Md: Israil Hosen

Roll: 2010876110

Computer Graphics Lab

August 20, 2025

Experiment-1

Simulate Hidden Surface Elimination or Visual Surface Detection

Python Script

```
import turtle

# Setup
screen = turtle.Screen()
screen.bgcolor("white")
screen.title("Visual Surface Detection")

pen = turtle.Turtle()
pen.speed(1)
pen.pensize(2)
pen.color("black")

# Draw Main Axes
def draw_axes():
    pen.speed(0)

    # X-axis
    pen.penup()
    pen.goto(-300, 0)
    pen.pendown()
    pen.goto(300, 0)
    # Y-axis
    pen.penup()
    pen.goto(0, -300)
    pen.pendown()
    pen.goto(0, 300)

    pen.speed(1)

# Draw Triangle
def drawTriangle():
    x = [10, 50, 100]
    y = [100, 20, 100]

    pen.penup()
    pen.goto(x[0], y[0])
    pen.pendown()
    pen.fillcolor("green")
    pen.begin_fill()
    pen.goto(x[1], y[1])
    pen.goto(x[2], y[2])
    pen.goto(x[0], y[0])
    pen.end_fill()

# Draw Circle
def drawCircle():
    pen.penup()
    pen.goto(100, 55)
    pen.pendown()
    pen.fillcolor("blue")
    pen.begin_fill()
    pen.circle(45)
    pen.end_fill()

# Draw Rectangle
def drawRectangle():
    x1, y1 = 100, 100
    x2, y2 = 180, 180

    pen.penup()
    pen.goto(x1, y1)
    pen.pendown()
    pen.fillcolor("red")
    pen.begin_fill()
    pen.goto(x2, y1)
    pen.goto(x2, y2)
    pen.goto(x1, y2)
    pen.goto(x1, y1)
    pen.end_fill()

#------------------------------
# Main Function
#------------------------------
# Sequence of drawing
draw_axes()
sequence = "RCT"
for shape in sequence:
    if shape == "C":
        drawCircle()
    elif shape == "T":
        drawTriangle()
    else:
        drawRectangle()

pen.hideturtle()
turtle.done()
```

Output

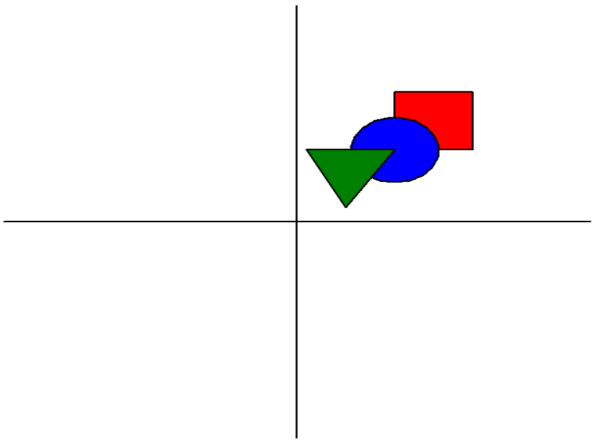

Figure 1: Hidden Surface Elimination

Experiment-2

Implement the Cohen Sutherland Line Clipping Algorithm

Python Script

```
import turtle
# Clipping Window Parameters
x_left, x_right = -100, 300
y_bottom, y_top = -50, 200
# Region Codes
Left, Right, Bottom, Top = 1, 2, 4, 8

# Draw axes
def draw_axes(pen, width, height):
    pen.penup()
    pen.goto(-width / 2, 0)
    pen.pendown()
    pen.goto(width / 2, 0)
    pen.write("X", align="center", font=("Arial", 12, "normal"))

    pen.penup()
    pen.goto(0, -height / 2)
    pen.pendown()
    pen.goto(0, height / 2)
    pen.write("Y", align="center", font=("Arial", 12, "normal"))
    pen.penup()

# Function to calculate region code for a point
def regionCode(x, y):
    code = 0
    if x > x_right:
        code |= Right
    elif x < x_left:
        code |= Left
    if y > y_top:
        code |= Top
    elif y < y_bottom:
        code |= Bottom
    return code

# Cohen-Sutherland Line Clipping Algorithm
def cohenSutherland(x1, y1, x2, y2, pen):
    code1 = regionCode(x1, y1)
    code2 = regionCode(x2, y2)
    while True:
        if not (code1 | code2):  # Line completely inside
            drawLine(x1, y1, x2, y2, "green", pen)
            return
        elif code1 & code2:  # Line completely outside
            return
        else:  # Line partially inside
```

```
            code = code1 if code1 else code2

            if code & Top:
                y = y_top
                x = x1 + (x2 - x1) * (y - y1) / (y2 - y1)
            elif code & Bottom:
                y = y_bottom
                x = x1 + (x2 - x1) * (y - y1) / (y2 - y1)
            elif code & Left:
                x = x_left
                y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)
            elif code & Right:
                x = x_right
                y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)

            if code == code1:
                x1, y1 = x, y
                code1 = regionCode(x1, y1)
            else:
                x2, y2 = x, y
                code2 = regionCode(x2, y2)

# Draw a line helper
def drawLine(x1, y1, x2, y2, color, pen):
    pen.penup()
    pen.goto(x1, y1)
    pen.pendown()
    pen.pencolor(color)
    pen.goto(x2, y2)

#------------------------------
# Main Function
#------------------------------
# screen stup
WIDTH, HEIGHT = 800, 600
screen = turtle.Screen()
screen.title("Cohen-Sutherland Line Clipping")
screen.setup(width=WIDTH, height=HEIGHT)
screen.bgcolor("white")

# pen setup
pen = turtle.Turtle()
pen.speed(2)
pen.pensize(2)
pen.pencolor("Black")
draw_axes(pen, WIDTH, HEIGHT)

# Draw the clipping rectangle
drawLine(x_left, y_bottom, x_right, y_bottom, "green", pen)
drawLine(x_right, y_bottom, x_right, y_top, "green", pen)
drawLine(x_right, y_top, x_left, y_top, "green", pen)
drawLine(x_left, y_top, x_left, y_bottom, "green", pen)

# Original line
x1, y1, x2, y2 = -180, -30, 300, 300
drawLine(x1, y1, x2, y2, "red", pen)

# Clipped line
cohenSutherland(x1, y1, x2, y2, pen)

pen.hideturtle()
turtle.done()
```

Output

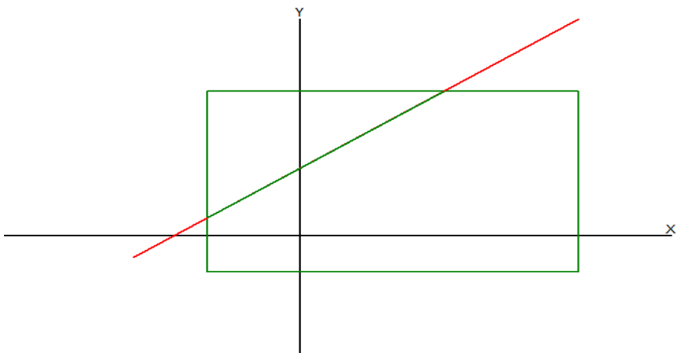

Figure 2: Cohen Sutherland Line Clipping

Experiment-3

Implement the Sutherland-Hodgman Polygon Clipping Algorithm

Python Script

```
import turtle

# Point structure equivalent
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# Clipping window
wMin = Point(100, 100)
wMax = Point(300, 250)

# Clipping edge constants
LEFT, RIGHT, BOTTOM, TOP = 0, 1, 2, 3

# Check if point is inside a clip edge
def inside(p, edge):
    if edge == LEFT:
        return p.x >= wMin.x
    elif edge == RIGHT:
        return p.x <= wMax.x
    elif edge == BOTTOM:
        return p.y >= wMin.y
    elif edge == TOP:
        return p.y <= wMax.y


# Intersection with clip edge
def intersect(p1, p2, edge):
    m = (p2.y - p1.y) / (p2.x - p1.x) if p1.x != p2.x else 1e9
    if edge == 0:
        x = wMin.x
        y = p1.y + (wMin.x - p1.x) * m
    elif edge == 1:
        x = wMax.x
        y = p1.y + (wMax.x - p1.x) * m
    elif edge == 2:
        y = wMin.y
        x = p1.x + (wMin.y - p1.y) / m if m != 0 else p1.x
    else:
        y = wMax.y
        x = p1.x + (wMax.y - p1.y) / m if m != 0 else p1.x
    return Point(x, y)


# Clip polygon against one edge
def clip_polygon(points, edge):
    clipped = []
    for i in range(len(points)):
        curr = points[i]
        prev = points[i - 1]
        curr_in = inside(curr, edge)
        prev_in = inside(prev, edge)

        if prev_in and curr_in:
            clipped.append(curr)
        elif not prev_in and curr_in:
            clipped.append(intersect(prev, curr, edge))
            clipped.append(curr)
        elif prev_in and not curr_in:
            clipped.append(intersect(prev, curr, edge))
    return clipped

# Drawing helpers
def draw_axes(pen, width, height):
    pen.penup()
    pen.goto(-width / 2, 0)
    pen.pendown()
    pen.goto(width / 2, 0)
    pen.write("X", align="center", font=("Arial", 12, "normal"))

    pen.penup()
    pen.goto(0, -height / 2)
    pen.pendown()
    pen.goto(0, height / 2)
    pen.write("Y", align="center", font=("Arial", 12, "normal"))
    pen.penup()

def draw_polygon(points, color):
    if not points:
        return
    pen.pencolor(color)
    pen.penup()
    pen.goto(points[0].x, points[0].y)
    pen.pendown()
    for p in points[1:]:
        pen.goto(p.x, p.y)
```

```
    pen.goto(points[0].x, points[0].y)

def draw_clip_window(pen):
    pen.pencolor("green")
    pen.penup()
    pen.goto(wMin.x, wMin.y)
    pen.pendown()
    pen.goto(wMax.x, wMin.y)
    pen.goto(wMax.x, wMax.y)
    pen.goto(wMin.x, wMax.y)
    pen.goto(wMin.x, wMin.y)

# -------------------------
# Main program
# -------------------------

# Setup turtle screen
WIDTH, HEIGHT = 800, 600
screen = turtle.Screen()
screen.title("Cohen-Sutherland Line Clipping")
screen.setup(width=WIDTH, height=HEIGHT)
screen.bgcolor("white")

# pen setup
pen = turtle.Turtle()
pen.speed(3)
pen.pensize(2)
pen.pencolor("Black")

# Define star polygon
star = [
    Point(200, 300), Point(250, 220), Point(330, 220), Point(260, 170),
    Point(300, 80), Point(200, 130), Point(100, 80), Point(140, 170),
    Point(70, 220), Point(150, 220)
]

# Draw scene
draw_axes(pen, WIDTH, HEIGHT)
draw_clip_window(pen)
draw_polygon(star, "black")

# Step-by-step clipping
edges = [LEFT, RIGHT, BOTTOM, TOP]
colors = ["red", "blue", "cyan", "green"]

for i, edge in enumerate(edges):
    screen.update()
    star = clip_polygon(star, edge)
    draw_polygon(star, colors[i])

pen.hideturtle()
turtle.done()
```

Output

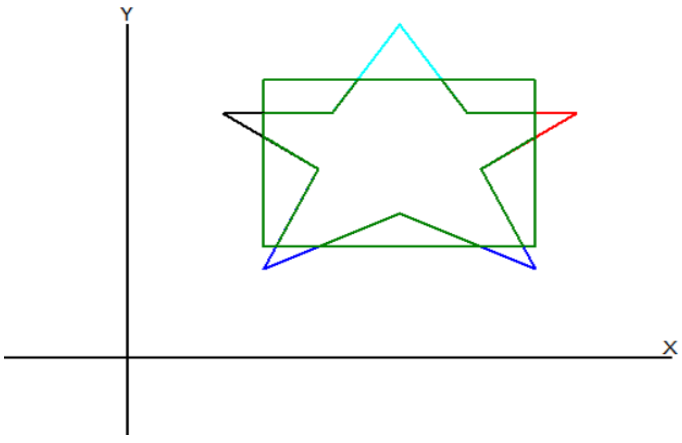

Figure 3: Sutherland Hodgman Polygon Clipping

Experiment-4

Create the Bezier Curve

Python Script

```
import turtle
import math

# ===================== Math Utilities ======================
def factorial(n):
    if n < 2:
        return 1
    return n * factorial(n - 1)

def nCr(n, r):
    return factorial(n) / (factorial(r) * factorial(n - r))

# Bernstein ploynomial
def bezier_basis(k, n, u):
    return nCr(n, k) * (u ** k) * ((1 - u) ** (n - k))

# ====================== Bezier Curve Function ======================
def bezier_curve(pen, points, steps=1000):
    n = len(points) - 1
    pen.pencolor("green")

    # Draw curve
    pen.penup()
    for i in range(steps + 1):
        u = i / steps
        x, y = 0, 0
        for k in range(n + 1):
            b = bezier_basis(k, n, u)
            x += points[k][0] * b
            y += points[k][1] * b
        if i == 0:
            pen.goto(x, y)
            pen.pendown()
        else:
            pen.goto(x, y)

    # Draw control points
    pen.pencolor("red")
    pen.penup()
    for (x, y) in points:
        pen.goto(x, y-3)
        pen.pendown()
        pen.circle(3)
        pen.penup()

    # Draw control polygon
    pen.pencolor("gray")
    pen.penup()
    pen.goto(points[0])
    pen.pendown()
    for (x, y) in points[1:]:
        pen.goto(x, y)

# Draw axes
def draw_axes(pen, width, height):
    pen.penup()
    pen.goto(-width / 2, 0)
    pen.pendown()
    pen.goto(width / 2, 0)
    pen.write("X")

    pen.penup()
    pen.goto(0, -height / 2)
    pen.pendown()
    pen.goto(0, height / 2)
    pen.write("Y")
    pen.penup()

# ====================== Main ======================
# screen stup
WIDTH, HEIGHT = 800, 600
screen = turtle.Screen()
screen.title("Bezier Curve using Python Turtle")
screen.setup(width=WIDTH, height=HEIGHT)
screen.bgcolor("white")
screen.tracer(0)

# pen setup
pen = turtle.Turtle()
pen.speed(0)
pen.pensize(2)
pen.pencolor("Black")
draw_axes(pen, WIDTH, HEIGHT)

# control points
control_points = [(27, 243), (101, 47), (324, 197), (437, 23)]

# Draw bezier curve
bezier_curve(pen, control_points, steps=1000)
```

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

Output

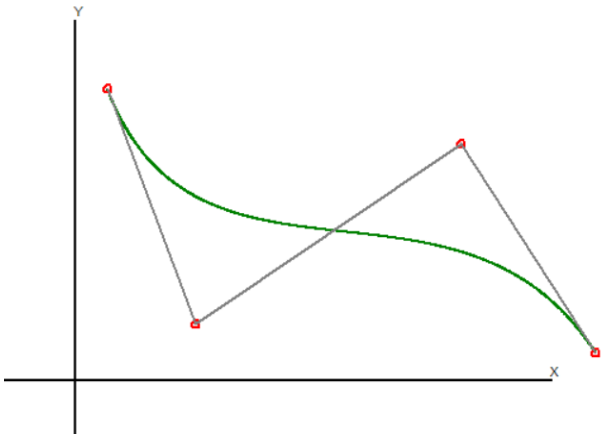

Figure 4: Bezier curve

Experiment-5

Simulate 2D Geometric Translation

Python Script

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

```
# ============================

# Screen setup
WIDTH, HEIGHT = 800, 600
screen = turtle.Screen()
screen.title("2D Triangle Translation with Turtle")
screen.setup(width=WIDTH, height=HEIGHT)
screen.bgcolor("white")

pen = turtle.Turtle()
pen.speed(0)
pen.pensize(2)
pen.pencolor("black")

# Draw axes
draw_axes(pen, WIDTH, HEIGHT)

# Define triangle vertices
original_triangle = [(50, 50), (150, 50), (100, 120)]

# Translation factors
tx, ty = (150, 100)

# Compute translated triangle
translated_triangle = []
for point in original_triangle:
    x, y = point
    new_x = x + tx
    new_y = y + ty
    translated_triangle.append((new_x, new_y))

# Draw original and translated triangles
draw_triangle(pen, original_triangle, "black", "Original")
draw_triangle(pen, translated_triangle, "green", "Translated")

pen.hideturtle()
turtle.done()
```

Output

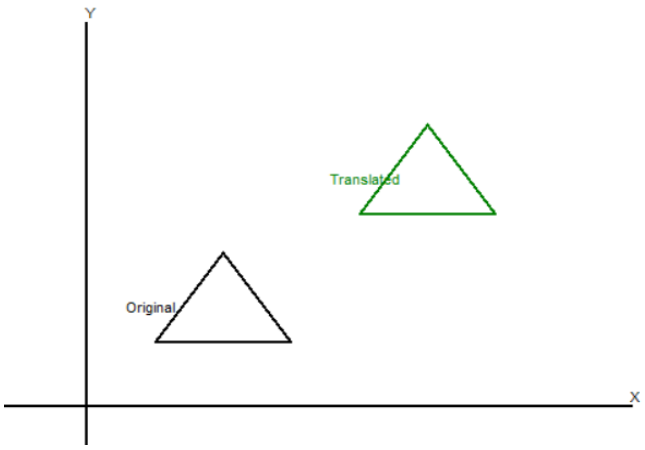

Figure 5: Geometric Translation

Experiment-5

Simulate 2D Geometric Rotation

Python Script

```
import turtle
import math

# ============================
# Helper Functions
# ============================
```

```
def draw_triangle(pen, triangle, color, label=None):
    pen.pencolor(color)
    pen.penup()
    pen.goto(triangle[0][0], triangle[0][1])

    pen.pendown()
    # Draw the triangle edges
    pen.goto(triangle[1][0], triangle[1][1])
    pen.goto(triangle[2][0], triangle[2][1])
    pen.goto(triangle[0][0], triangle[0][1])
    pen.penup()

    # Draw label if provided
    if label:
        pen.goto(triangle[0][0] - 20, triangle[0][1] + 20)
        pen.write(label)

def draw_axes(pen, width, height):
    # X-axis
    pen.penup()
    pen.goto(-width / 2, 0)
    pen.pendown()
    pen.goto(width / 2, 0)
    pen.write("X")

    # Y-axis
    pen.penup()
    pen.goto(0, -height / 2)
    pen.pendown()
    pen.goto(0, height / 2)
    pen.write("Y")

    pen.penup()

def rotate_triangle(triangle, angle_deg, pivot):
    xp, yp = pivot
    angle_rad = math.radians(angle_deg)
    cos_theta = math.cos(angle_rad)
    sin_theta = math.sin(angle_rad)

    rotated = []
    for x, y in triangle:
        # Translate(shift) point relative to pivot
        x_shift = x - xp
        y_shift = y - yp
        # Apply rotation
        new_x = xp + (x_shift * cos_theta - y_shift * sin_theta)
        new_y = yp + (x_shift * sin_theta + y_shift * cos_theta)
        rotated.append((new_x, new_y))

    return rotated

# ============================
# Main Program
# ============================

# Screen setup
WIDTH, HEIGHT = 800, 600
screen = turtle.Screen()
screen.title("2D Triangle Rotation with Turtle")
screen.setup(width=WIDTH, height=HEIGHT)
screen.bgcolor("white")

pen = turtle.Turtle()
pen.speed(0)
pen.pensize(2)
pen.pencolor("black")

# Draw axes
draw_axes(pen, WIDTH, HEIGHT)

# Define triangle vertices
original_triangle = [(50, 50), (150, 50), (100, 120)]

# Rotation settings
angle = 45
pivot_point = (100, 50)

# Compute rotated triangle
rotated_triangle = rotate_triangle(original_triangle, angle, pivot_point)

# Draw original and rotated triangles
draw_triangle(pen, original_triangle, "black", "Original")
draw_triangle(pen, rotated_triangle, "red", "Rotated")

pen.hideturtle()
turtle.done()
```

Output

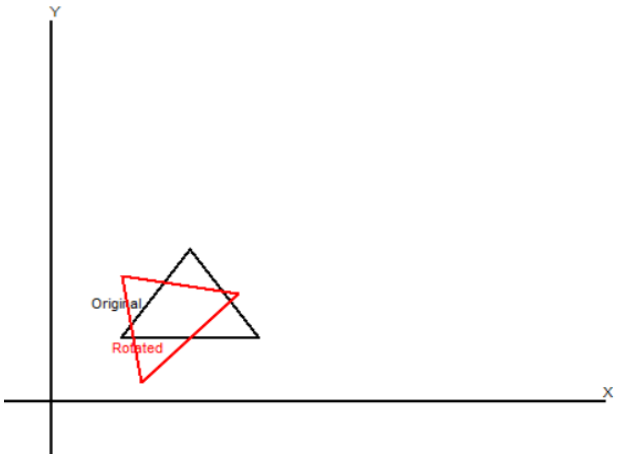

Figure 6: Geometric Rotation

Experiment-5

Simulate 2D Geometric Scaling

Python Script

```
import turtle

# ============================
# Helper Functions
# ============================
def draw_triangle(pen, triangle, color, label=None):
    pen.pencolor(color)
    pen.penup()
    pen.goto(triangle[0][0], triangle[0][1])

    pen.pendown()
    # Draw the triangle edges
    pen.goto(triangle[1][0], triangle[1][1])
    pen.goto(triangle[2][0], triangle[2][1])
    pen.goto(triangle[0][0], triangle[0][1])
    pen.penup()

    # Draw label if provided
    if label:
        pen.goto(triangle[0][0] - 20, triangle[0][1] + 20)
        pen.write(label)

def draw_axes(pen, width, height):
    # X-axis
    pen.penup()
    pen.goto(-width / 2, 0)
    pen.pendown()
    pen.goto(width / 2, 0)
    pen.write("X")

    # Y-axis
    pen.penup()
    pen.goto(0, -height / 2)
    pen.pendown()
    pen.goto(0, height / 2)
    pen.write("Y")

    pen.penup()

def scale_triangle(triangle, sx, sy, pivot):
    xp, yp = pivot
    scaled = []
    for x, y in triangle:
        # Translate point relative to pivot
        x_shift = x - xp
        y_shift = y - yp
        # Apply scaling
        new_x = xp + x_shift * sx
```

```
        new_y = yp + y_shift * sy
        scaled.append((new_x, new_y))
    return scaled

# ============================
# Main Program
# ============================

# Screen setup
WIDTH, HEIGHT = 800, 600
screen = turtle.Screen()
screen.title("2D Triangle Scaling with Turtle")
screen.setup(width=WIDTH, height=HEIGHT)
screen.bgcolor("white")

pen = turtle.Turtle()
pen.speed(0)
pen.pensize(2)
pen.pencolor("black")

# Draw axes
draw_axes(pen, WIDTH, HEIGHT)

# Define triangle vertices
original_triangle = [(50, 50), (150, 50), (100, 120)]

# Scaling settings
sx, sy = 1.5, 0.8
pivot_point = (100, 50)

# Compute scaled triangle
scaled_triangle = scale_triangle(original_triangle, sx, sy, pivot_point)

# Draw original and scaled triangles
draw_triangle(pen, original_triangle, "black")
draw_triangle(pen, scaled_triangle, "green")

pen.hideturtle()
turtle.done()
```

Output

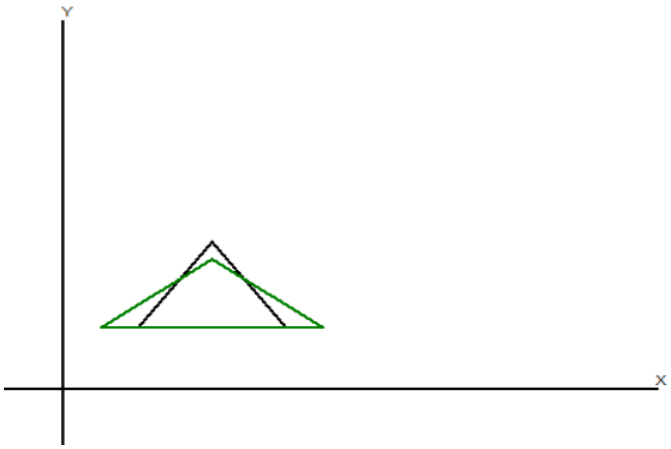

Figure 7: Geometric Scaling

Experiment-6

Draw a Line with Bresenham Line Drawing Algorithm

Python Script

```
import turtle

# ============================
# Helper Functions
# ============================
def draw_pixel(pen, x, y, color="black"):
    pen.penup()
    pen.goto(x, y)
    pen.dot(4, color)
    pen.pendown()

def bresenham_line(pen, x1, y1, x2, y2, color="black"):
    # Ensure left-to-right drawing
    if x1 > x2:
        x1, x2 = x2, x1
        y1, y2 = y2, y1
    dx = x2 - x1
    dy = abs(y2 - y1)
    p = 2 * dy - dx

    y = y1
    for x in range(x1, x2 + 1):
        draw_pixel(pen, x, y, color)
        if p >= 0:
            p += 2 * (dy - dx)
            y += 1 if y1 < y2 else -1
        else:
            p += 2 * dy

def draw_axes(pen, width, height):
    pen.pencolor("gray")
    # X-axis
    pen.penup()
    pen.goto(-width//2, 0)
    pen.pendown()
    pen.goto(width//2, 0)
    pen.write("X", align="center", font=("Arial", 12, "normal"))

    # Y-axis
    pen.penup()
    pen.goto(0, -height//2)
    pen.pendown()
    pen.goto(0, height//2)
    pen.write("Y", align="center", font=("Arial", 12, "normal"))
    pen.penup()

# ============================
# Main Program
# ============================

# screen setup
WIDTH, HEIGHT = 800, 600
screen = turtle.Screen()
screen.title("Bresenham Line Drawing with Turtle")
screen.setup(width=WIDTH, height=HEIGHT)
screen.bgcolor("white")
screen.tracer(0)

# pen setup
pen = turtle.Turtle()
pen.speed(0)
pen.pensize(2)
pen.pencolor("black")

# Draw axes for reference
draw_axes(pen, WIDTH, HEIGHT)

# Draw lines using Bresenham algorithm
lines = [
    ((-200, -100), (200, 100)),
    ((-200,100), (200,-100))

]

for (start, end) in lines:
    x0, y0 = start
    x1, y1 = end
    bresenham_line(pen, x0, y0, x1, y1, color="blue")

pen.hideturtle()
turtle.done()
```

Output

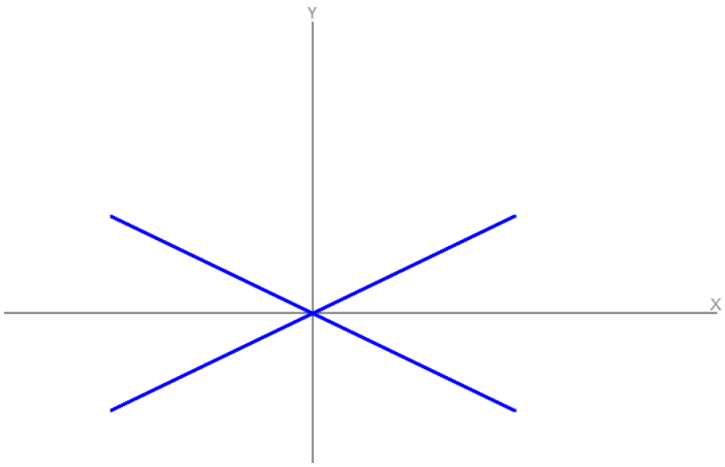

Figure 8: Bresenham Line Drawing

Experiment-7

Draw a Circle with Bresenham Circle Drawing Algorithm

Python Script

```
import turtle

# ============================
# Helper Functions
# ============================
def draw_pixel(pen, x, y, color="black"):
    pen.penup()
    pen.goto(x, y)
    pen.dot(4, color)
    pen.pendown()

def bresenham_circle(pen, xc, yc, r, color="black"):
    x = 0
    y = r
    p = 3 - 2 * r

    def plot_circle_points(xc, yc, x, y):
        # Draw all 8 symmetric points
        draw_pixel(pen, xc + x, yc + y, color)
        draw_pixel(pen, xc - x, yc + y, color)
        draw_pixel(pen, xc + x, yc - y, color)
        draw_pixel(pen, xc - x, yc - y, color)
        draw_pixel(pen, xc + y, yc + x, color)
        draw_pixel(pen, xc - y, yc + x, color)
        draw_pixel(pen, xc + y, yc - x, color)
        draw_pixel(pen, xc - y, yc - x, color)

    plot_circle_points(xc, yc, x, y)

    while x < y:
        x += 1
        if p < 0:
            p += 4 * x + 6
        else:
            y -= 1
            p += 4 * (x - y) + 10
        plot_circle_points(xc, yc, x, y)

def draw_axes(pen, width, height):
    pen.pencolor("gray")
    # X-axis
    pen.penup()
```

```
    pen.goto(-width//2, 0)
    pen.pendown()
    pen.goto(width//2, 0)
    pen.write("X", align="center",
    font=("Arial", 12, "normal"))

    # Y-axis
    pen.penup()
    pen.goto(0, -height//2)
    pen.pendown()
    pen.goto(0, height//2)
    pen.write("Y", align="center",
    font=("Arial", 12, "normal"))
    pen.penup()

# ============================
# Main Program
# ============================

#setup screen
WIDTH, HEIGHT = 800, 600
screen = turtle.Screen()
screen.title("Bresenham Circle
    Drawing with Turtle")
screen.setup(width=WIDTH, height=
    HEIGHT)
screen.bgcolor("white")
screen.tracer(0)

# setup pen
pen = turtle.Turtle()
pen.speed(0)
pen.pensize(2)
pen.pencolor("black")

draw_axes(pen, WIDTH, HEIGHT)

# Example circles: (center_x,
    center_y, radius)
circles = [(0, 0, 100), (-150, 50,
    50), (200, -100, 75)]

for xc, yc, r in circles:
    bresenham_circle(pen, xc, yc, r
    , color="green")

pen.hideturtle()
turtle.done()
```

Output

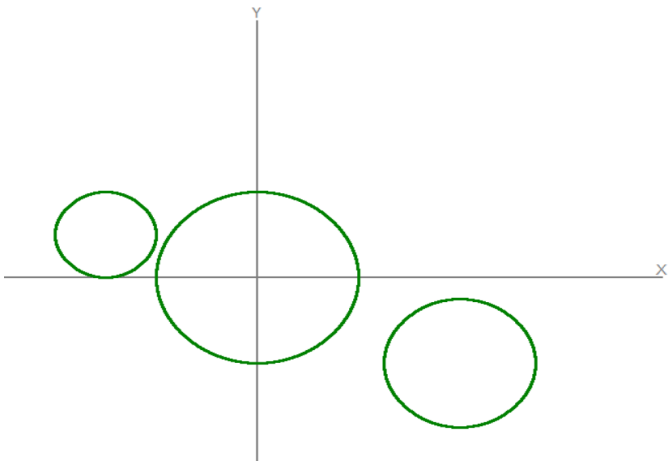

Figure 9: Bresenham Circle Drawing

Experiment-8

Draw the Snowflake Pattern with Fractal Geometry

Python Script

```
import turtle
import time
import math

# ============================
# snowflake pattern Draw Function
# ============================
def draw_snowflake_pattern(pen, start, end, depth):
    if depth == 0:
        pen.penup()
        pen.goto(start)
        pen.pendown()
        pen.goto(end)
        time.sleep(0.01)
        screen.update()
        return

    # Divide the segment into three equal parts
    dx = (end[0] - start[0]) / 3
    dy = (end[1] - start[1]) / 3

    # First point at one-third
    p1 = (start[0] + dx, start[1] + dy)

    # Third point at two-thirds
    p3 = (start[0] + 2 * dx, start[1] + 2 * dy)

    # Calculate the peak of the equilateral triangle
    px = p1[0] + (p3[0] - p1[0]) / 2 + math.sqrt(3) * (p3[1] - p1[1]) / 2
    py = p1[1] + (p3[1] - p1[1]) / 2 - math.sqrt(3) * (p3[0] - p1[0]) / 2
    p2 = (px, py)

    # Recursively draw the four segments
    draw_snowflake_pattern(pen, start, p1, depth - 1)
    draw_snowflake_pattern(pen, p1, p2, depth - 1)
    draw_snowflake_pattern(pen, p2, p3, depth - 1)
    draw_snowflake_pattern(pen, p3, end, depth - 1)

# ============================
# Main program
# ============================

#setup the screen
WIDTH, HEIGHT = 800, 600
screen = turtle.Screen()
screen.title("Snowflake Pattern with Turtle")
screen.setup(width=WIDTH, height=HEIGHT)
screen.bgcolor("white")
screen.tracer(0)

# setup the pen
pen = turtle.Turtle()
pen.hideturtle()
pen.pensize(3)
pen.pencolor("green")
pen.speed(0)

# Define Triangle Points
p0 = (0, 250)
p1 = (-200, -100)
p2 = (200, -100)

# Draw Koch Snowflake
recursion_depth = 3
draw_snowflake_pattern(pen, p0, p1, recursion_depth)
draw_snowflake_pattern(pen, p1, p2, recursion_depth)
draw_snowflake_pattern(pen, p2, p0, recursion_depth)

# Final update and keep window open
screen.update()
screen.mainloop()
```

Output

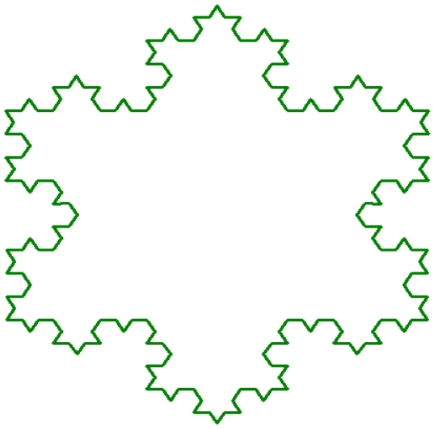

Figure 10: Snowflake Pattern