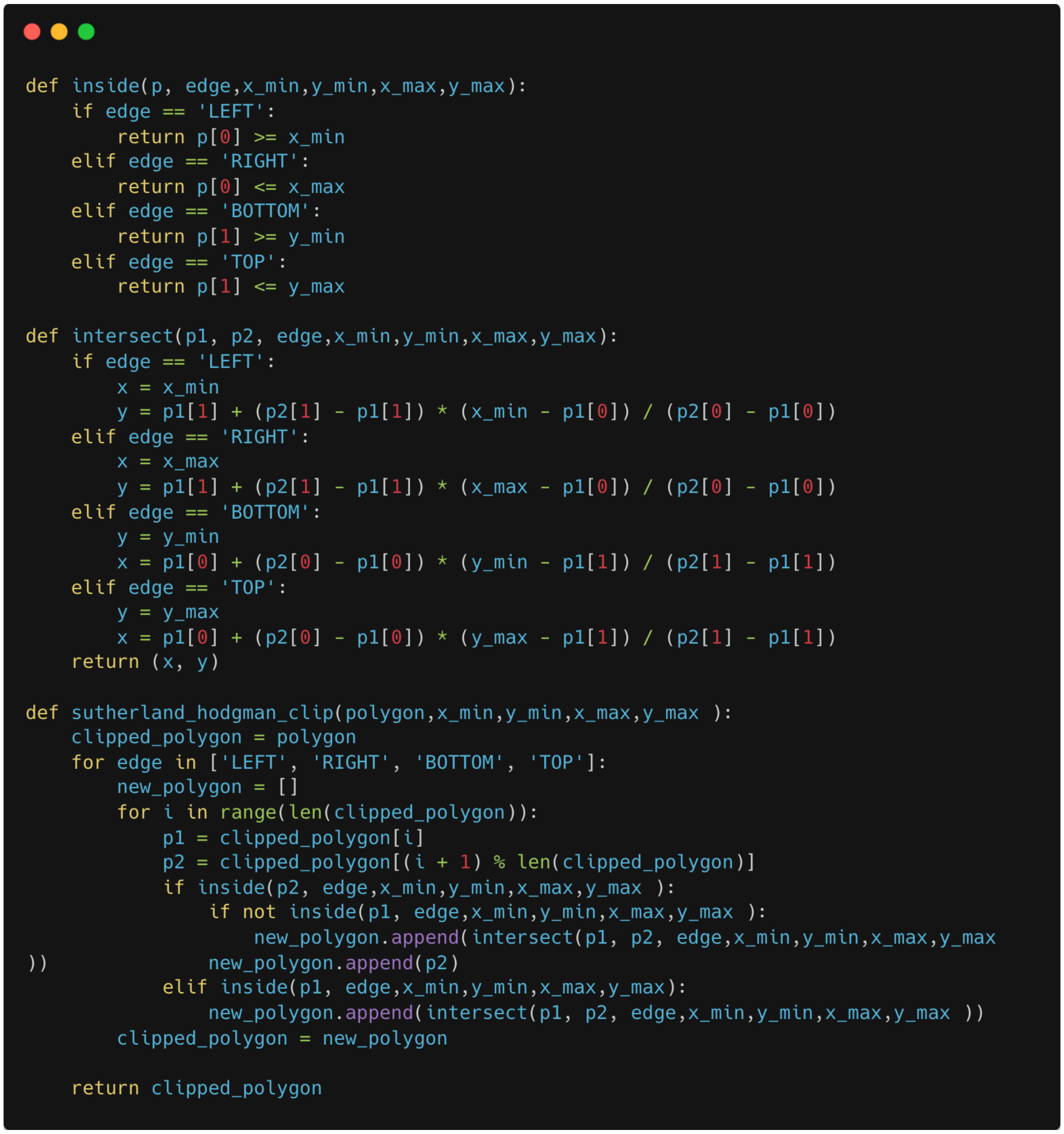
Implement Sutherland Hodgeman Polygon Clipping Algorithm

Algorithm:

- Traverse edges for borders; 4 cases:
  - V1 outside, V2 inside: take V1' and V2
  - V1 inside, V2 inside: take V1 and V2
  - V1 inside, V2 outside: take V1 and V2'
  - V1 outside, V2 outside: take none

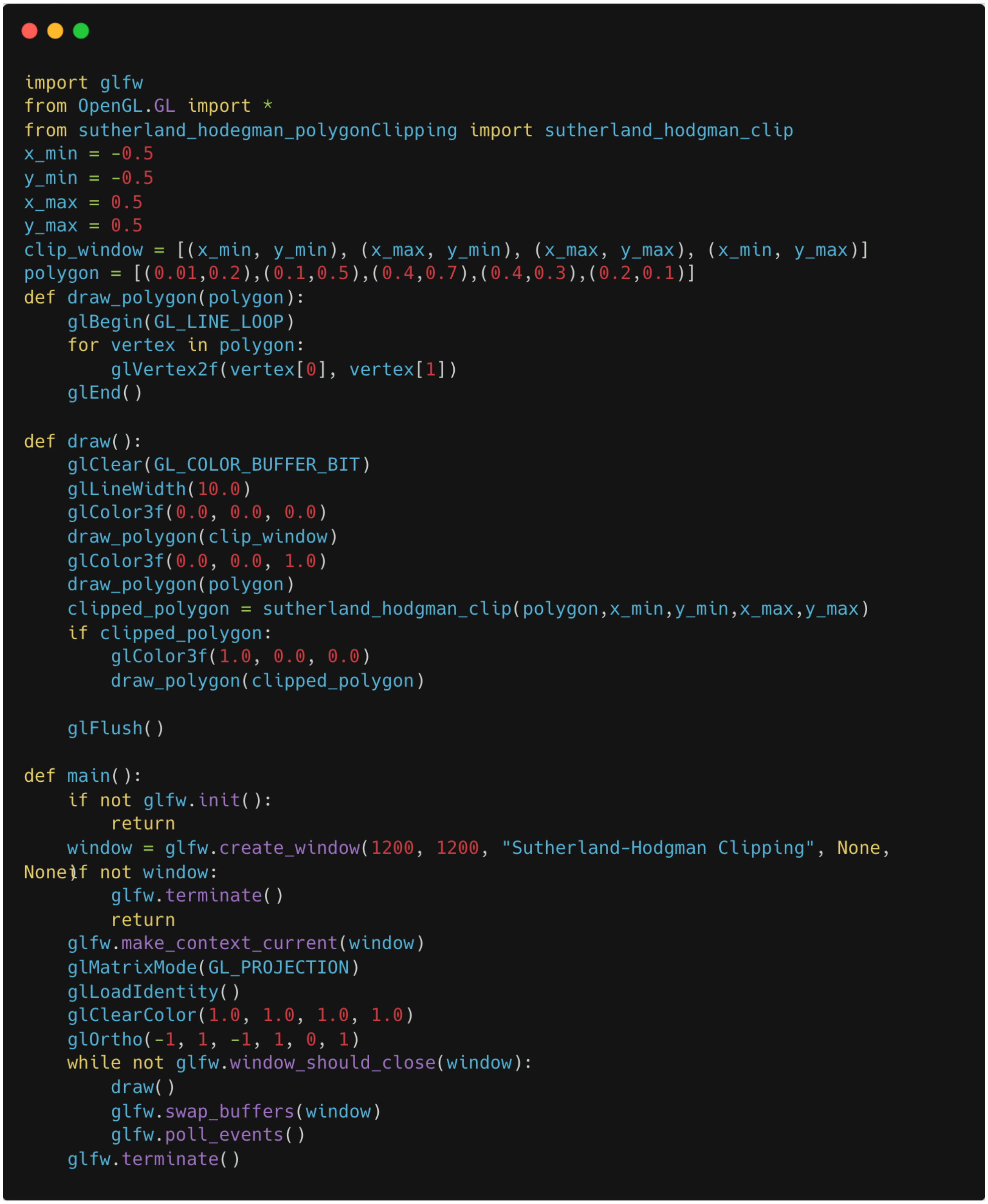


Code Implementation:



```
def inside(p, edge,x_min,y_min,x_max,y_max):
    if edge == 'LEFT':
        return p[0] >= x_min
    elif edge == 'RIGHT':
        return p[0] <= x_max
    elif edge == 'BOTTOM':
        return p[1] >= y_min
    elif edge == 'TOP':
        return p[1] <= y_max

def intersect(p1, p2, edge,x_min,y_min,x_max,y_max):
    if edge == 'LEFT':
        x = x_min
        y = p1[1] + (p2[1] - p1[1]) * (x_min - p1[0]) / (p2[0] - p1[0])
    elif edge == 'RIGHT':
        x = x_max
        y = p1[1] + (p2[1] - p1[1]) * (x_max - p1[0]) / (p2[0] - p1[0])
    elif edge == 'BOTTOM':
        y = y_min
        x = p1[0] + (p2[0] - p1[0]) * (y_min - p1[1]) / (p2[1] - p1[1])
    elif edge == 'TOP':
        y = y_max
        x = p1[0] + (p2[0] - p1[0]) * (y_max - p1[1]) / (p2[1] - p1[1])
    return (x, y)

def sutherland_hodgman_clip(polygon,x_min,y_min,x_max,y_max ):
    clipped_polygon = polygon
    for edge in ['LEFT', 'RIGHT', 'BOTTOM', 'TOP']:
        new_polygon = []
        for i in range(len(clipped_polygon)):
            p1 = clipped_polygon[i]
            p2 = clipped_polygon[(i + 1) % len(clipped_polygon)]
            if inside(p2, edge,x_min,y_min,x_max,y_max ):
                if not inside(p1, edge,x_min,y_min,x_max,y_max ):
                    new_polygon.append(intersect(p1, p2, edge,x_min,y_min,x_max,y_max
))              new_polygon.append(p2)
            elif inside(p1, edge,x_min,y_min,x_max,y_max):
                new_polygon.append(intersect(p1, p2, edge,x_min,y_min,x_max,y_max ))
        clipped_polygon = new_polygon

    return clipped_polygon
```

Displaying in OpenGL:



```python
import glfw
from OpenGL.GL import *
from sutherland_hodegman_polygonClipping import sutherland_hodgman_clip
x_min = -0.5
y_min = -0.5
x_max = 0.5
y_max = 0.5
clip_window = [(x_min, y_min), (x_max, y_min), (x_max, y_max), (x_min, y_max)]
polygon = [(0.01,0.2),(0.1,0.5),(0.4,0.7),(0.4,0.3),(0.2,0.1)]
def draw_polygon(polygon):
    glBegin(GL_LINE_LOOP)
    for vertex in polygon:
        glVertex2f(vertex[0], vertex[1])
    glEnd()

def draw():
    glClear(GL_COLOR_BUFFER_BIT)
    glLineWidth(10.0)
    glColor3f(0.0, 0.0, 0.0)
    draw_polygon(clip_window)
    glColor3f(0.0, 0.0, 1.0)
    draw_polygon(polygon)
    clipped_polygon = sutherland_hodgman_clip(polygon,x_min,y_min,x_max,y_max)
    if clipped_polygon:
        glColor3f(1.0, 0.0, 0.0)
        draw_polygon(clipped_polygon)

    glFlush()

def main():
    if not glfw.init():
        return
    window = glfw.create_window(1200, 1200, "Sutherland-Hodgman Clipping", None,
None)if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glClearColor(1.0, 1.0, 1.0, 1.0)
    glOrtho(-1, 1, -1, 1, 0, 1)
    while not glfw.window_should_close(window):
        draw()
        glfw.swap_buffers(window)
        glfw.poll_events()
    glfw.terminate()
```

Output:

The red ploygon represents the clipped polygon, and the blue ploygon represents the original polygon.