Computer Graphics Lab - CS352

Weekly Assignment Report

Question 1:

Python

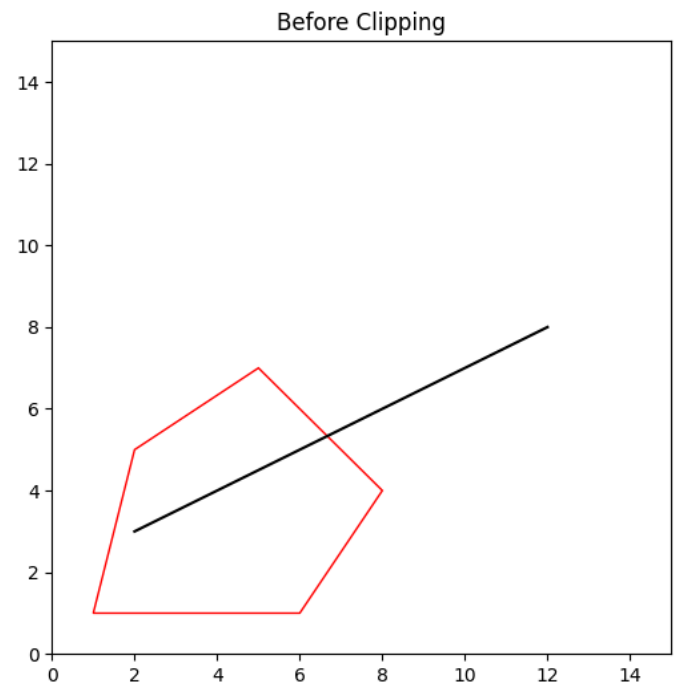
```python
import matplotlib.pyplot as plt

import numpy as np

# Function to draw a line in matplotlib

def drawline(p0, p1):

    plt.plot([p0[0], p1[0]], [p0[1], p1[1]], 'k-')


# Function to draw a polygon, given vertices

def drawPolygon(vertices):

    vertices.append(vertices[0]) # repeat the first point to create a 'closed loop'

    xs, ys = zip(*vertices) # create lists of x and y values

    plt.fill(xs, ys, edgecolor='r', fill=False)


# Function to take dot product

def dot(p0, p1):

    return p0[0] * p1[0] + p0[1] * p1[1]


# Function to calculate the max from a list of floats

def max(t):
```

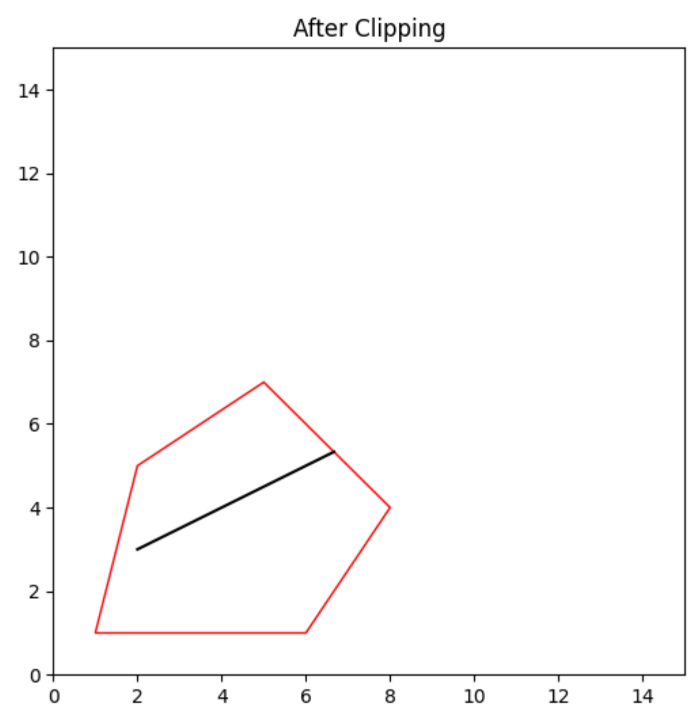
```python
    return np.max(t)


# Function to calculate the min from a list of floats
def min(t):
    return np.min(t)


# Cyrus Beck function, returns a pair of values
# that are then displayed as a line
def CyrusBeck(vertices, line):
    n = len(vertices)
    # Calculate the direction vector of the line
    P1_P0 = (line[1][0] - line[0][0], line[1][1] - line[0][1])
    # Calculate the normal vectors for each edge of the polygon
    normal = [(vertices[i][1] - vertices[(i + 1) % n][1], vertices[(i + 1) %
n][0] - vertices[i][0]) for i in range(n)]
    # Calculate the vector from the line's starting point to each vertex of the
polygon
    P0_PEi = [(vertices[i][0] - line[0][0], vertices[i][1] - line[0][1]) for i
in range(n)]
    # Calculate the numerator for the parameter t for each edge
    numerator = [dot(normal[i], P0_PEi[i]) for i in range(n)]
    # Calculate the denominator for the parameter t for each edge
    denominator = [dot(normal[i], P1_P0) for i in range(n)]
    # Calculate the parameter t for each edge
```

```
    t = [numerator[i] / denominator[i] if denominator[i] != 0 else 0 for i in range(n)]

    # Separate the t values into entering and leaving points
    tE = [t[i] for i in range(n) if denominator[i] > 0]
    tL = [t[i] for i in range(n) if denominator[i] < 0]
    tE.append(0)  # Add 0 to the entering points
    tL.append(1)  # Add 1 to the leaving points

    # Calculate the maximum entering point and the minimum leaving point
    temp = [max(tE), min(tL)]

    # If the maximum entering point is greater than the minimum leaving point, the line is outside the polygon
    if temp[0] > temp[1]:
        return None

    # Calculate the clipped line segment
    newPair = [(line[0][0] + P1_P0[0] * temp[0], line[0][1] + P1_P0[1] * temp[0]),
               (line[0][0] + P1_P0[0] * temp[1], line[0][1] + P1_P0[1] * temp[1])]
    return newPair


vertices = [(1, 1), (6, 1), (8, 4), (5, 7), (2, 5)]
line = [(2, 3), (12, 8)]  # New line coordinates
```

```
    # Before Clipping

plt.figure(figsize=(6, 6))

plt.title('Before Clipping')

drawPolygon(vertices)

drawline(line[0], line[1])  # Draw the original line

plt.xlim(0, 15)

plt.ylim(0, 15)

plt.show()


# After Clipping

newPair = CyrusBeck(vertices, line)

print("New line points", newPair)

if newPair is not None:

  plt.figure(figsize=(6, 6))

  plt.title('After Clipping')

  drawPolygon(vertices)

  drawline(newPair[0], newPair[1])  # Draw the clipped line

  plt.xlim(0, 15)

  plt.ylim(0, 15)

  plt.show()
```

Output:





New line points [(2.0, 3.0), (6.666666666666667, 5.333333333333334)]

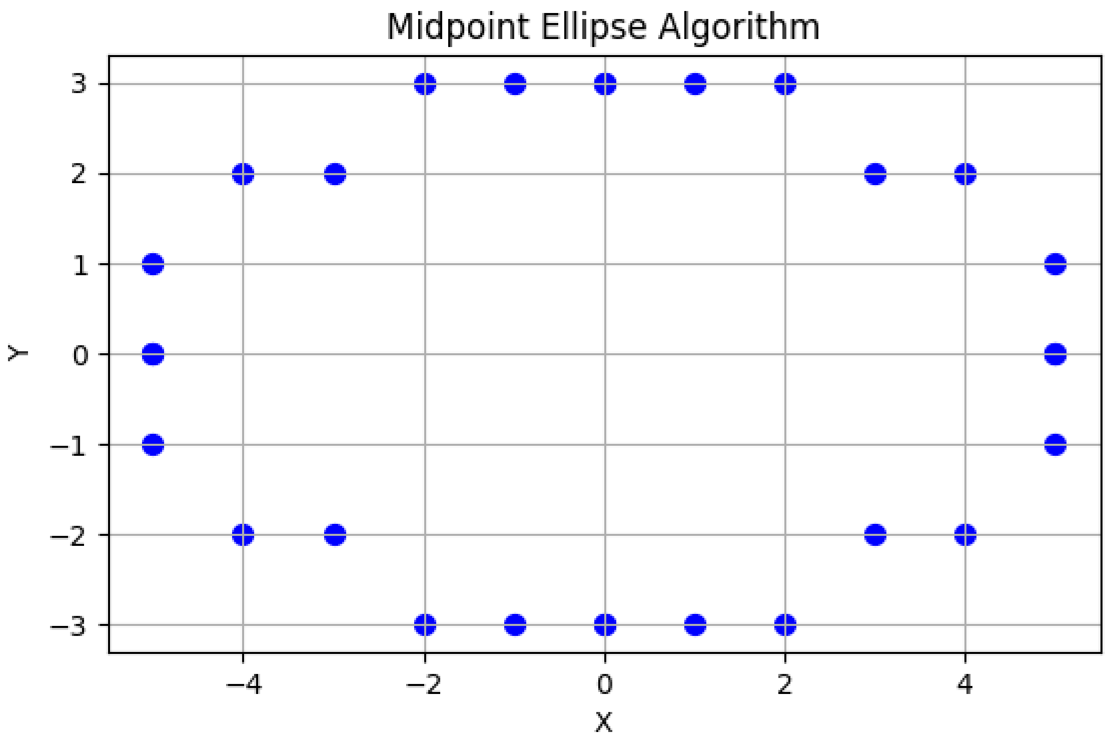
Question 2:

```python
import matplotlib.pyplot as plt

import numpy as np

def midptellipse(rx, ry, xc, yc):

    x = 0

    y = ry


    x_points = []

    y_points = []


    # Initial decision parameter for region 1

    d1 = (ry**2) - (rx**2 * ry) + (0.25 * rx**2)

    dx = 2 * ry**2 * x

    dy = 2 * rx**2 * y


    # Region 1

    while dx < dy:

        # Store points using 4-way symmetry

        x_points.extend([x + xc, -x + xc, x + xc, -x + xc])

        y_points.extend([y + yc, y + yc, -y + yc, -y + yc])


        if d1 < 0:  # Choose E pixel
```

```
            x += 1
            dx += 2 * ry**2
            d1 += dx + ry**2
        else:  # Choose SE pixel
            x += 1
            y -= 1
            dx += 2 * ry**2
            dy -= 2 * rx**2
            d1 += dx - dy + ry**2

    # Initial decision parameter for region 2
    d2 = ((ry**2) * ((x + 0.5)**2)) + ((rx**2) * ((y - 1)**2)) - (rx**2 * ry**2)

    # Region 2
    while y >= 0:
        # Store points using 4-way symmetry
        x_points.extend([x + xc, -x + xc, x + xc, -x + xc])
        y_points.extend([y + yc, y + yc, -y + yc, -y + yc])

        if d2 > 0:  # Choose S pixel
            y -= 1
            dy -= 2 * rx**2
            d2 += rx**2 - dy
```

```
        else:  # Choose SE pixel
            y -= 1
            x += 1
            dx += 2 * ry**2
            dy -= 2 * rx**2
            d2 += dx - dy + rx**2

    # Plot the computed points
    plt.scatter(x_points, y_points, s=50, color="blue")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.title("Midpoint Ellipse Algorithm")
    plt.gca().set_aspect('equal', adjustable='box')
    plt.grid(True)
    plt.show()

midptellipse(5, 3, 0, 0)
```

Output:

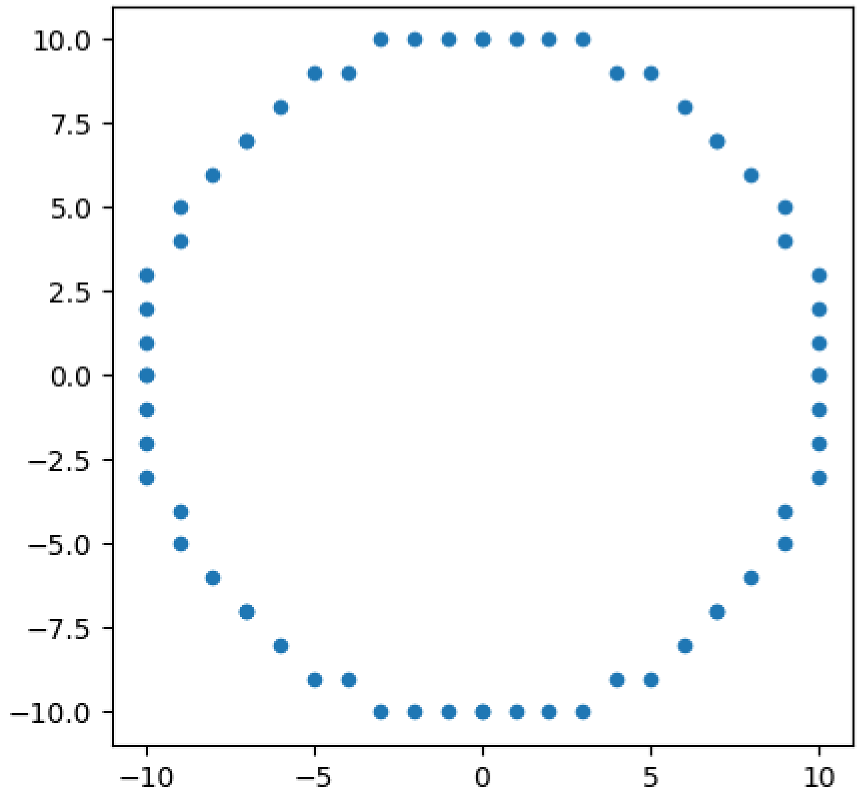


Question 3:

```python
import matplotlib.pyplot as plt

import numpy as np

def draw_circle(xc, yc, r):

    x, y = 0, r  #starting from the topmost point (0,r) on the circle

    p = 1 - r    # Initial decision parameter to move East (E) or South-East
(SE)

    points = []  # store the computed circle points


    def plot_circle_points(xc, yc, x, y):

        points.extend([(xc + x, yc + y), (xc - x, yc + y),

                       (xc + x, yc - y), (xc - x, yc - y),

                       (xc + y, yc + x), (xc - y, yc + x),

                       (xc + y, yc - x), (xc - y, yc - x)])


    plot_circle_points(xc, yc, x, y)


    while x < y:

        print(points)

        x = x + 1

        if p < 0:

            p += 2*x + 1  # Move Right

        else:
```

```
            y -= 1
            p += 2*x - 2*y + 1  # Move Diagonally

        plot_circle_points(xc, yc, x, y)

    # Plot the points
    #print(points)
    x_cordinate, y_cordinate = zip(*points)
    plt.scatter(x_cordinate, y_cordinate, s=20)
    plt.gca().set_aspect('equal')
    #plt.show()

# Example: Draw a circle with center (0,0) and radius 10
draw_circle(0, 0, 10)
```

Output:



Question 4:

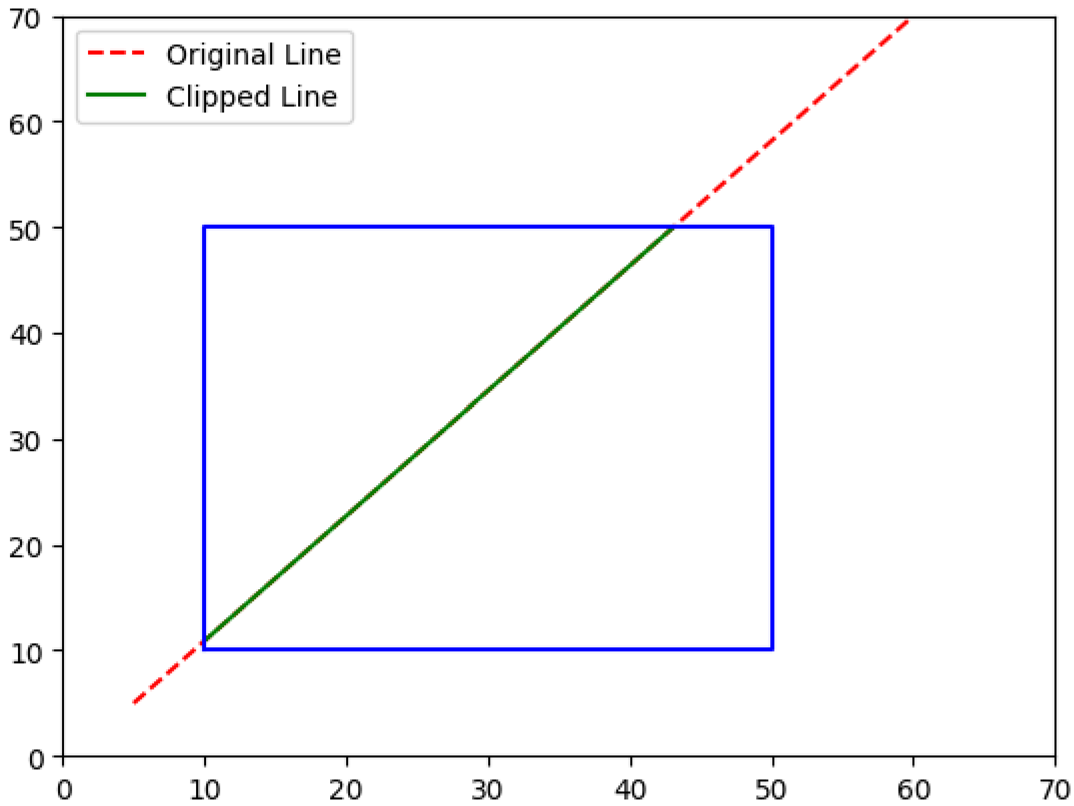
```python
import matplotlib.pyplot as plt

import numpy as np


# Define clipping window

xmin, ymin, xmax, ymax = 10, 10, 50, 50


# Define region codes

INSIDE = 0  # 0000

LEFT = 1    # 0001

RIGHT = 2   # 0010

BOTTOM = 4  # 0100

TOP = 8     # 1000


def compute_code(x, y):

    code = INSIDE

    if x < xmin:

      code = code | LEFT

    if x > xmax:

      code = code | RIGHT

    if y < ymin:
```

```
      code = code | BOTTOM
    if y > ymax:
      code = code | TOP
    return code


def cohen_sutherland_clip(x1, y1, x2, y2):
    code1 = compute_code(x1, y1)
    code2 = compute_code(x2, y2)
    accept = False

    while True:        # Infinite loop with two exit criteria.
        if code1 == 0 and code2 == 0:    #Trivial Accept
            accept = True
            break                        # Loop Exit criteria 1

        #if (code1&code2 != 0)⇒ Reject the line
        elif code1 & code2:              #Trivial Reject
            break                        # Loop Exit criteria 2
        else:
            if code1:
                code_out = code1
            else:
                 code_out = code2
```

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

```python
        return None

# Test line
x1, y1, x2, y2 = 5, 5, 60, 70

clipped_line = cohen_sutherland_clip(x1, y1, x2, y2)
fig, ax = plt.subplots()
ax.set_xlim(0, 70)
ax.set_ylim(0, 70)
ax.plot([x1, x2], [y1, y2], 'r--', label="Original Line")

if clipped_line:
    x1_clip, y1_clip, x2_clip, y2_clip = clipped_line
    ax.plot([x1_clip, x2_clip], [y1_clip, y2_clip], 'g-', label="Clipped Line")

# Draw clipping window
ax.plot([xmin, xmax, xmax, xmin, xmin], [ymin, ymin, ymax, ymax, ymin], 'b-')

ax.legend()
plt.show()
```

Output:



Code used is here for execution.