Submitted To : Sir Deepak Sharma

Submitted By : Khushi Chhatwani

Course: B.Sc (hons.) Computer Science, III Year, VI Semester

College Roll no. : CSC/21/55

University Roll no. : 21059570021

Practical file :Computer Graphics

1) Write a program to implement DDA and Bresenham's line drawing algorithm.

CODE:

```
import matplotlib.pyplot as plt
def round(a):
return int(a + 0.5)
def dda(x1, y1, x2, y2):
dx = x2 - x1
dy = y2 - y1
steps = abs(dx) if abs(dx) > abs(dy) else abs(dy)
Xinc = dx / steps
Yinc = dy / steps
x, y = x1, y1
points = []
for _ in range(steps):
points.append((round(x), round(y)))
x += Xinc
y += Yinc
return points
def bresenham(x1, y1, x2, y2):
dx = abs(x2 - x1)
dy = abs(y2 - y1)
slope = dy / dx if dx != 0 else float('inf')
x, y = x1, y1
points = []
if slope <= 1:
p = 2 * dy - dx
for _ in range(dx):
points.append((x, y))
x += 1
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

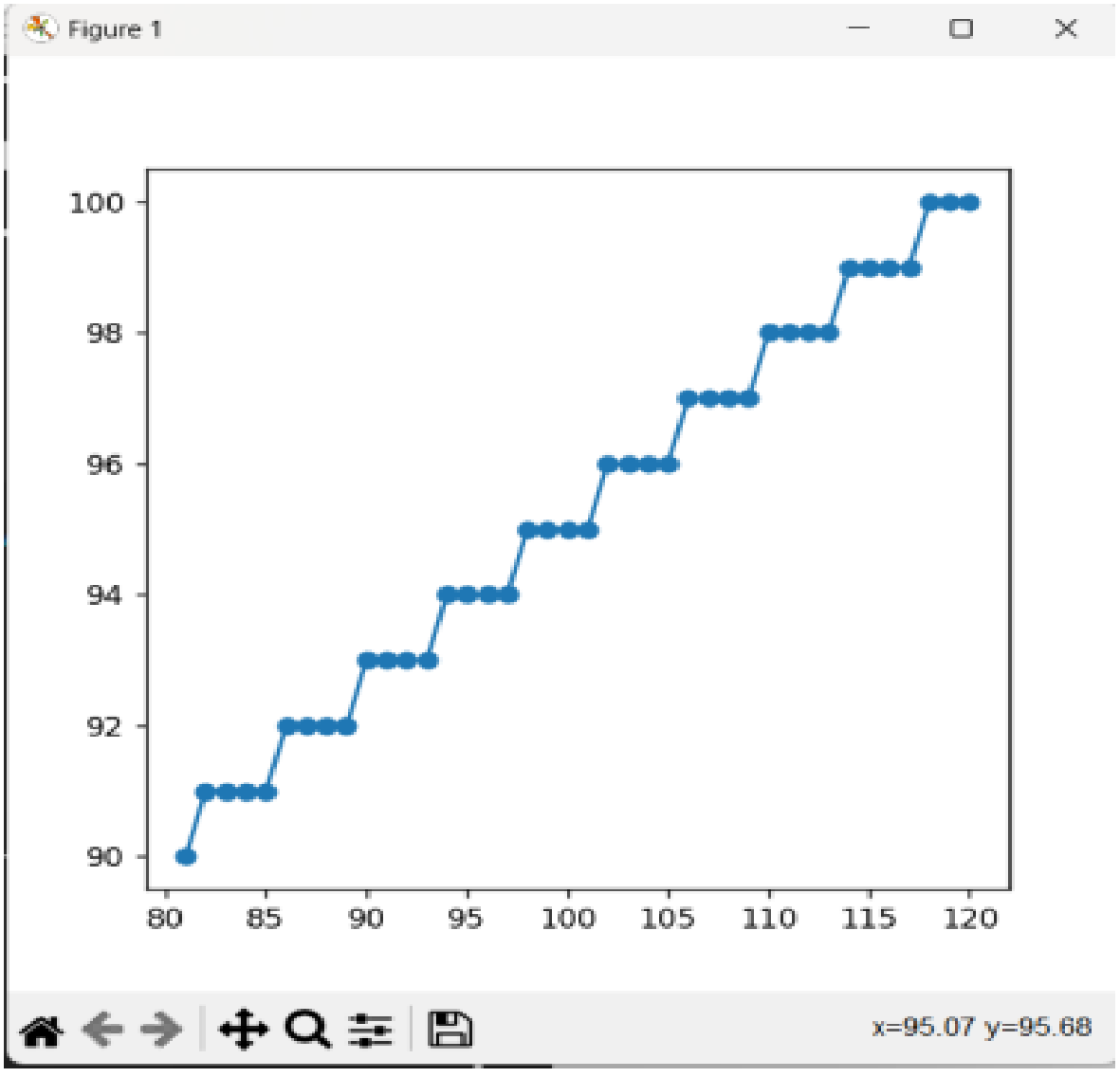
```
28  if p < 0:
29  p = p + 2 * dy
30  else:
31  y += 1
32  p = p + 2 * dy - 2 * dx
33  else:
34  p = 2 * dx - dy
35  for _ in range(dy):
36  points.append((x, y))
37  y += 1
38  if p < 0:
39  p = p + 2 * dx
40  else:
41  Garima Rohilla /21059570010
42  PRACTICAL FILE : COMPUTER GRAPHICS
43  x += 1
44  p = p + 2 * dx - 2 * dy
45  return points
46  def plot_line(points):
47  plt.figure(figsize=(5,5))
48  plt.plot(*zip(*points), marker='o') plt.show()
49  if __name__ == "__main__":
50  x1 = int(input("Enter x1: "))
51  y1 = int(input("Enter y1: "))
52  x2 = int(input("Enter x2: "))
53  y2 = int(input("Enter y2: "))
54  print("DDA:")
```

```
51  y1 = int(input("Enter y1: "))
52  x2 = int(input("Enter x2: "))
53  y2 = int(input("Enter y2: "))
54  print("DDA:")
55  points = dda(x1, y1, x2, y2)
56  plot_line(points)
57  print("Bresenham:")
58  points = bresenham(x1, y1, x2, y2)
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

OUTPUT:

```
PS C:\Users\kngou\OneDrive\Desktop\CG> python -u "c:\Users\kngou\OneDrive\Desktop\CG\P01.py"
Enter x1: 120
Enter y1: 100
Enter x2: 80
Enter y2: 90
DDA:
```



Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

Q2) Write a program to implement mid-point circle drawing algorithm.

CODE:

```
import matplotlib.pyplot as plt
import numpy as np
def mid_point_circle_draw(x_centre, y_centre, r):
x = r
y = 0
points = []
# Printing the initial point on the axes after translation points.append((x + x_centre,
# When radius is zero, only a single point is printed
if r > 0:
points.append((-x + x_centre, -y + y_centre))
points.append((y + x_centre, x + y_centre))
points.append((-y + x_centre, -x + y_centre))
# Initialising the value of P
P = 1 - r
while x > y:
y += 1
# Mid-point inside or on the perimeter
if P <= 0:
P = P + 2 * y + 1
# Mid-point outside the perimeter
else:
x -= 1
P = P + 2 * y - 2 * x + 1
# All the perimeter points have already been printed
if x < y:
break
# Printing the generated point its reflection in the other octants  after translation
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

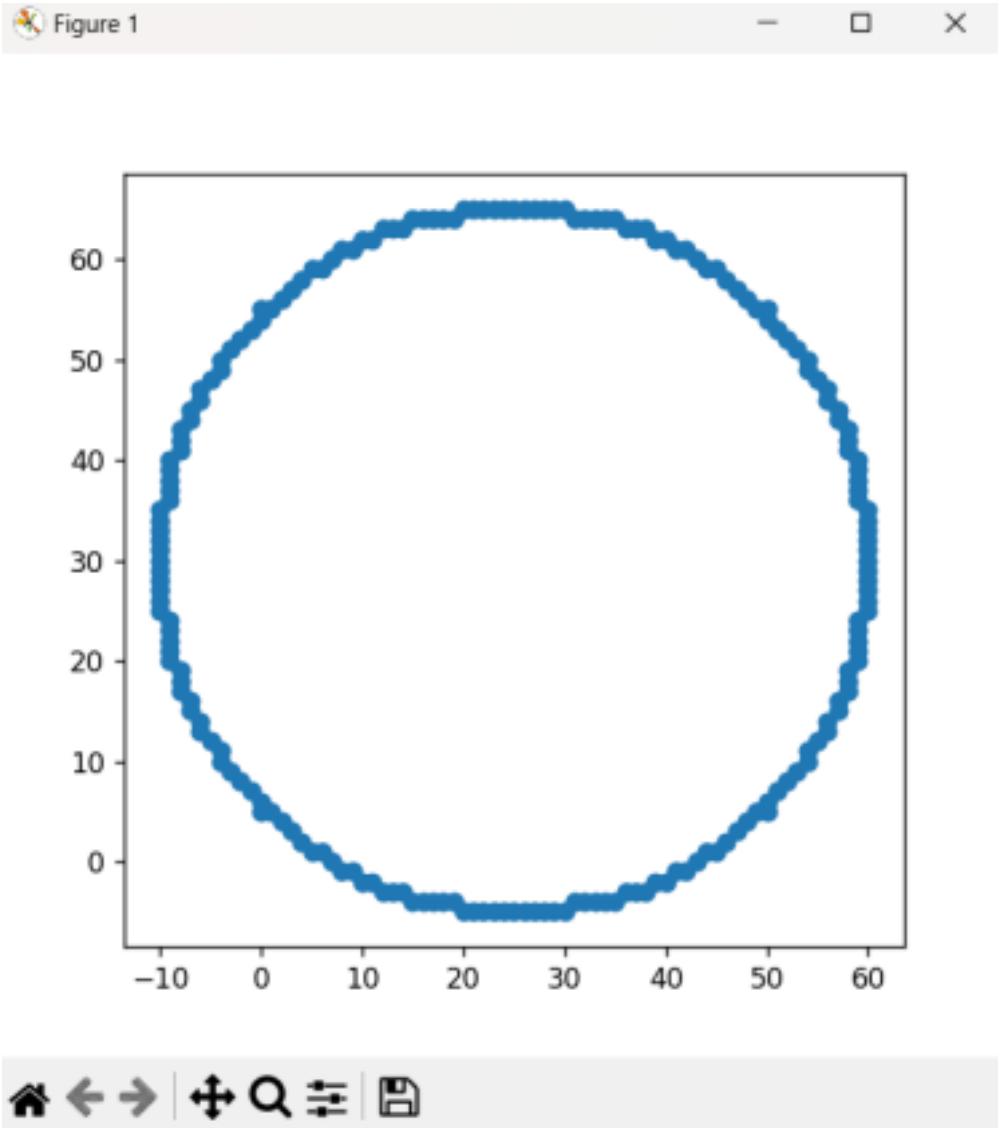
```
23      P = P + 2 * y - 2 * x + 1
24      # All the perimeter points have already been printed
25      if x < y:
26      break
27      # Printing the generated point its reflection in the other octants  after translation
28      points.append((x + x_centre, y + y_centre))
29      points.append((-x + x_centre, y + y_centre))
30      points.append((x + x_centre, -y + y_centre))
31      points.append((-x + x_centre, -y + y_centre))
32      # If the generated point is on the line x = y then the perimeter  points have already b
33      if x != y:
34      points.append((y + x_centre, x + y_centre))
35      points.append((-y + x_centre, x + y_centre))
36      points.append((y + x_centre, -x + y_centre))
37      points.append((-y + x_centre, -x + y_centre))
38      return points
39      def plot_circle(points):
40      plt.figure(figsize=(5,5))
41      plt.scatter(*zip(*points), marker='o')
42      plt.show()
43      if __name__ == "__main__":
44      x_centre = int(input("Enter x_centre: "))
45      y_centre = int(input("Enter y_centre: "))
46      r = int(input("Enter radius: "))
47      points = mid_point_circle_draw(x_centre, y_centre, r) plot_circle(points)
48
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\kngou\OneDrive\Desktop\CG> python -u "c:\Users\kngou\OneDrive\Desktop\CG\P02.py"
Enter x_centre: 25
Enter y_centre: 30
Enter radius: 35
```



Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

Q3)Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

CODE:

```
import matplotlib.pyplot as plt
# Defining region codes
INSIDE = 0 # 0000
LEFT = 1 # 0001
RIGHT = 2 # 0010
BOTTOM = 4 # 0100
TOP = 8 # 1000
# Function to compute region code for a point(x, y)
def compute_code(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
# Implementing Cohen-Sutherland algorithm
# Clipping a line from P1 = (x1, y1) to P2 = (x2, y2)
def cohen_sutherland(x1, y1, x2, y2, x_min, y_min, x_max, y_max): # Compute region codes for P1, P2
code1 = compute_code(x1, y1, x_min, y_min, x_max, y_max)
code2 = compute_code(x2, y2, x_min, y_min, x_max, y_max)
accept = False
while True:
# If both endpoints lie within rectangle
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

```
28    if code1 == 0 and code2 == 0:
29    accept = True
30    break
31    # If both endpoints are outside rectangle
32    elif (code1 & code2) != 0:
33    break
34    else:
35    # Some segment of line lies within the rectangle
36    # At least one endpoint is outside the rectangle, pick it.
37    code_out = code1 if code1 != 0 else code2
38    # Find intersection point
39    if code_out & TOP:
40    # point is above the clip rectangle
41    x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1)
42    y = y_max
43    elif code_out & BOTTOM:
44    # point is below the rectangle
45    x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1)
46    y = y_min
47    elif code_out & RIGHT:
48    # point is to the right of rectangle
49    y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1)
50    x = x_max
51    elif code_out & LEFT:
52    # point is to the left of rectangle
53    y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1)
54    x = x_min
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

```
    # Now intersection point x, y is found
    # We replace point outside rectangle by intersection point
    if code_out == code1:
    x1, y1 = x, y
    code1 = compute_code(x1, y1, x_min, y_min, x_max, y_max)
    else:
    x2, y2 = x, y
    code2 = compute_code(x2, y2, x_min, y_min, x_max, y_max)
    if accept:
    return ((x1, y1), (x2, y2))
    else:
    return None
    def plot_line(x1, y1, x2, y2, x_min, y_min, x_max, y_max):
    plt.figure()
    plt.plot([x1, x2], [y1, y2], label='Line')
    plt.plot([x_min, x_max, x_max, x_min, x_min], [y_min, y_min, y_max, y_ma
    plt.legend()
    plt.show()
    if __name__ == "__main__":
    x1 = int(input("Enter x1: "))
    y1 = int(input("Enter y1: "))
    x2 = int(input("Enter x2: "))
    y2 = int(input("Enter y2: "))
    x_min = int(input("Enter x_min: "))
    y_min = int(input("Enter y_min: "))
    x_max = int(input("Enter x_max: "))
    y_max = int(input("Enter y_max: "))
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

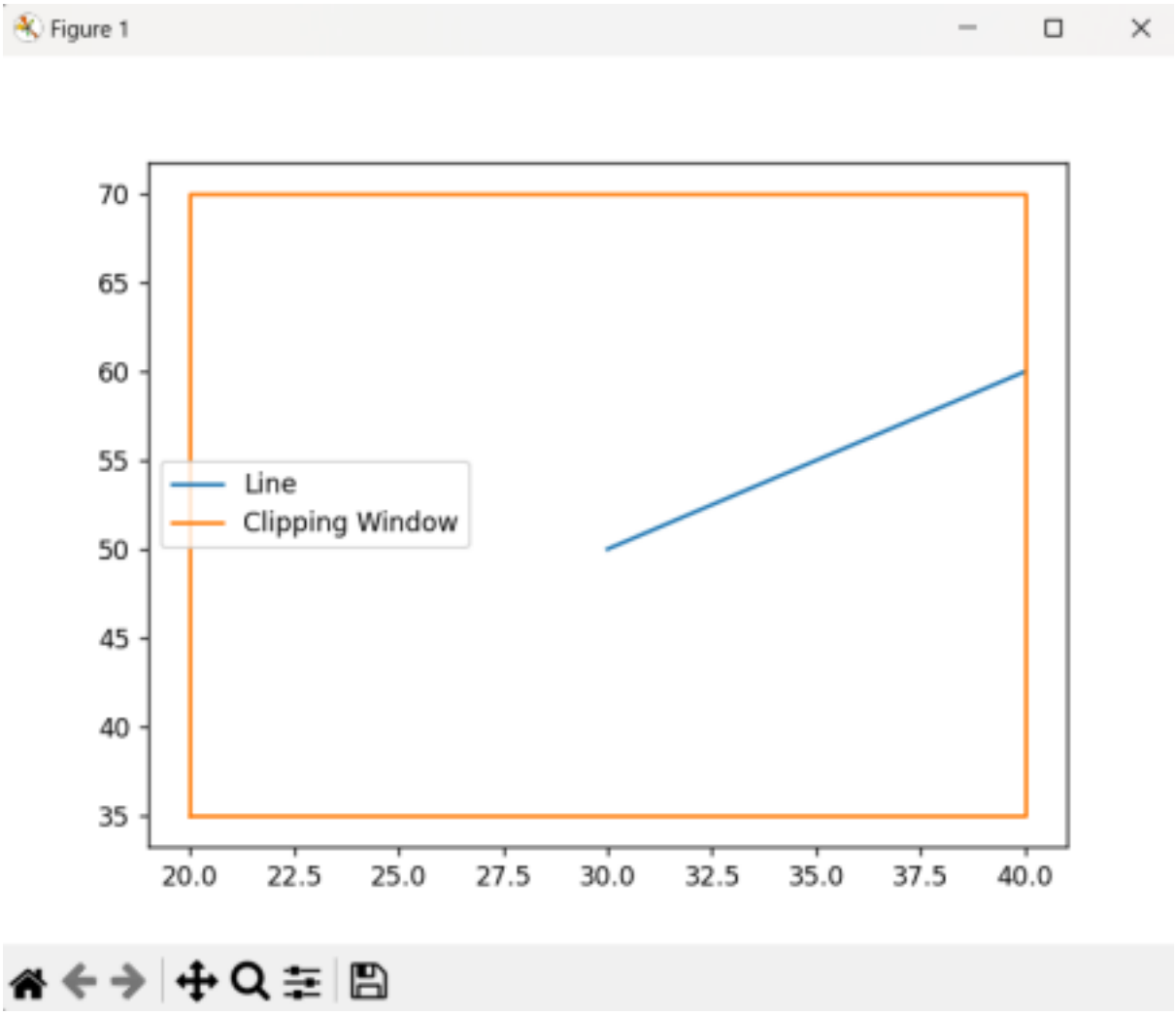
```
72  plt.show()
73  if __name__ == "__main__":
74  x1 = int(input("Enter x1: "))
75  y1 = int(input("Enter y1: "))
76  x2 = int(input("Enter x2: "))
77  y2 = int(input("Enter y2: "))
78  x_min = int(input("Enter x_min: "))
79  y_min = int(input("Enter y_min: "))
80  x_max = int(input("Enter x_max: "))
81  y_max = int(input("Enter y_max: "))
82  result = cohen_sutherland(x1, y1, x2, y2, x_min, y_min, x_max, y_max) if resu
83  print(f"The line from ({x1}, {y1}) to ({x2}, {y2}) clips to  {result}.")
84  plot_line(x1, y1, x2, y2, x_min, y_min, x_max, y_max)
85  plot_line(result[0][0], result[0][1], result[1][0], result[1][1], x_min, y_min, x_m
86  else:
87  print("The line doesn't lie within the rectangle.")
```

OUTPUT:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\kngou\OneDrive\Desktop\CG> python -u "c:\Users\kngou\OneDrive\Desktop\CG\P02.py"
Enter x_centre: 25
Enter y_centre: 30
Enter radius: 35
PS C:\Users\kngou\OneDrive\Desktop\CG> python -u "c:\Users\kngou\OneDrive\Desktop\CG\tempCodeRunnerFile.py"
Enter x1: 30
Enter y1: 50
Enter x2: 40
Enter y2: 60
Enter x_min: 20
Enter y_min: 35
Enter x_max: 40
Enter y_max: 70
The line from (30, 50) to (40, 60) clips to ((30, 50), (40, 60)).
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021



Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

Q4)Write a program to clip a polygon using Sutherland Hodgeman algorithm.

CODE:

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np
import warnings
# POINTS NEED TO BE PRESENTED CLOCKWISE OR ELSE THIS WONT WORK class
def __init__(self,warn_if_empty=True):
self.warn_if_empty = warn_if_empty
def is_inside(self,p1,p2,q):
R = (p2[0] - p1[0]) * (q[1] - p1[1]) - (p2[1] - p1[1]) * (q[0] - p1[0])
if R <= 0:
return True
else:
return False
def compute_intersection(self,p1,p2,p3,p4):
# if first line is vertical
if p2[0] - p1[0] == 0:
x = p1[0]
# slope and intercept of second line
m2 = (p4[1] - p3[1]) / (p4[0] - p3[0])
b2 = p3[1] - m2 * p3[0]
# y-coordinate of intersection
y = m2 * x + b2
# if second line is vertical
elif p4[0] - p3[0] == 0:
x = p3[0]
# slope and intercept of first line
m1 = (p2[1] - p1[1]) / (p2[0] - p1[0])
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

```
28  b1 = p1[1] - m1 * p1[0]
29  # y-coordinate of intersection
30  y = m1 * x + b1
31  # if neither line is vertical
32  else:
33  m1 = (p2[1] - p1[1]) / (p2[0] - p1[0])
34  b1 = p1[1] - m1 * p1[0]
35  # slope and intercept of second line
36  m2 = (p4[1] - p3[1]) / (p4[0] - p3[0])
37  b2 = p3[1] - m2 * p3[0]
38  # x-coordinate of intersection
39  x = (b2 - b1) / (m1 - m2)
40  # y-coordinate of intersection
41  y = m1 * x + b1
42  intersection = (x,y)
43  return intersection
44  def clip(self,subject_polygon,clipping_polygon):
45  final_polygon = subject_polygon.copy()
46  for i in range(len(clipping_polygon)):
47  # stores the vertices of the next iteration of the clipping  procedure
48  next_polygon = final_polygon.copy()
49  # stores the vertices of the final clipped polygon
50  final_polygon = []
51  # these two vertices define a line segment (edge) in the clipping # polygon. It is assu
52  c_edge_start = clipping_polygon[i - 1]
53  c_edge_end = clipping_polygon[i]
54  for j in range(len(next_polygon)):
55  # these two vertices define a line segment (edge) in the
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

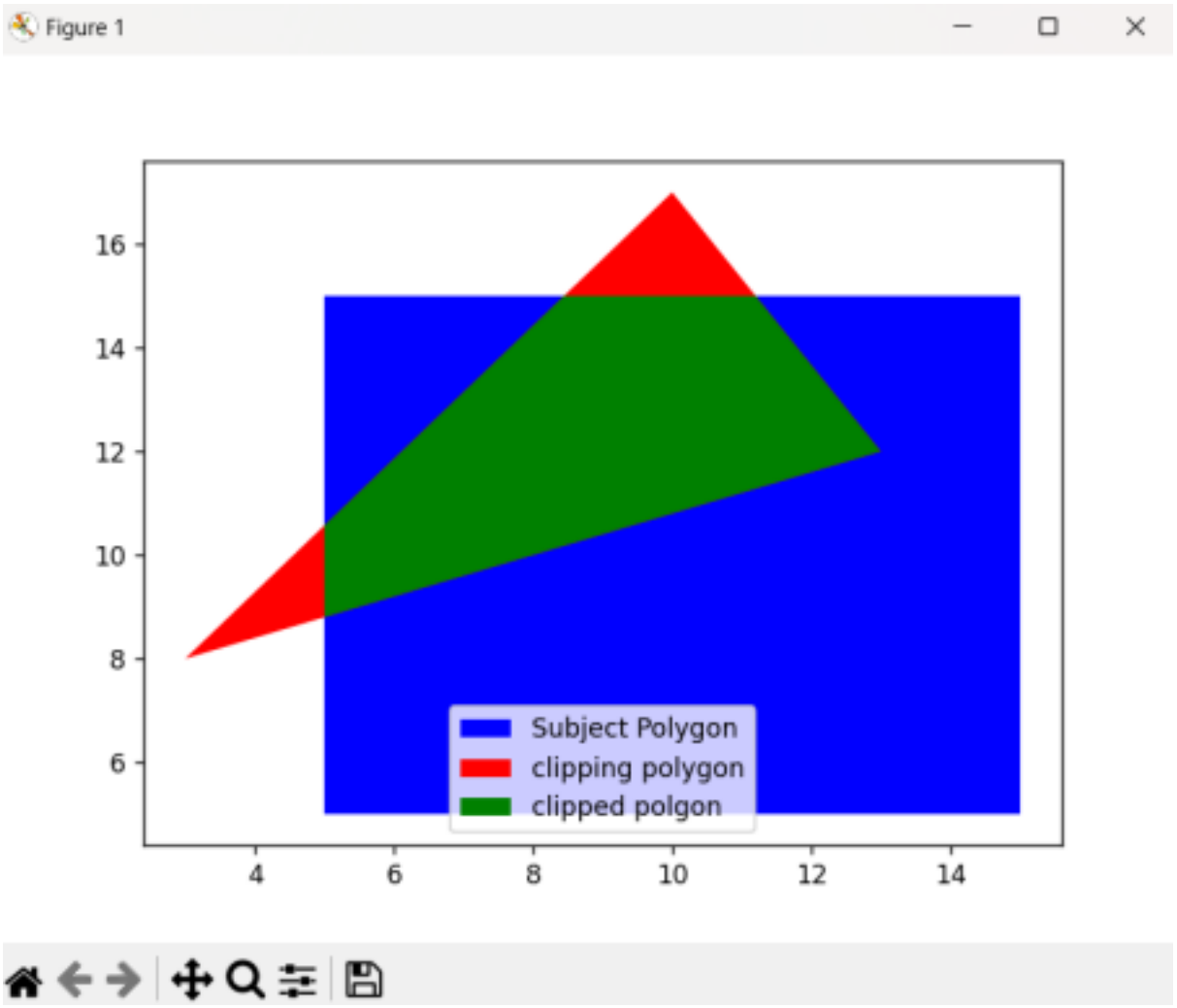
```
# these two vertices define a line segment (edge) in the
subject
# polygon
s_edge_start = next_polygon[j - 1]
s_edge_end = next_polygon[j]
if self.is_inside(c_edge_start,c_edge_end,s_edge_end):
if not
self.is_inside(c_edge_start,c_edge_end,s_edge_start):
intersection =
self.compute_intersection(s_edge_start,s_edge_end,c_edge_start,c_edge_end) fi
final_polygon.append(tuple(s_edge_end))
elif self.is_inside(c_edge_start,c_edge_end,s_edge_start): intersection =
self.compute_intersection(s_edge_start,s_edge_end,c_edge_start,c_edge_end) fi
return np.asarray(final_polygon)
def __call__(self,A,B):
clipped_polygon = self.clip(A,B)
if len(clipped_polygon) == 0 and self.warn_if_empty:
warnings.warn("No intersections found. Are you sure your \ polygon coordinates are
return clipped_polygon
if __name__ == '__main__':
# some test polygons
clip = PolygonClipper()
# subject_polygon = [(0,3),(0.5,0.5),(3,0),(0.5,-0.5),(0,-3),(-0.5,- 0.5),(-3,0),(-0.5,0.5)]
# clipping_polygon = [(-2,-2),(-2,2),(2,2),(2,-2)]
subject_polygon = [(5,5),(15,5),(15,15),(5,15)]
clipping_polygon = [(10,17),(13,12),(3,8)]
# star and triangle
```

```
# star and triangle
# subject_polygon = [(0,3),(0.5,0.5),(3,0),(0.5,-0.5),(0,-3),(-0.5,- 0.5),(-3,0),(-0.5,0.5)]
# clipping_polygon = [(0,2),(2,-2),(-2,-2)]
subject_polygon = np.array(subject_polygon)
clipping_polygon = np.array(clipping_polygon)
clipped_polygon = clip(subject_polygon,clipping_polygon) fig, ax = plt.subplots()
xs, ys = zip(*subject_polygon)# create lists of x and y values plt.fill(xs, ys, 'b',label='Su
xs, ys = zip(*clipping_polygon)# create lists of x and y values plt.fill(xs, ys, 'r',label='cl

xs, ys = zip(*clipped_polygon) # create lists of x and y values plt.fill(xs, ys, 'g',label='c
plt.legend()
plt.show()
print(clipped_polygon)
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

OUTPUT:



Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

Q5)Write a program to fill a polygon using Scan line fill algorithm.

CODE:

```
def draw_graph(V):
x=[]
y=[]
for i in V:
x.append(i[0])
y.append(i[1])
plt.title("Before Appling Scan line Algorithm")
plt.plot(x,y,c="blue",label="Polygon")
plt.show()
plt.plot(x,y,c="blue",label="Polygon")
return max(y),min(y),max(x),min(x)
def get_range(V):
y2,y1,x2,x1=draw_graph(V)
x=[]
y=[]
for i in range(len(V)-1):
x1=V[i][0]
y1=V[i][1]
x2=V[i+1][0]
y2=V[i+1][1]
x3,y3=calpoints(x1,y1,x2,y2,x,y)
x=x3
y=y3
for i in range(int(len(y)/2)):
if i!= int(len(y)/2) and i!= 0:
plt.plot([x[i],x[y.index(y[i],i+1)]],[y[i],y[i]],c="red")
plt.title("After Appling Scan Line algorithm")
```

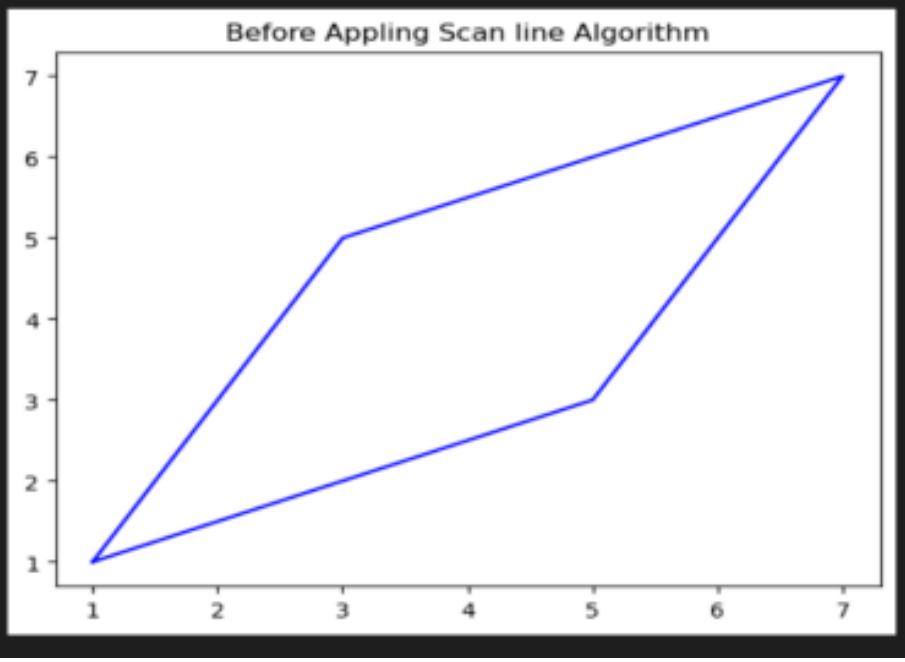
Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

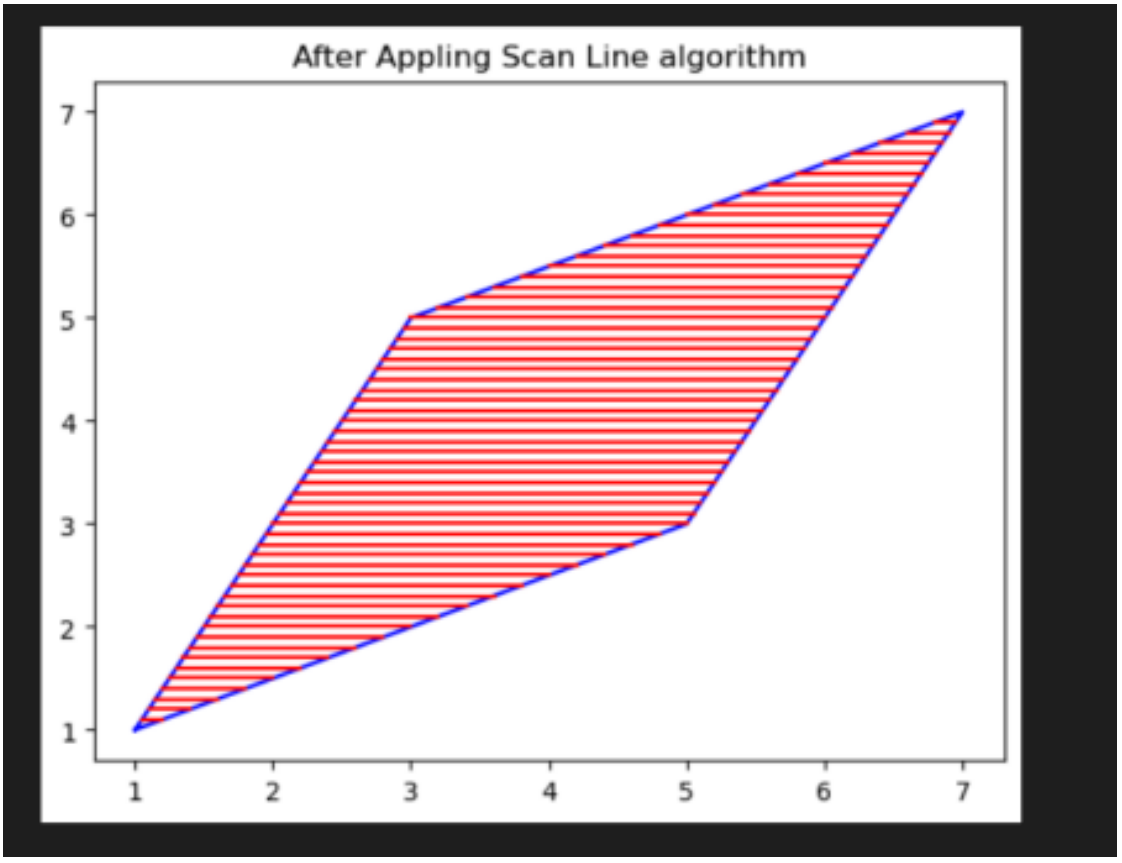
```
27  plt.title("After Appling Scan Line algorithm")
28  plt.show()
29  def calpoints(x1,y1,x2,y2,xlist,ylist):
30  m=(y2-y1)/(x2-x1)
31  c=y1-(m*x1)
32  for i in range(abs(y2-y1)*10):
33  if x1<=x2:
34  y=(i/10)+y1
35  x=(y-c)/m
36  xlist.append(x)
37  ylist.append(y)
38  else:
39  y=y1-(i/10)
40  x=(y-c)/m
41  xlist.append(x)
42  ylist.append(y)
43  # print(xlist,ylist)
44  return xlist,ylist
45  count=int(input("Enter Number of vertices in a Polygon :- ")) V=[]
46  print("Start entering Points :- ")
47  for i in range(count):
48  print("Enter Cordinates of ",i+1," Vertex :- ")
49  x,y=input().split()
50  V.append((int(x),int(y)))
51  V.append((V[0][0],V[0][1]))
52  get_range(V)
53
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

OUTPUT:

```
Enter Number of vertices in a Polygon :- 4
Start entering Points :-
Enter Cordinates of  1  Vertex :-
1 1
Enter Cordinates of  2  Vertex :-
3 5
Enter Cordinates of  3  Vertex :-
7 7
Enter Cordinates of  4  Vertex :-
5 3
```





Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

Q6)Write a program to apply various 2D transformations on a 2D object (use homogenous Coordinates).

CODE:

```
import numpy as np
def translate(point, tx, ty):
transformation_matrix = np.array([[1, 0, tx],
[0, 1, ty],
[0, 0, 1]])
return np.dot(transformation_matrix, point)
def rotate(point, theta):
theta = np.radians(theta)
transformation_matrix = np.array([[np.cos(theta), -np.sin(theta), 0], [np.sin(theta), n
[0, 0, 1]])
return np.dot(transformation_matrix, point)
def scale(point, sx, sy):
transformation_matrix = np.array([[sx, 0, 0],
[0, sy, 0],
[0, 0, 1]])
return np.dot(transformation_matrix, point)
# Example usage:
pointT = np.array([0, 0, 1]) # Homogeneous coordinates
pointR = np.array([4,3,1])
pointS = np.array([2,2,1])
point = translate(pointT, 2, 2)
print("After translation: ", point)
point = rotate(pointR, 45)
print("After rotation: ", point)
point = scale(pointS, 1.5, 0.5)
print("After scaling: ", point)
```

OUTPUT:

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

```
PS C:\Users\kngou\OneDrive\Desktop\CG> python -u "c:\Users\kngou\OneDrive\Desktop\CG\P06.py"
After translation:  [2 2 1]
After rotation:  [0.70710678 4.94974747 1.        ]
After scaling:  [3. 1. 1.]
PS C:\Users\kngou\OneDrive\Desktop\CG>
```

Q7) Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

CODE:

```
import numpy as np
def translate(point, tx, ty, tz):
transformation_matrix = np.array([[1, 0, 0, tx],
[0, 1, 0, ty],
[0, 0, 1, tz],
[0, 0, 0, 1]])
return np.dot(transformation_matrix, point)
def rotate(point, ax, ay, az):
ax, ay, az = np.radians(ax), np.radians(ay), np.radians(az) Rx = np.array([[1, 0, 0, 0],
[0, np.cos(ax), -np.sin(ax), 0],
[0, np.sin(ax), np.cos(ax), 0],
[0, 0, 0, 1]])
Ry = np.array([[np.cos(ay), 0, np.sin(ay), 0],
[0, 1, 0, 0],
[-np.sin(ay), 0, np.cos(ay), 0],
[0, 0, 0, 1]])
Rz = np.array([[np.cos(az), -np.sin(az), 0, 0],
[np.sin(az), np.cos(az), 0, 0],
[0, 0, 1, 0],
[0, 0, 0, 1]])
R = np.dot(Rz, np.dot(Ry, Rx))
return np.dot(R, point)
def scale(point, sx, sy, sz):
transformation_matrix = np.array([[sx, 0, 0, 0],
[0, sy, 0, 0],
[0, 0, sz, 0],
[0, 0, 0, 1]])
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

```
    [0, 0, sz, 0],
    [0, 0, 0, 1]])
    return np.dot(transformation_matrix, point)
    def parallel_project(point):
    projection_matrix = np.array([[1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 1]])
    return np.dot(projection_matrix, point)
    def perspective_project(point, d):
    projection_matrix = np.array([[1, 0, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 1, -1/d],
    [0, 0, 0, 1]])
    return np.dot(projection_matrix, point)
    # Example usage:
    point = np.array([1, 2, 3, 1]) # Homogeneous coordinates
    point = translate(point, 2, 3, 4)
    print("After translation: ", point)
    point = rotate(point, 45, 45, 45)
    print("After rotation: ", point)
    point = scale(point, 2, 2, 2)
    print("After scaling: ", point)
    point = parallel_project(point)
    print("After parallel projection: ", point)
    point = perspective_project(point, 1)
    print("After perspective projection: ", point)
```

OUTPUT:

```
PS C:\Users\kngou\OneDrive\Desktop\CG> python -u "c:\Users\kngou\OneDrive\Desktop\CG\P07.py"
After translation:  [3 5 7 1]
After rotation:  [6.74264069 4.74264069 3.87867966 1.        ]
After scaling:  [13.48528137  9.48528137  7.75735931  1.        ]
After parallel projection:  [13.48528137  9.48528137  0.          1.        ]
After perspective projection:  [13.48528137  9.48528137 -1.          1.        ]
```

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021

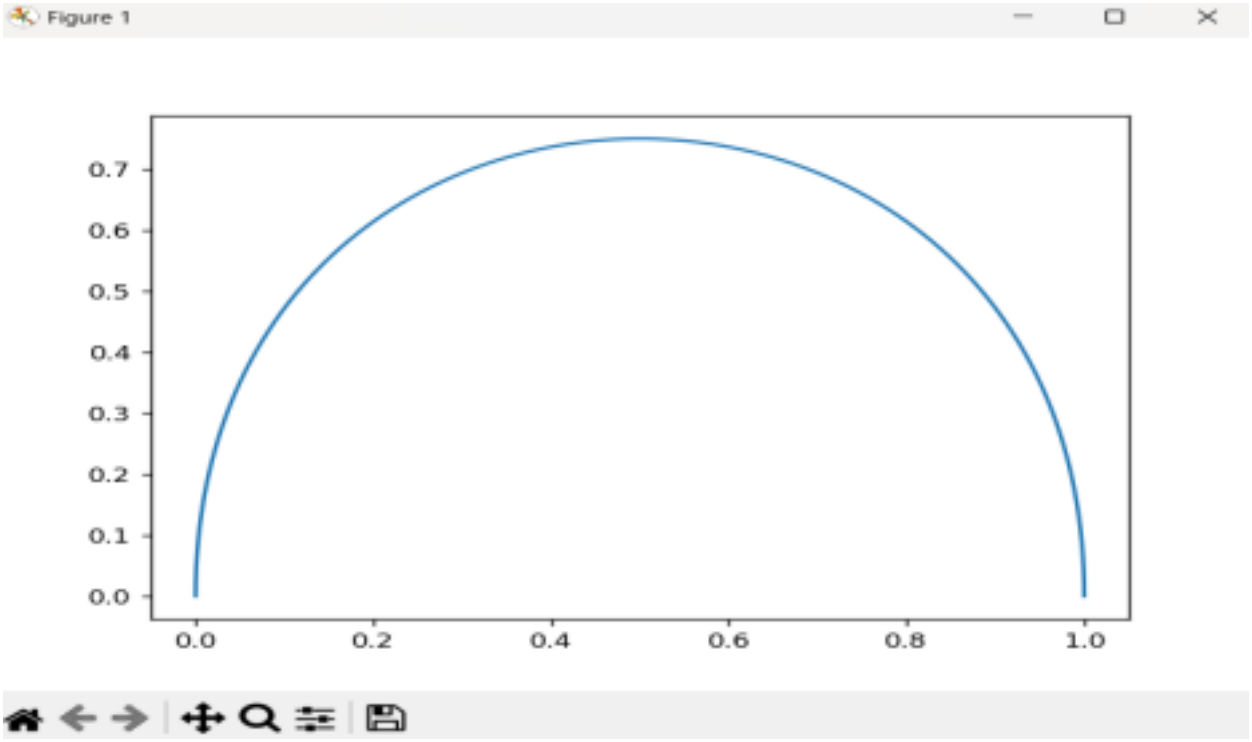
Q8)Write a program to draw Hermite /Bezier curve.

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy as scipy
def bezier_curve(points, t):
n = len(points)
B = np.zeros(2)
for i in range(n):
B += scipy.special.comb(n-1, i) * ((1-t)**(n-1-i)) * (t**i) * points[i]
return B
points = np.array([[0, 0], [0, 1], [1, 1], [1, 0]])
t = np.linspace(0, 1, num=1000)
curve = np.array([bezier_curve(points, i) for i in t])
plt.plot(curve[:, 0], curve[:, 1])
plt.show()
```

OUTPUT:

Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021



Name: Khushi Chhatwani
College Roll no. : CSC/21/55
University Roll no. : 21059570021