

Md: Israil Hosen

Roll: 2010876110

Computer Graphics Lab

August 20, 2025

Experiment-1

Simulate Hidden Surface Elimination or Visual Surface Detection

Python Script

```
1 import turtle
2
3 # Setup
4 screen = turtle.Screen()
5 screen.bgcolor("white")
6 screen.title("Visual Surface
    Detection")
7
8 pen = turtle.Turtle()
9 pen.speed(1)
10 pen.pensize(2)
11 pen.color("black")
12
13 # Draw Main Axes
14 def draw_axes():
15     pen.speed(0)
16
17     # X-axis
18     pen.penup()
19     pen.goto(-300, 0)
20     pen.pendown()
21     pen.goto(300, 0)
22     # Y-axis
23     pen.penup()
24     pen.goto(0, -300)
25     pen.pendown()
26     pen.goto(0, 300)
27
28     pen.speed(1)
29
30 # Draw Triangle
31 def drawTriangle():
32     x = [10, 50, 100]
33     y = [100, 20, 100]
34
35     pen.penup()
36     pen.goto(x[0], y[0])
37     pen.pendown()
38     pen.fillcolor("green")
39     pen.begin_fill()
40     pen.goto(x[1], y[1])
41     pen.goto(x[2], y[2])
42     pen.goto(x[0], y[0])
43     pen.end_fill()
44
45 # Draw Circle
46 def drawCircle():
47     pen.penup()
48     pen.goto(100, 55)
49     pen.pendown()
50     pen.fillcolor("blue")
51     pen.begin_fill()
52     pen.circle(45)
53     pen.end_fill()
54
55 # Draw Rectangle
56 def drawRectangle():
57     x1, y1 = 100, 100
58     x2, y2 = 180, 180
59
60     pen.penup()
61     pen.goto(x1, y1)
62     pen.pendown()
63     pen.fillcolor("red")
64     pen.begin_fill()
65     pen.goto(x2, y1)
66     pen.goto(x2, y2)
67     pen.goto(x1, y2)
68     pen.goto(x1, y1)
69     pen.end_fill()
70
71 #-----
72 # Main Function
73 #-----
74 # Sequence of drawing
75 draw_axes()
76 sequence = "RCT"
77 for shape in sequence:
78     if shape == "C":
79         drawCircle()
80     elif shape == "T":
81         drawTriangle()
82     else:
83         drawRectangle()
84
85 pen.hideturtle()
86 turtle.done()
```

Output

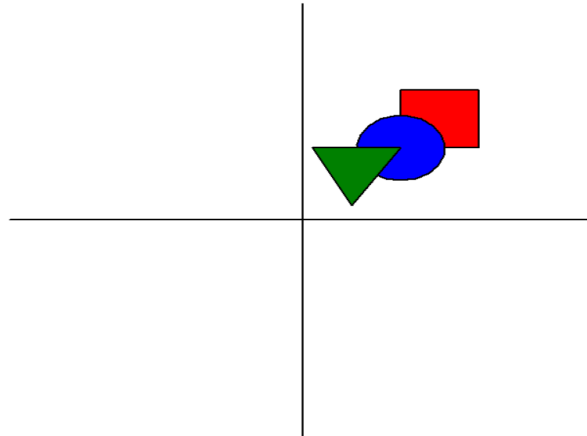


Figure 1: Hidden Surface Elimination

Experiment-2

Implement the Cohen Sutherland Line Clipping Algorithm

Python Script

```
1 import turtle
2 # Clipping Window Parameters
3 x_left, x_right = -100, 300
4 y_bottom, y_top = -50, 200
5 # Region Codes
6 Left, Right, Bottom, Top = 1, 2, 4, 8
7
8 # Draw axes
9 def draw_axes(pen, width, height):
10     pen.penup()
11     pen.goto(-width / 2, 0)
12     pen.pendown()
13     pen.goto(width / 2, 0)
14     pen.write("X", align="center",
15             font=("Arial", 12, "normal"))
16     pen.penup()
17     pen.goto(0, -height / 2)
18     pen.pendown()
19     pen.goto(0, height / 2)
20     pen.write("Y", align="center",
21             font=("Arial", 12, "normal"))
22     pen.penup()
23 # Function to calculate region code
24 def regionCode(x, y):
25     code = 0
26     if x > x_right:
27         code |= Right
28     elif x < x_left:
29         code |= Left
30     if y > y_top:
31         code |= Top
32     elif y < y_bottom:
33         code |= Bottom
34     return code
35
36 # Cohen-Sutherland Line Clipping
37 # Algorithm
38 def cohenSutherland(x1, y1, x2, y2,
39                     pen):
40     code1 = regionCode(x1, y1)
41     code2 = regionCode(x2, y2)
42     while True:
43         if not (code1 | code2): #
44             # Line completely inside
45             drawLine(x1, y1, x2, y2
46                     , "green", pen)
47             return
48         elif code1 & code2: # Line
49             # completely outside
50             return
51         else: # Line partially
52             # inside
```

```

47         code = code1 if code1
else code2
48
49         if code & Top:
50             y = y_top
51             x = x1 + (x2 - x1)
* (y - y1) / (y2 - y1)
52         elif code & Bottom:
53             y = y_bottom
54             x = x1 + (x2 - x1)
* (y - y1) / (y2 - y1)
55         elif code & Left:
56             x = x_left
57             y = y1 + (y2 - y1)
* (x - x1) / (x2 - x1)
58         elif code & Right:
59             x = x_right
60             y = y1 + (y2 - y1)
* (x - x1) / (x2 - x1)
61
62         if code == code1:
63             x1, y1 = x, y
64             code1 = regionCode(
x1, y1)
65         else:
66             x2, y2 = x, y
67             code2 = regionCode(
x2, y2)
68
69 # Draw a line helper
70 def drawLine(x1, y1, x2, y2, color, pen):
71     pen.penup()
72     pen.goto(x1, y1)
73     pen.pendown()
74     pen.pencolor(color)
75     pen.goto(x2, y2)
76
77 #-----
78 # Main Function
79 #-----
80 # screen setup
81 WIDTH, HEIGHT = 800, 600
82 screen = turtle.Screen()
83 screen.title("Cohen-Sutherland Line
Clipping")
84 screen.setup(width=WIDTH, height=
HEIGHT)
85 screen.bgcolor("white")
86
87 # pen setup
88 pen = turtle.Turtle()
89 pen.speed(2)
90 pen.pensize(2)
91 pen.pencolor("Black")
92 draw_axes(pen, WIDTH, HEIGHT)
93
94 # Draw the clipping rectangle
95 drawLine(x_left, y_bottom, x_right,
y_bottom, "green", pen)
96 drawLine(x_right, y_bottom, x_right
, y_top, "green", pen)
97 drawLine(x_right, y_top, x_left,
y_top, "green", pen)
98 drawLine(x_left, y_top, x_left,
y_bottom, "green", pen)
99
100 # Original line
101 x1, y1, x2, y2 = -180, -30, 300,
300
102 drawLine(x1, y1, x2, y2, "red", pen
)
103
104 # Clipped line
105 cohenSutherland(x1, y1, x2, y2, pen
)
106
107 pen.hideturtle()
108 turtle.done()

```

Output

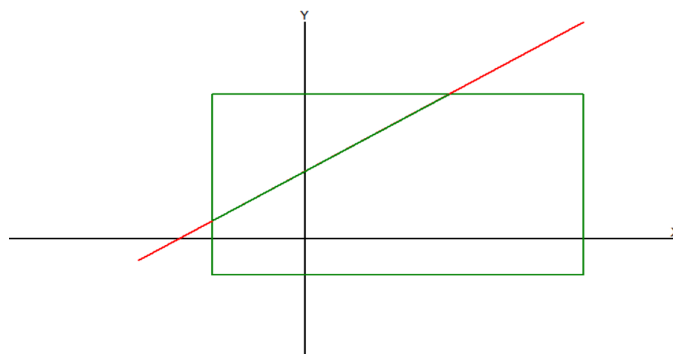


Figure 2: Cohen Sutherland Line Clipping

Experiment-3

Implement the Sutherland-Hodgman Polygon Clipping Algorithm

Python Script

```
1 import turtle
2
3 # Point structure equivalent
4 class Point:
5     def __init__(self, x, y):
6         self.x = x
7         self.y = y
8
9 # Clipping window
10 wMin = Point(100, 100)
11 wMax = Point(300, 250)
12
13 # Clipping edge constants
14 LEFT, RIGHT, BOTTOM, TOP = 0, 1, 2, 3
15
16 # Check if point is inside a clip
    edge
17 def inside(p, edge):
18     if edge == LEFT:
19         return p.x >= wMin.x
20     elif edge == RIGHT:
21         return p.x <= wMax.x
22     elif edge == BOTTOM:
23         return p.y >= wMin.y
24     elif edge == TOP:
25         return p.y <= wMax.y
26
27
28 # Intersection with clip edge
29 def intersect(p1, p2, edge):
30     m = (p2.y - p1.y) / (p2.x - p1
31     x) if p1.x != p2.x else 1e9
32     if edge == 0:
33         x = wMin.x
34         y = p1.y + (wMin.x - p1.x)
35         * m
36     elif edge == 1:
37         x = wMax.x
38         y = p1.y + (wMax.x - p1.x)
39         * m
40     elif edge == 2:
41         y = wMin.y
42         x = p1.x + (wMin.y - p1.y)
43         / m if m != 0 else p1.x
44     else:
45         y = wMax.y
46         x = p1.x + (wMax.y - p1.y)
47         / m if m != 0 else p1.x
48     return Point(x, y)
49
50 # Clip polygon against one edge
51 def clip_polygon(points, edge):
52     clipped = []
53     for i in range(len(points)):
54         curr = points[i]
55         prev = points[i - 1]
56         curr_in = inside(curr, edge)
57         prev_in = inside(prev, edge)
58
59         if prev_in and curr_in:
60             clipped.append(curr)
61         elif not prev_in and
62             curr_in:
63             clipped.append(
64                 intersect(prev, curr, edge))
65             clipped.append(curr)
66         elif prev_in and not
67             curr_in:
68             clipped.append(
69                 intersect(prev, curr, edge))
70         return clipped
71
72 # Drawing helpers
73 def draw_axes(pen, width, height):
74     pen.penup()
75     pen.goto(-width / 2, 0)
76     pen.pendown()
77     pen.goto(width / 2, 0)
78     pen.write("X", align="center",
79             font=("Arial", 12, "normal"))
80
81     pen.penup()
82     pen.goto(0, -height / 2)
83     pen.pendown()
84     pen.goto(0, height / 2)
85     pen.write("Y", align="center",
86             font=("Arial", 12, "normal"))
87     pen.penup()
88
89 def draw_polygon(points, color):
90     if not points:
91         return
92     pen.pencolor(color)
93     pen.penup()
94     pen.goto(points[0].x, points
95             [0].y)
96     pen.pendown()
97     for p in points[1:]:
98         pen.goto(p.x, p.y)
```

```

87     pen.goto(points[0].x, points
114 pen.pencolor("Black")
115
116 # Define star polygon
117 star = [
118     Point(200, 300), Point(250,
220), Point(330, 220), Point
(260, 170),
119     Point(300, 80), Point(200, 130)
, Point(100, 80), Point(140,
170),
120     Point(70, 220), Point(150, 220)
121 ]
122
123 # Draw scene
124 draw_axes(pen, WIDTH, HEIGHT)
125 draw_clip_window(pen)
126 draw_polygon(star, "black")
127
128 # Step-by-step clipping
129 edges = [LEFT, RIGHT, BOTTOM, TOP]
130 colors = ["red", "blue", "cyan", "
green"]
131
132 for i, edge in enumerate(edges):
133     screen.update()
134     star = clip_polygon(star, edge)
135     draw_polygon(star, colors[i])
136
137 pen.hideturtle()
138 turtle.done()

```

Output

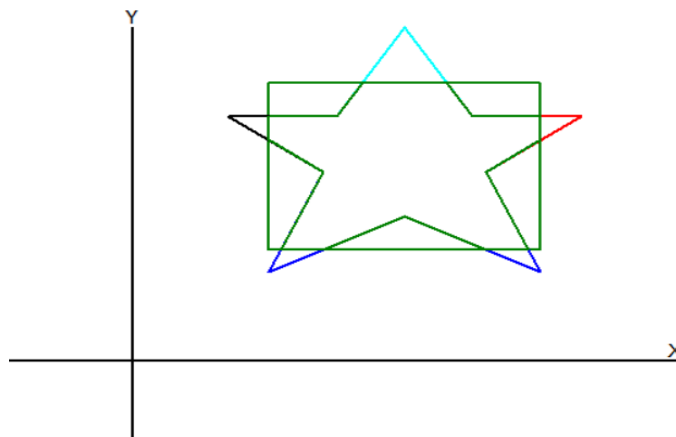


Figure 3: Sutherland Hodgman Polygon Clipping

Experiment-4

Create the Bezier Curve

Python Script

```
1 import turtle
2 import math
3
4 # ===== Math
5 def factorial(n):
6     if n < 2:
7         return 1
8     return n * factorial(n - 1)
9
10 def nCr(n, r):
11     return factorial(n) / (
12         factorial(r) * factorial(n - r)
13     )
14
15 # Bernstein ploynomial
16 def bezier_basis(k, n, u):
17     return nCr(n, k) * (u ** k) *
18     ((1 - u) ** (n - k))
19
20 # ===== Bezier
21 # Curve Function
22
23 def bezier_curve(pen, points, steps
24 =1000):
25     n = len(points) - 1
26     pen.pencolor("green")
27
28     # Draw curve
29     pen.penup()
30     for i in range(steps + 1):
31         u = i / steps
32         x, y = 0, 0
33         for k in range(n + 1):
34             b = bezier_basis(k, n,
35 u)
36             x += points[k][0] * b
37             y += points[k][1] * b
38         if i == 0:
39             pen.goto(x, y)
40             pen.pendown()
41         else:
42             pen.goto(x, y)
43
44     # Draw control points
45     pen.pencolor("red")
46     pen.penup()
47     for (x, y) in points:
48         pen.goto(x, y-3)
49         pen.pendown()
50         pen.circle(3)
51
52     pen.penup()
53
54     # Draw control polygon
55     pen.pencolor("gray")
56     pen.penup()
57     pen.goto(points[0])
58     pen.pendown()
59     for (x, y) in points[1:]:
60         pen.goto(x, y)
61
62     # Draw axes
63     def draw_axes(pen, width, height):
64         pen.penup()
65         pen.goto(-width / 2, 0)
66         pen.pendown()
67         pen.goto(width / 2, 0)
68         pen.write("X")
69
70         pen.penup()
71         pen.goto(0, -height / 2)
72         pen.pendown()
73         pen.goto(0, height / 2)
74         pen.write("Y")
75
76     pen.penup()
77
78     # ===== Main
79     =====
80     # screen stup
81     WIDTH, HEIGHT = 800, 600
82     screen = turtle.Screen()
83     screen.title("Bezier Curve using
84 Python Turtle")
85     screen.setup(width=WIDTH, height=
86 HEIGHT)
87     screen.bgcolor("white")
88     screen.tracer(0)
89
90     # pen setup
91     pen = turtle.Turtle()
92     pen.speed(0)
93     pen.pensize(2)
94     pen.pencolor("Black")
95     draw_axes(pen, WIDTH, HEIGHT)
96
97     # control points
98     control_points = [(27, 243), (101,
99 47), (324, 197), (437, 23)]
100
101     # Draw bezier curve
102     bezier_curve(pen, control_points,
103 steps=1000)
```

```

90
91 pen.hideturtle()
92 screen.update()
93 turtle.done()

```

Output

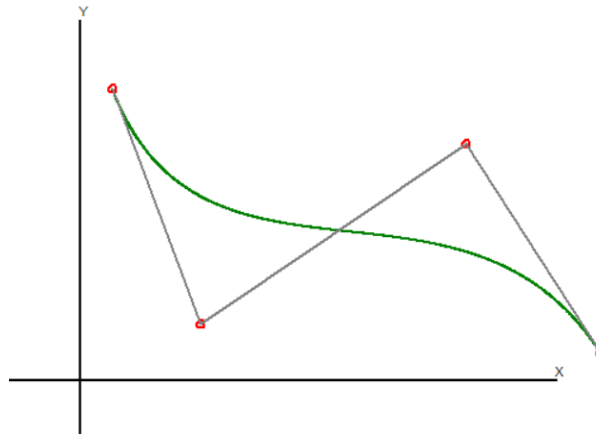


Figure 4: Bezier curve

Experiment-5

Simulate 2D Geometric Translation

Python Script

```

1 import turtle
2
3 # =====
4 # Helper Functions
5 # =====
6
7 def draw_triangle(pen, triangle,
8                 color, label=None):
9     pen.pencolor(color)
10    pen.penup()
11    pen.goto(triangle[0][0],
12            triangle[0][1])
13
14    pen.pendown()
15    # Draw the triangle edges
16    pen.goto(triangle[1][0],
17            triangle[1][1])
18    pen.goto(triangle[2][0],
19            triangle[2][1])
20    pen.goto(triangle[0][0],
21            triangle[0][1])
22    pen.penup()
23
24    # Draw label if provided
25
26    if label:
27        pen.goto(triangle[0][0] -
28                20, triangle[0][1] + 20)
29        pen.write(label)
30
31 def draw_axes(pen, width, height):
32     # X-axis
33     pen.penup()
34     pen.goto(-width / 2, 0)
35     pen.pendown()
36     pen.goto(width / 2, 0)
37     pen.write("X")
38
39     # Y-axis
40     pen.penup()
41     pen.goto(0, -height / 2)
42     pen.pendown()
43     pen.goto(0, height / 2)
44     pen.write("Y")
45
46     pen.penup()
47
48 # =====
49 # Main Program

```



```

43 # =====
44
45 # Screen setup
46 WIDTH, HEIGHT = 800, 600
47 screen = turtle.Screen()
48 screen.title("2D Triangle
    Translation with Turtle")
49 screen.setup(width=WIDTH, height=
    HEIGHT)
50 screen.bgcolor("white")
51
52 pen = turtle.Turtle()
53 pen.speed(0)
54 pen.pensize(2)
55 pen.pencolor("black")
56
57 # Draw axes
58 draw_axes(pen, WIDTH, HEIGHT)
59
60 # Define triangle vertices
61 original_triangle = [(50, 50),
    (150, 50), (100, 120)]
62
63 # Translation factors
64 tx, ty = (150, 100)
65
66 # Compute translated triangle
67 translated_triangle = []
68 for point in original_triangle:
69     x, y = point
70     new_x = x + tx
71     new_y = y + ty
72     translated_triangle.append((
        new_x, new_y))
73
74 # Draw original and translated
    triangles
75 draw_triangle(pen,
    original_triangle, "black", "
    Original")
76 draw_triangle(pen,
    translated_triangle, "green", "
    Translated")
77
78 pen.hideturtle()
79 turtle.done()

```

Output

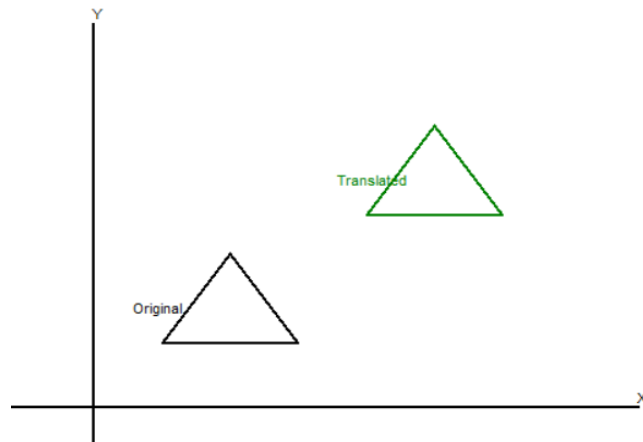


Figure 5: Geometric Translation

Experiment-5

Simulate 2D Geometric Rotation

Python Script

```

1 import turtle
2 import math
3
4 # =====
5 # Helper Functions
6 # =====

```

```

7 def draw_triangle(pen, triangle,
12     color, label=None):
13     pen.pencolor(color)
14     pen.penup()
15     pen.goto(triangle[0][0],
16             triangle[0][1])
17
18     pen.pendown()
19     # Draw the triangle edges
20     pen.goto(triangle[1][0],
21             triangle[1][1])
22     pen.goto(triangle[2][0],
23             triangle[2][1])
24     pen.goto(triangle[0][0],
25             triangle[0][1])
26     pen.penup()
27
28     # Draw label if provided
29     if label:
30         pen.goto(triangle[0][0] -
31                 20, triangle[0][1] + 20)
32         pen.write(label)
33
34 def draw_axes(pen, width, height):
35     # X-axis
36     pen.penup()
37     pen.goto(-width / 2, 0)
38     pen.pendown()
39     pen.goto(width / 2, 0)
40     pen.write("X")
41
42     # Y-axis
43     pen.penup()
44     pen.goto(0, -height / 2)
45     pen.pendown()
46     pen.goto(0, height / 2)
47     pen.write("Y")
48
49     pen.penup()
50
51 def rotate_triangle(triangle,
52                     angle_deg, pivot):
53     xp, yp = pivot
54     angle_rad = math.radians(
55         angle_deg)
56     cos_theta = math.cos(angle_rad)
57     sin_theta = math.sin(angle_rad)
58
59     rotated = []
60     for x, y in triangle:
61         # Translate(shift) point
62         # relative to pivot
63         x_shift = x - xp
64         y_shift = y - yp
65         # Apply rotation
66
67         new_x = xp + (x_shift *
68                     cos_theta - y_shift * sin_theta)
69
70         new_y = yp + (x_shift *
71                     sin_theta + y_shift * cos_theta)
72
73         rotated.append((new_x,
74                         new_y))
75
76     return rotated
77
78 # =====
79 # Main Program
80 # =====
81
82 # Screen setup
83 WIDTH, HEIGHT = 800, 600
84 screen = turtle.Screen()
85 screen.title("2D Triangle Rotation
86             with Turtle")
87 screen.setup(width=WIDTH, height=
88             HEIGHT)
89 screen.bgcolor("white")
90
91 pen = turtle.Turtle()
92 pen.speed(0)
93 pen.pensize(2)
94 pen.pencolor("black")
95
96 # Draw axes
97 draw_axes(pen, WIDTH, HEIGHT)
98
99 # Define triangle vertices
100 original_triangle = [(50, 50),
101                     (150, 50), (100, 120)]
102
103 # Rotation settings
104 angle = 45
105 pivot_point = (100, 50)
106
107 # Compute rotated triangle
108 rotated_triangle = rotate_triangle(
109     original_triangle, angle,
110     pivot_point)
111
112 # Draw original and rotated
113 # triangles
114 draw_triangle(pen,
115               original_triangle, "black", "
116               Original")
117 draw_triangle(pen, rotated_triangle
118             , "red", "Rotated")
119
120 pen.hideturtle()
121 turtle.done()

```

Output

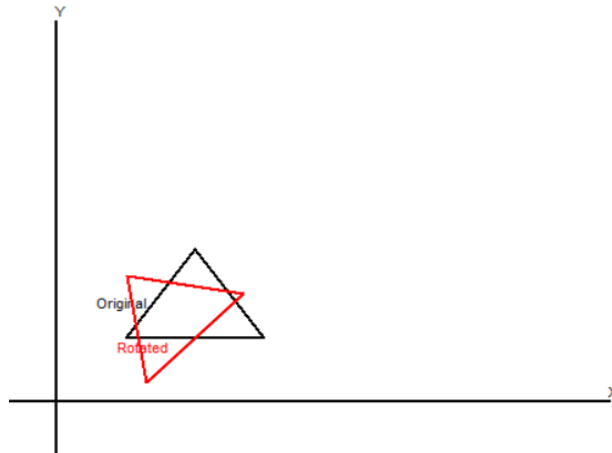


Figure 6: Geometric Rotation

Experiment-5

Simulate 2D Geometric Scaling

Python Script

```
1 import turtle
2
3 # =====
4 # Helper Functions
5 # =====
6 def draw_triangle(pen, triangle,
7                  color, label=None):
8     pen.pencolor(color)
9     pen.penup()
10    pen.goto(triangle[0][0],
11            triangle[0][1])
12
13    pen.pendown()
14    # Draw the triangle edges
15    pen.goto(triangle[1][0],
16            triangle[1][1])
17    pen.goto(triangle[2][0],
18            triangle[2][1])
19    pen.goto(triangle[0][0],
20            triangle[0][1])
21    pen.penup()
22
23    # Draw label if provided
24    if label:
25        pen.goto(triangle[0][0] -
26                20, triangle[0][1] + 20)
27        pen.write(label)
28
29 def draw_axes(pen, width, height):
30     # X-axis
31     pen.penup()
32     pen.goto(-width / 2, 0)
33     pen.pendown()
34     pen.goto(width / 2, 0)
35     pen.write("X")
36
37     # Y-axis
38     pen.penup()
39     pen.goto(0, -height / 2)
40     pen.pendown()
41     pen.goto(0, height / 2)
42     pen.write("Y")
43
44     pen.penup()
45
46 def scale_triangle(triangle, sx, sy,
47                  pivot):
48     xp, yp = pivot
49     scaled = []
50     for x, y in triangle:
51         # Translate point relative
52         # to pivot
53         x_shift = x - xp
54         y_shift = y - yp
55         # Apply scaling
56         new_x = xp + x_shift * sx
```

```

49         new_y = yp + y_shift * sy
50         scaled.append((new_x, new_y))
51     ))
52     return scaled
53 # =====
54 # Main Program
55 # =====
56
57 # Screen setup
58 WIDTH, HEIGHT = 800, 600
59 screen = turtle.Screen()
60 screen.title("2D Triangle Scaling
    with Turtle")
61 screen.setup(width=WIDTH, height=
    HEIGHT)
62 screen.bgcolor("white")
63
64 pen = turtle.Turtle()
65 pen.speed(0)
66 pen.pensize(2)
67 pen.pencolor("black")
68
69 # Draw axes
70 draw_axes(pen, WIDTH, HEIGHT)
71
72 # Define triangle vertices
73 original_triangle = [(50, 50),
74                     (150, 50), (100, 120)]
75
76 # Scaling settings
77 sx, sy = 1.5, 0.8
78 pivot_point = (100, 50)
79
80 # Compute scaled triangle
81 scaled_triangle = scale_triangle(
82     original_triangle, sx, sy,
83     pivot_point)
84
85 # Draw original and scaled
86 # triangles
87 draw_triangle(pen,
88               original_triangle, "black")
89 draw_triangle(pen, scaled_triangle,
90               "green")
91
92 pen.hideturtle()
93 turtle.done()

```

Output

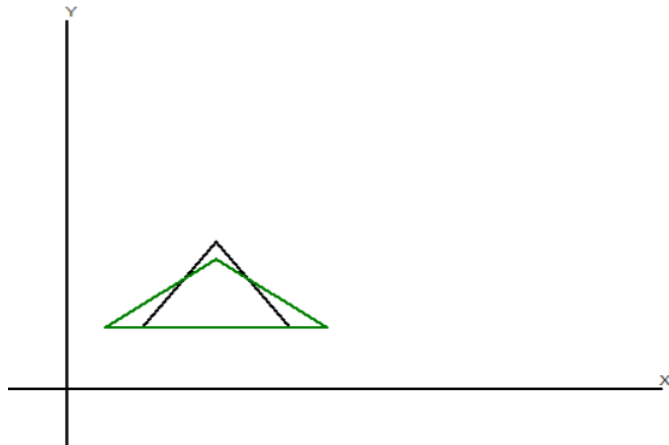


Figure 7: Geometric Scaling

Experiment-6

Draw a Line with Bresenham Line Drawing Algorithm

Python Script

```
1 import turtle
2
3 # =====
4 # Helper Functions
5 # =====
6 def draw_pixel(pen, x, y, color="
black"):
7     pen.penup()
8     pen.goto(x, y)
9     pen.dot(4, color)
10    pen.pendown()
11
12 def bresenham_line(pen, x1, y1, x2, y2, color="black"):
13     # Ensure left-to-right drawing
14     if x1 > x2:
15         x1, x2 = x2, x1
16         y1, y2 = y2, y1
17     dx = x2 - x1
18     dy = abs(y2 - y1)
19     p = 2 * dy - dx
20
21     y = y1
22     for x in range(x1, x2 + 1):
23         draw_pixel(pen, x, y, color)
24         if p >= 0:
25             p += 2 * (dy - dx)
26             y += 1 if y1 < y2 else
-1
27         else:
28             p += 2 * dy
29
30 def draw_axes(pen, width, height):
31     pen.pencolor("gray")
32     # X-axis
33     pen.penup()
34     pen.goto(-width//2, 0)
35     pen.pendown()
36     pen.goto(width//2, 0)
37     pen.write("X", align="center",
font=("Arial", 12, "normal"))
38
39     # Y-axis
40     pen.penup()
41     pen.goto(0, -height//2)
42     pen.pendown()
43     pen.goto(0, height//2)
44     pen.write("Y", align="center",
font=("Arial", 12, "normal"))
45     pen.penup()
46
47 # =====
48 # Main Program
49 # =====
50
51 # screen setup
52 WIDTH, HEIGHT = 800, 600
53 screen = turtle.Screen()
54 screen.title("Bresenham Line
Drawing with Turtle")
55 screen.setup(width=WIDTH, height=
HEIGHT)
56 screen.bgcolor("white")
57 screen.tracer(0)
58
59 # pen setup
60 pen = turtle.Turtle()
61 pen.speed(0)
62 pen.pensize(2)
63 pen.pencolor("black")
64
65 # Draw axes for reference
66 draw_axes(pen, WIDTH, HEIGHT)
67
68 # Draw lines using Bresenham
algorithm
69 lines = [
70     ((-200, -100), (200, 100)),
71     ((-200, 100), (200, -100))
72 ]
73
74
75 for (start, end) in lines:
76     x0, y0 = start
77     x1, y1 = end
78     bresenham_line(pen, x0, y0, x1,
y1, color="blue")
79
80 pen.hideturtle()
81 turtle.done()
```

Output

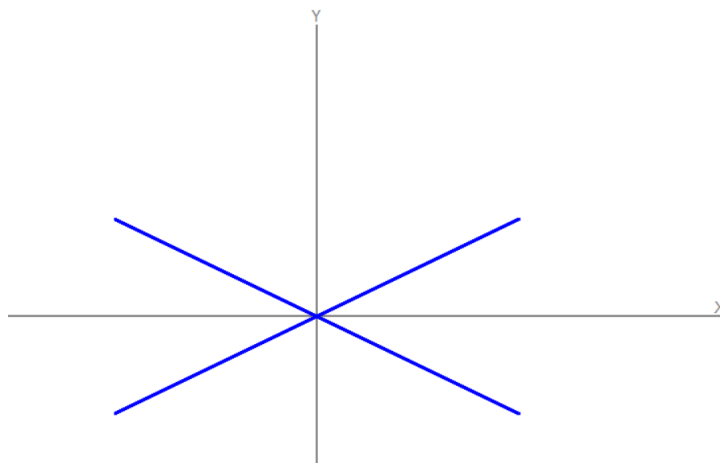


Figure 8: Bresenham Line Drawing

Experiment-7

Draw a Circle with Bresenham Circle Drawing Algorithm

Python Script

```
1 import turtle
2
3 # =====
4 # Helper Functions
5 # =====
6 def draw_pixel(pen, x, y, color="black"):
7     pen.penup()
8     pen.goto(x, y)
9     pen.dot(4, color)
10    pen.pendown()
11
12 def bresenham_circle(pen, xc, yc, r, color="black"):
13     x = 0
14     y = r
15     p = 3 - 2 * r
16
17     def plot_circle_points(xc, yc, x, y):
18         # Draw all 8 symmetric points
19         draw_pixel(pen, xc + x, yc + y, color)
20         draw_pixel(pen, xc - x, yc + y, color)
21         draw_pixel(pen, xc + x, yc - y, color)
22         draw_pixel(pen, xc - x, yc - y, color)
23         draw_pixel(pen, xc + y, yc + x, color)
24         draw_pixel(pen, xc - y, yc + x, color)
25         draw_pixel(pen, xc + y, yc - x, color)
26         draw_pixel(pen, xc - y, yc - x, color)
27
28     plot_circle_points(xc, yc, x, y)
29
30     while x < y:
31         x += 1
32         if p < 0:
33             p += 4 * x + 6
34         else:
35             y -= 1
36             p += 4 * (x - y) + 10
37         plot_circle_points(xc, yc, x, y)
38
39 def draw_axes(pen, width, height):
40     pen.pencolor("gray")
41     # X-axis
42     pen.penup()
```

```

43     pen.goto(-width//2, 0)
44     pen.pendown()
45     pen.goto(width//2, 0)
46     pen.write("X", align="center",
font=("Arial", 12, "normal"))
47
48     # Y-axis
49     pen.penup()
50     pen.goto(0, -height//2)
51     pen.pendown()
52     pen.goto(0, height//2)
53     pen.write("Y", align="center",
font=("Arial", 12, "normal"))
54     pen.penup()
55
56     # =====
57     # Main Program
58     # =====
59
60     #setup screen
61     WIDTH, HEIGHT = 800, 600
62     screen = turtle.Screen()
63     screen.title("Bresenham Circle
Drawing with Turtle")
64     screen.setup(width=WIDTH, height=
HEIGHT)
65     screen.bgcolor("white")
66     screen.tracer(0)
67
68     # setup pen
69     pen = turtle.Turtle()
70     pen.speed(0)
71     pen.pensize(2)
72     pen.pencolor("black")
73
74     draw_axes(pen, WIDTH, HEIGHT)
75
76     # Example circles: (center_x,
center_y, radius)
77     circles = [(0, 0, 100), (-150, 50,
50), (200, -100, 75)]
78
79     for xc, yc, r in circles:
80         bresenham_circle(pen, xc, yc, r
, color="green")
81
82     pen.hideturtle()
83     turtle.done()

```

Output

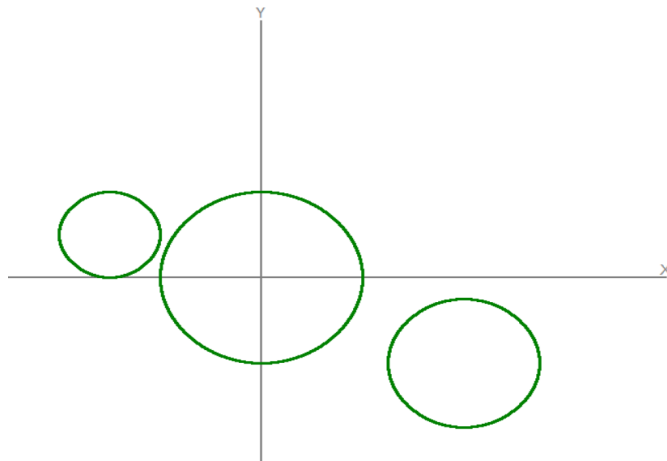


Figure 9: Bresenham Circle Drawing

Experiment-8

Draw the Snowflake Pattern with Fractal Geometry

Python Script

```
1 import turtle
2 import time
3 import math
4
5 # =====
6 # snowflake pattern Draw Function
7 # =====
8 def draw_snowflake_pattern(pen,
    start, end, depth):
9     if depth == 0:
10         pen.penup()
11         pen.goto(start)
12         pen.pendown()
13         pen.goto(end)
14         time.sleep(0.01)
15         screen.update()
16         return
17
18 # Divide the segment into three
    equal parts
19 dx = (end[0] - start[0]) / 3
20 dy = (end[1] - start[1]) / 3
21
22 # First point at one-third
23 p1 = (start[0] + dx, start[1] +
    dy)
24
25 # Third point at two-thirds
26 p3 = (start[0] + 2 * dx, start
    [1] + 2 * dy)
27
28 # Calculate the peak of the
    equilateral triangle
29 px = p1[0] + (p3[0] - p1[0]) /
    2 + math.sqrt(3) * (p3[1] - p1
    [1]) / 2
30 py = p1[1] + (p3[1] - p1[1]) /
    2 - math.sqrt(3) * (p3[0] - p1
    [0]) / 2
31 p2 = (px, py)
32
33 # Recursively draw the four
    segments
34 draw_snowflake_pattern(pen,
    start, p1, depth - 1)
35 draw_snowflake_pattern(pen, p1,
    p2, depth - 1)
36 draw_snowflake_pattern(pen, p2,
    p3, depth - 1)
37 draw_snowflake_pattern(pen, p3,
    end, depth - 1)
38
39 # =====
40 # Main program
41 # =====
42
43 #setup the screen
44 WIDTH, HEIGHT = 800, 600
45 screen = turtle.Screen()
46 screen.title("Snowflake Pattern
    with Turtle")
47 screen.setup(width=WIDTH, height=
    HEIGHT)
48 screen.bgcolor("white")
49 screen.tracer(0)
50
51 # setup the pen
52 pen = turtle.Turtle()
53 pen.hideturtle()
54 pen.pensize(3)
55 pen.pencolor("green")
56 pen.speed(0)
57
58 # Define Triangle Points
59 p0 = (0, 250)
60 p1 = (-200, -100)
61 p2 = (200, -100)
62
63 # Draw Koch Snowflake
64 recursion_depth = 3
65 draw_snowflake_pattern(pen, p0, p1,
    recursion_depth)
66 draw_snowflake_pattern(pen, p1, p2,
    recursion_depth)
67 draw_snowflake_pattern(pen, p2, p0,
    recursion_depth)
68
69 # Final update and keep window open
70 screen.update()
71 screen.mainloop()
```


Output

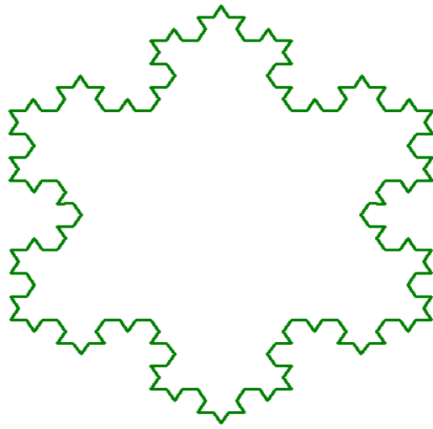


Figure 10: Snowflake Pattern