Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Computer Graphics Lab Report 05

on

‘Polygon and Line Clipping Algorithms - Lab 05 Task’

Submitted By:

Reewaj Khanal (61)

Submitted to:

Mr. Dhiraj Shrestha

Assistant Professor

Department of Computer Science and Engineering

School of Engineering

Kathmandu University

Dhulikhel, Kavre

Submission Date: Wednesday 12 June 2024

Question No. 1 Implement Liang Barsky Line Clipping algorithm

Answer:

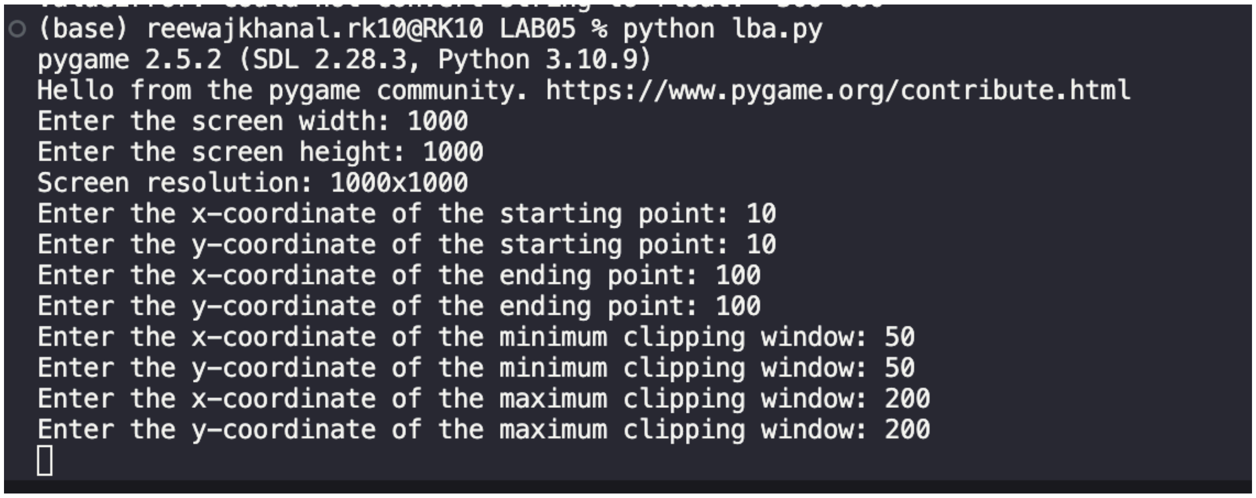
```
import pygame

from pygame.locals import *

from OpenGL.GL import *

from OpenGL.GLU import *


def liang_barsky(x0, y0, x1, y1, xmin, ymin, xmax, ymax):

   """

   Liang-Barsky line clipping algorithm.


   Parameters:

   x0, y0: float, float

       Starting coordinates of the line.

   x1, y1: float, float

       Ending coordinates of the line.

   xmin, ymin, xmax, ymax: float, float, float, float

       Coordinates of the clipping window.


   Returns:

   clipped_line: tuple

        Clipped line coordinates (x0, y0, x1, y1) or None if the line is outside the
window.

   """

   def clip(p, q, t0, t1):

       """

       Helper function to perform clipping against one boundary.

```

```
    Parameters:
    p: float
        The delta value for the current boundary.
    q: float
        The distance to the boundary.
    t0: float
        The lower bound parameter.
    t1: float
        The upper bound parameter.

    Returns:
    result: bool
        Whether the line is within the boundary.
    t0: float
        Updated lower bound parameter.
    t1: float
        Updated upper bound parameter.
    """
    if p < 0.0:
        r = q / p
        if r > t1:
            return False, t0, t1
        elif r > t0:
            t0 = r
    elif p > 0.0:
        r = q / p
        if r < t0:
            return False, t0, t1
        elif r < t1:
            t1 = r
        elif q < 0.0:
            return False, t0, t1
        return True, t0, t1

    # Calculate the differences
    dx = x1 - x0
    dy = y1 - y0

    # Coefficients and constants for inequalities
    p = [-dx, dx, -dy, dy]
    q = [x0 - xmin, xmax - x0, y0 - ymin, ymax - y0]

    # Initialize parameters
    t0, t1 = 0.0, 1.0

    # Process each boundary
    for i in range(4):
        result, t0, t1 = clip(p[i], q[i], t0, t1)
        if not result:
            return None  # Line is outside the clipping window

    # Calculate the clipped coordinates
    x0_clipped = x0 + t0 * dx
    y0_clipped = y0 + t0 * dy
    x1_clipped = x0 + t1 * dx
    y1_clipped = y0 + t1 * dy

    return x0_clipped, y0_clipped, x1_clipped, y1_clipped

def draw_line(x0, y0, x1, y1):
    """
    Draw a line using OpenGL.

    Parameters:
    x0, y0: float, float
        Starting coordinates of the line.
    x1, y1: float, float
        Ending coordinates of the line.
    """
    glBegin(GL_LINES)
    glVertex2f(x0, y0)
    glVertex2f(x1, y1)
    glEnd()

 def main():
    # Get user input for the screen resolution
    screen_width = int(input("Enter the screen width: "))
    screen_height = int(input("Enter the screen height: "))

    # Print the screen resolution
    print(f"Screen resolution: {screen_width}x{screen_height}")

    # Get user input for the line coordinates
    x0 = float(input("Enter the x-coordinate of the starting point: "))
    y0 = float(input("Enter the y-coordinate of the starting point: "))
    x1 = float(input("Enter the x-coordinate of the ending point: "))
    y1 = float(input("Enter the y-coordinate of the ending point: "))

    # Get user input for the clipping window coordinates
    xmin = float(input("Enter the x-coordinate of the minimum clipping window: "))
```

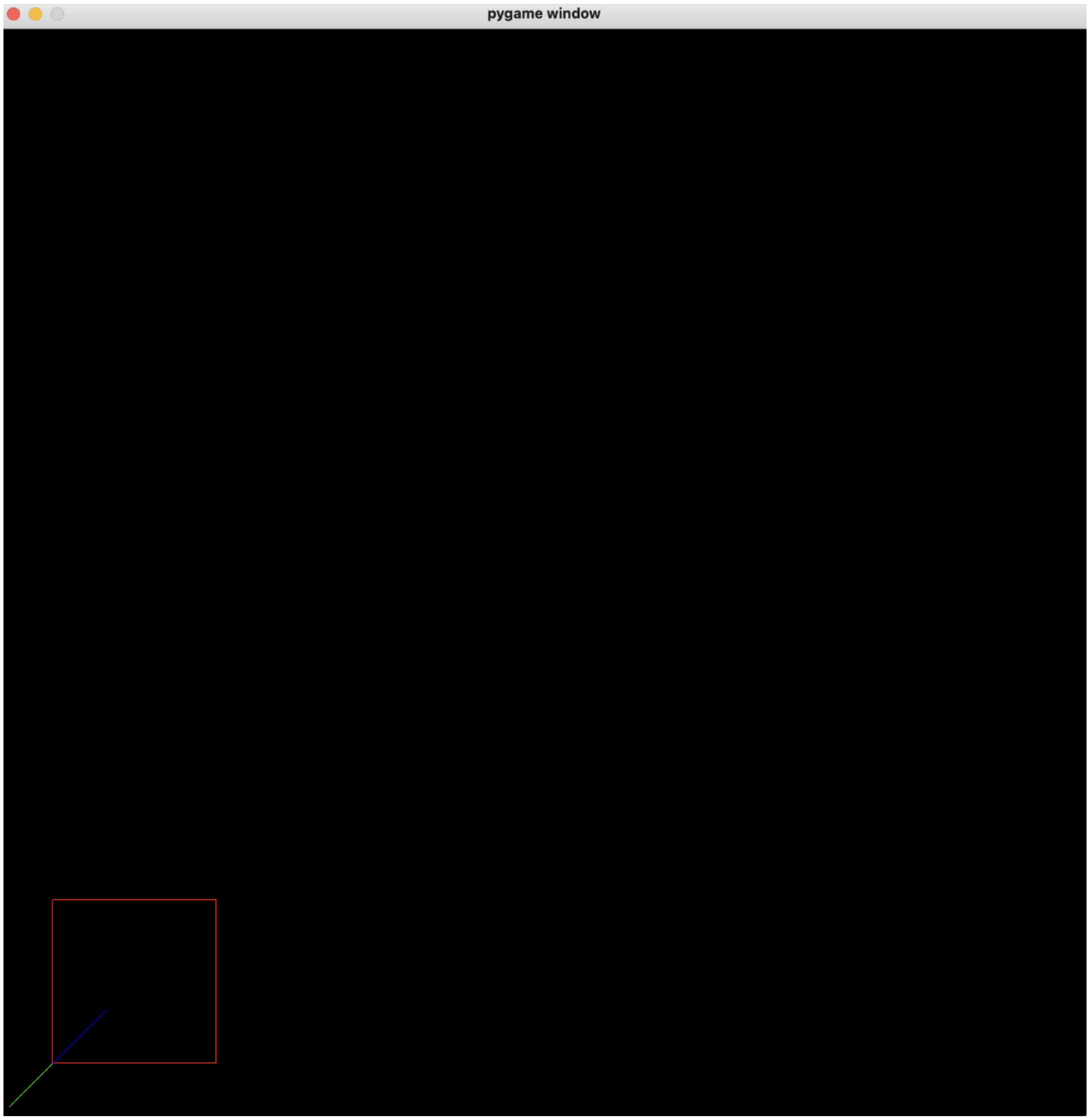
```
ymin = float(input("Enter the y-coordinate of the minimum clipping window: "))

xmax = float(input("Enter the x-coordinate of the maximum clipping window: "))

ymax = float(input("Enter the y-coordinate of the maximum clipping window: "))


# Perform the clipping

clipped_line = liang_barsky(x0, y0, x1, y1, xmin, ymin, xmax, ymax)


# Initialize Pygame and set up the OpenGL display

pygame.init()

display = (screen_width, screen_height)

pygame.display.set_mode(display, DOUBLEBUF | OPENGL)

gluOrtho2D(0, screen_width, 0, screen_height)


running = True

while running:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            running = False


    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)


    # Draw the clipping window

    glColor3f(1.0, 0.0, 0.0)

    glBegin(GL_LINE_LOOP)

    glVertex2f(xmin, ymin)

    glVertex2f(xmax, ymin)

    glVertex2f(xmax, ymax)

    glVertex2f(xmin, ymax)

    glEnd()
```

```
        # Draw the original line

        glColor3f(0.0, 1.0, 0.0)

        draw_line(x0, y0, x1, y1)


        # Draw the clipped line

        if clipped_line:

            glColor3f(0.0, 0.0, 1.0)

            draw_line(*clipped_line)


        pygame.display.flip()

        pygame.time.wait(10)


    pygame.quit()


if __name__ == "__main__":

    main()
```

Inputs and Outputs:



```
(base) reewajkhanal.rk10@RK10 LAB05 % python lba.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.9)
Hello from the pygame community. https://www.pygame.org/contribute.html
Enter the screen width: 1000
Enter the screen height: 1000
Screen resolution: 1000x1000
Enter the x-coordinate of the starting point: 10
Enter the y-coordinate of the starting point: 10
Enter the x-coordinate of the ending point: 100
Enter the y-coordinate of the ending point: 100
Enter the x-coordinate of the minimum clipping window: 50
Enter the y-coordinate of the minimum clipping window: 50
Enter the x-coordinate of the maximum clipping window: 200
Enter the y-coordinate of the maximum clipping window: 200
```


pygame window

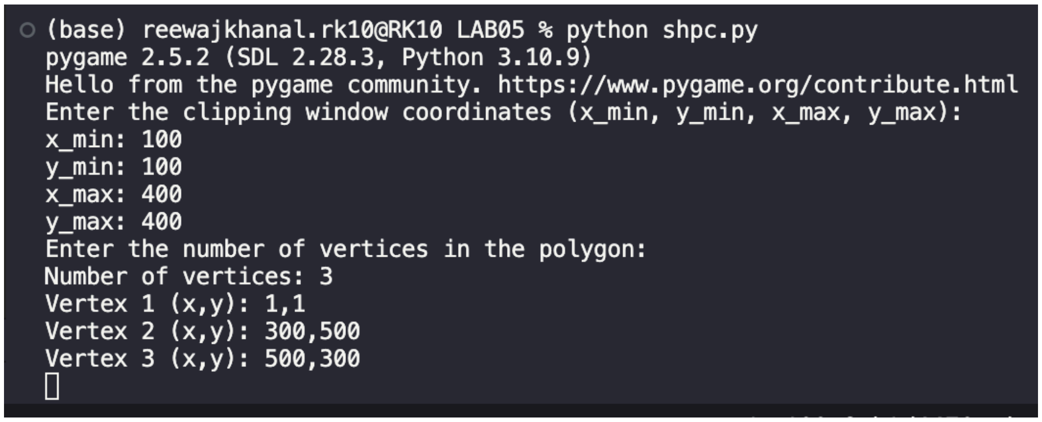
Question No. 2 Implement Sutherland Hodgemann polygon clipping algorithm

Answer:

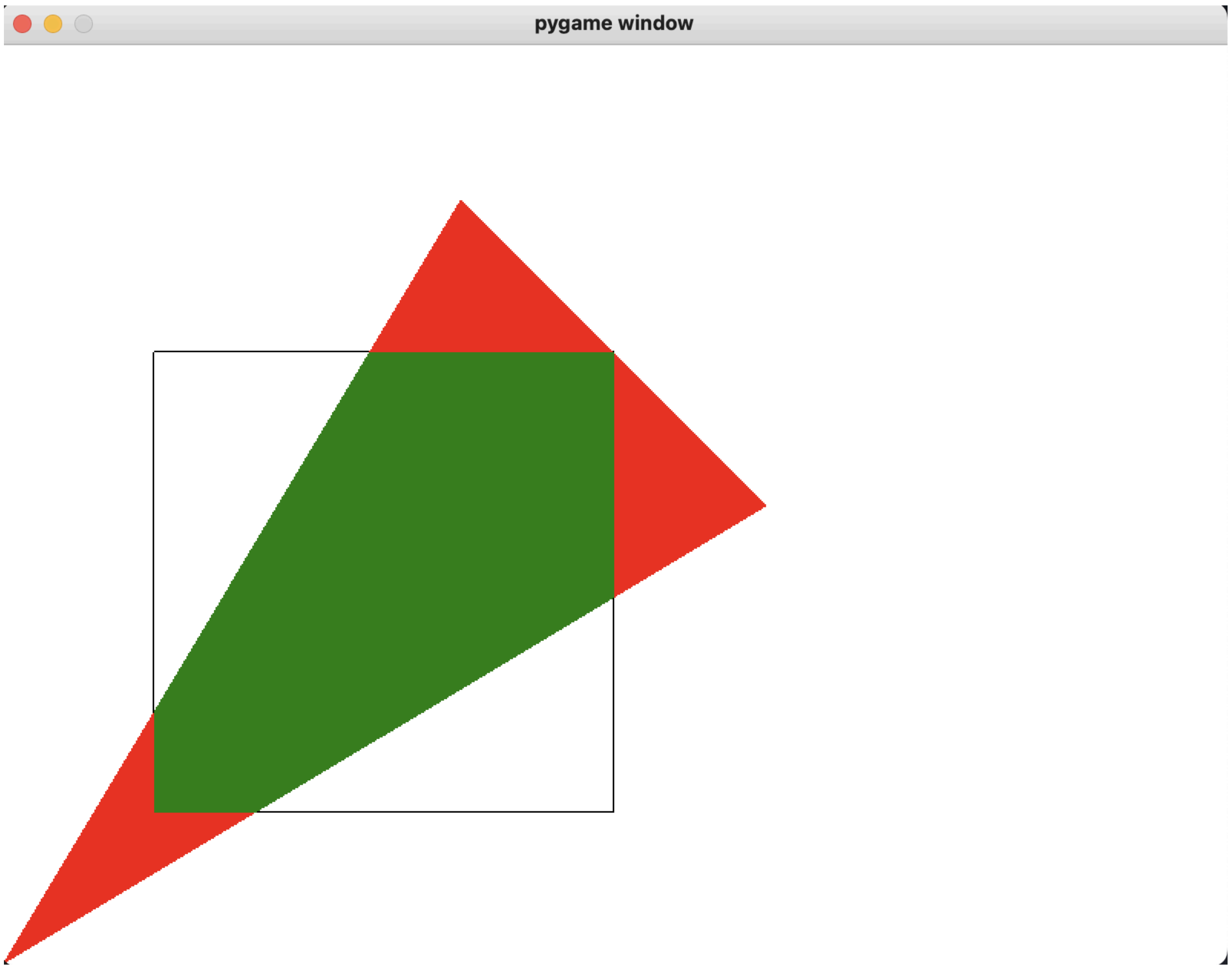
```
import pygame

from pygame.locals import *

from OpenGL.GL import *

from OpenGL.GLUT import *


# Define the clip region boundaries

LEFT = 0

RIGHT = 1

BOTTOM = 2

TOP = 3


# Function to check if a point is inside the clip boundary

def inside(point, boundary, value):

   if boundary == LEFT:

       return point[0] >= value

   elif boundary == RIGHT:

       return point[0] <= value

   elif boundary == BOTTOM:

       return point[1] >= value

   elif boundary == TOP:

       return point[1] <= value

   return False


# Function to compute the intersection point with the clip boundary

def intersect(point1, point2, boundary, value):

   if boundary == LEFT or boundary == RIGHT:
       x = value
         y = point1[1] + (point2[1] - point1[1]) * (value - point1[0]) / (point2[0] -
point1[0])
   elif boundary == BOTTOM or boundary == TOP:
       y = value
         x = point1[0] + (point2[0] - point1[0]) * (value - point1[1]) / (point2[1] -
point1[1])
   return [x, y]

# Sutherland-Hodgman polygon clipping algorithm
def sutherland_hodgman_clip(polygon, clip_window):
   clipped_polygon = polygon
   for boundary, value in clip_window.items():
       input_list = clipped_polygon
       clipped_polygon = []
       if not input_list:
           break
       s = input_list[-1]

       for e in input_list:
           if inside(e, boundary, value):
               if not inside(s, boundary, value):
                   clipped_polygon.append(intersect(s, e, boundary, value))
               clipped_polygon.append(e)
           elif inside(s, boundary, value):
               clipped_polygon.append(intersect(s, e, boundary, value))
           s = e

   return clipped_polygon

# Function to get user input for the clipping window
```

```
def get_window():

   print("Enter the clipping window coordinates (x_min, y_min, x_max, y_max):")

   x_min = int(input("x_min: "))

   y_min = int(input("y_min: "))

   x_max = int(input("x_max: "))

   y_max = int(input("y_max: "))

   return {LEFT: x_min, RIGHT: x_max, BOTTOM: y_min, TOP: y_max}


# Function to get user input for the polygon vertices

def get_polygon():

   print("Enter the number of vertices in the polygon:")

   num_vertices = int(input("Number of vertices: "))

   polygon = []

   for i in range(num_vertices):

       xy = input(f"Vertex {i+1} (x,y): ")

       x, y = map(int, xy.split(','))

       polygon.append([x, y])

   return polygon


def main():

   # Initialize Pygame and set up the OpenGL context

   pygame.init()

   display = (800, 600)

   pygame.display.set_mode(display, DOUBLEBUF | OPENGL)

   glOrtho(0, 800, 0, 600, -1, 1)


   clip_window = get_window()


   running = True

   while running:
       # Get user input for the polygon vertices
       polygon = get_polygon()


       # Perform the clipping
       clipped_polygon = sutherland_hodgman_clip(polygon, clip_window)


       drawing = True
       while drawing:
           for event in pygame.event.get():
               if event.type == pygame.QUIT:
                   running = False
                   drawing = False


           glClearColor(1.0, 1.0, 1.0, 1.0)  # Set background color to white
           glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)


           # Draw the clipping window
           glColor3f(0, 0, 0)  # Black
           glBegin(GL_LINE_LOOP)
           glVertex2f(clip_window[LEFT], clip_window[BOTTOM])
           glVertex2f(clip_window[RIGHT], clip_window[BOTTOM])
           glVertex2f(clip_window[RIGHT], clip_window[TOP])
           glVertex2f(clip_window[LEFT], clip_window[TOP])
           glEnd()


           # Draw and fill the original polygon
           glColor3f(1, 0, 0)  # Red
           glBegin(GL_POLYGON)
           for vertex in polygon:
               glVertex2f(vertex[0], vertex[1])
           glEnd()

           # Draw and fill the clipped polygon if it exists
           if clipped_polygon:
               glColor3f(0, 0.5, 0)  # Dark Green
               glBegin(GL_POLYGON)
               for vertex in clipped_polygon:
                   glVertex2f(vertex[0], vertex[1])
               glEnd()

           pygame.display.flip()
           pygame.time.wait(10)

       # Ask the user if they want to draw another polygon
       another_polygon = input("Draw another polygon? (y/n): ").strip().lower()
       if another_polygon != 'y':
           running = False

   pygame.quit()

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

Inputs and Outputs:



```
(base) reewajkhanal.rk10@RK10 LAB05 % python shpc.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.9)
Hello from the pygame community. https://www.pygame.org/contribute.html
Enter the clipping window coordinates (x_min, y_min, x_max, y_max):
x_min: 100
y_min: 100
x_max: 400
y_max: 400
Enter the number of vertices in the polygon:
Number of vertices: 3
Vertex 1 (x,y): 1,1
Vertex 2 (x,y): 300,500
Vertex 3 (x,y): 500,300
```


pygame window