Lab Assignment: 3

Course Name: Computer Graphics Lab

Course Code: CSE422

Submitted to:

Md. Zami Al Zunaed Farabe

Lecturer

Department of CSE

Daffodil International University

Submitted By:

Tanjina Ahmed Tuly

ID: 221-15-4902

Section: 61_A1

Department of CSE

Daffodil International University

Submission Date: 24/10/2025

Title: Implement Mid Point Circle Algorithm in OpenGL.

Introduction: In this lab, I implemented the Mid Point Circle Drawing Algorithm (also known as Bresenham's Circle Algorithm) to render multiple circles using OpenGL and GLUT. The algorithm efficiently draws circles by exploiting eight-way symmetry, computing points in only one octant and reflecting them to the other seven octants. This approach minimizes computation while producing smooth, pixel-perfect circles.

The project demonstrates a central large circle (radius 150) surrounded by eight smaller circles (radius 75) positioned at equal distances around the center. The pattern creates a flower-like or atomic orbital visualization. The algorithm uses only integer arithmetic and a decision parameter to determine whether to move east (E) or southeast (SE) at each step, making it highly efficient for raster graphics. All circles are rendered in yellow color on a black background, and the implementation showcases how geometric primitives can be constructed using pure algorithmic approaches without relying on OpenGL's built-in circle drawing functions.

Contents:

1. **Functions Used:**

    a. **MidpointCircle(radius, x0, y0)**

    - Implements the Mid Point Circle Drawing Algorithm
    - Parameters: radius (circle radius), x0, y0 (center coordinates)
    - Process:
        - Initializes: x = 0, y = radius, decision parameter d = 1 - radius
        - While x < y:
            - If d < 0: Move East → d = d + 2x + 3, increment x only
            - If d ≥ 0: Move Southeast → d = d + 2x - 2y + 5, increment x and decrement y
        - Calls Circlepoints() at each step to plot 8-way symmetric points
    - Computes points for only one octant (0° to 45°) and uses symmetry for the rest

b. Circlepoints(x, y, x0, y0)

- Exploits eight-way symmetry of circles
- Takes a point (x, y) relative to origin and center (x0, y0)
- Plots 8 symmetric points in all octants:
  - (x+x0, y+y0), (y+x0, x+y0) – Octants 1, 2
  - (-y+x0, x+y0), (-x+x0, y+y0) – Octants 3, 4
  - (-x+x0, -y+y0), (-y+x0, -x+y0) – Octants 5, 6
  - (y+x0, -x+y0), (x+x0, -y+y0) – Octants 7, 8
- This symmetry reduces computation by 8x

c. draw_points(x, y)

- Wrapper function to plot individual pixels using OpenGL
- Sets point size to 3 pixels for better visibility
- Uses glBegin(GL_POINTS) and glVertex2f() to render a single point
- Called by Circlepoints() for each symmetric position

d. showScreen()

- Main display callback function
- Clears screen, sets yellow color (RGB: 1.0, 1.0, 0.0)
- Draws 9 circles total:
  - 1 large center circle: radius 150 at (250, 250)
  - 8 surrounding circles: radius 75 each, positioned at:
  - Cardinal directions: (x+150, y), (x-150, y), (x, y+150), (x, y-150)
  - Diagonal directions: (x±106, y±106) – positioned at 45° intervals
- Uses glutSwapBuffers() for smooth double-buffered rendering

2. **Shapes Implemented:**

    a. **9 circles in total**

        - 1 large circle (radius 150, center)
        - 8 smaller circles (radius 75 each, surrounding)
        - All rendered in yellow with 3-pixel point size
        - Pattern creates a symmetric flower/orbital design

Graph:

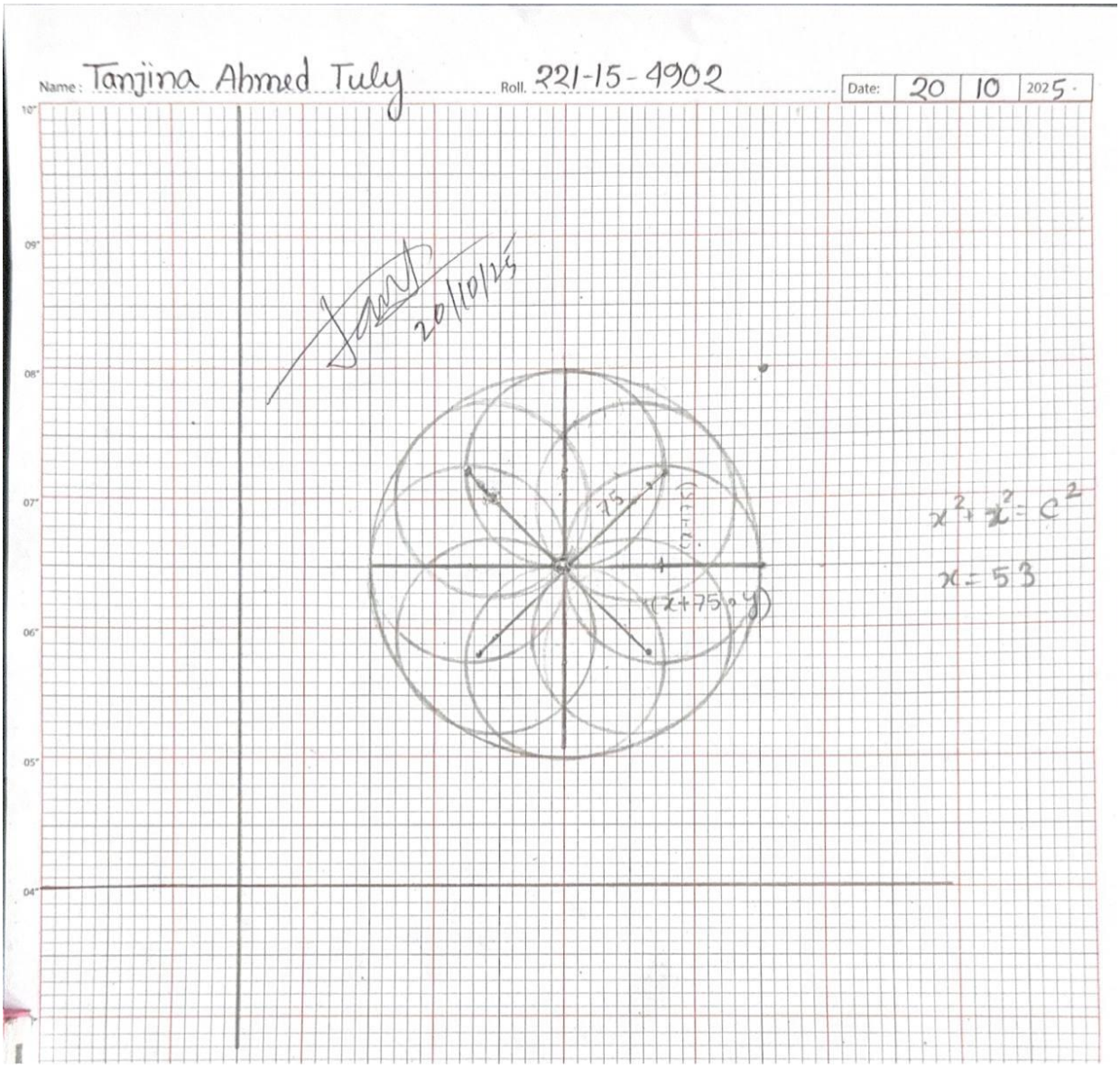


Figure 1: Simple Cube and House using Mid Point Line Algorithm.

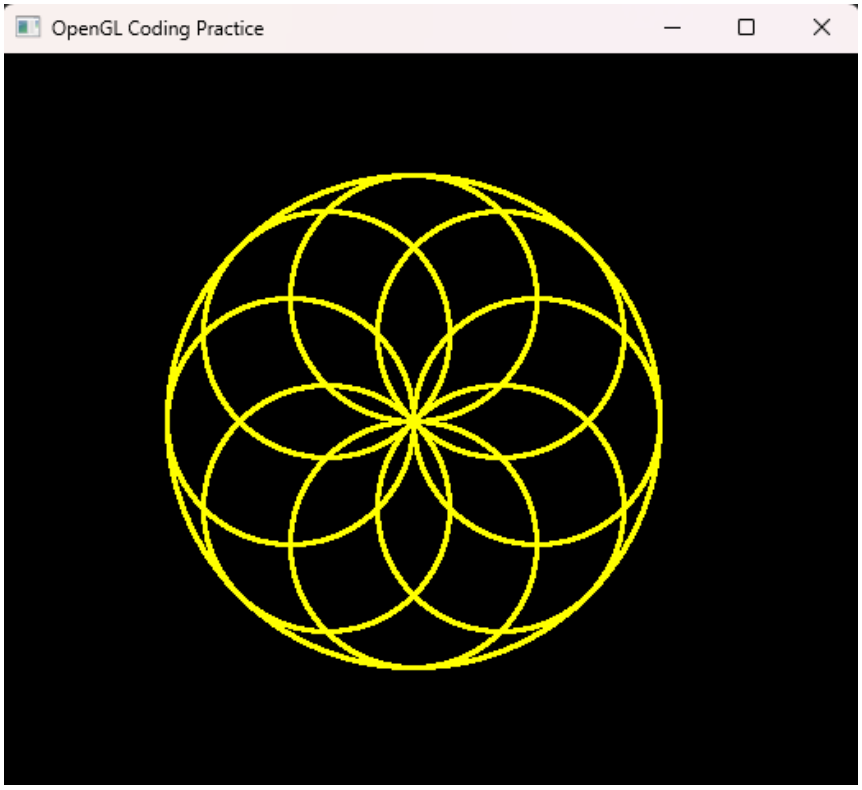
Code:

Circle:

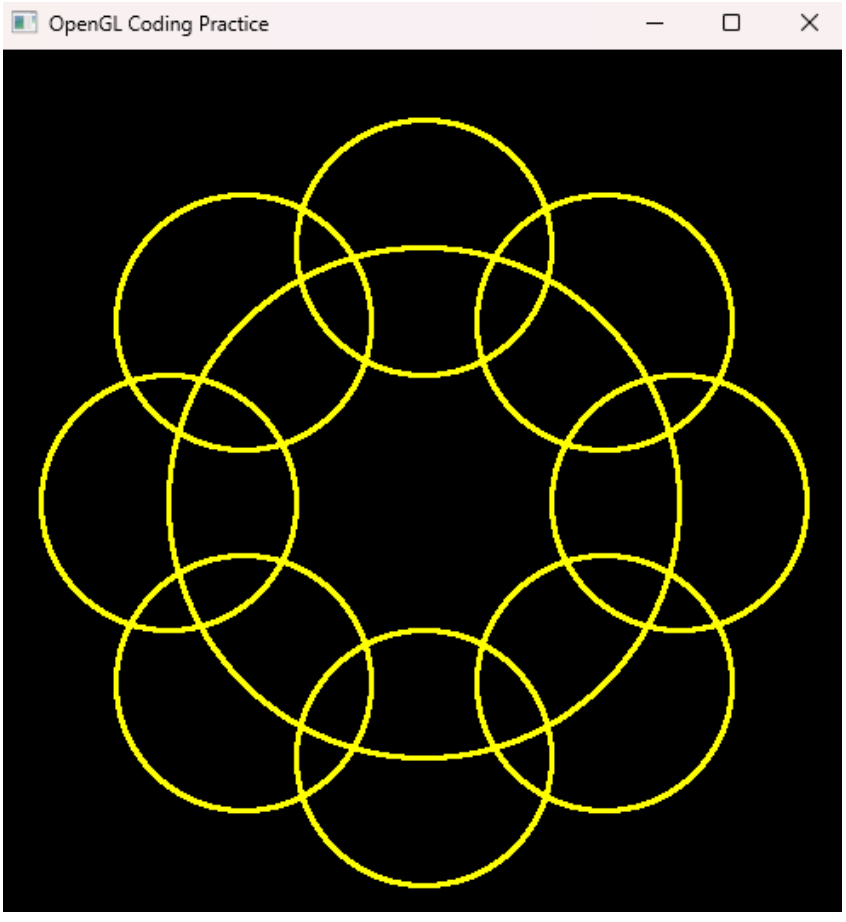
```
def MidpointCircle(radius, x0, y0):
    """
    Mid Point Circle Drawing Algorithm
    Draws a circle using integer arithmetic and 8-way symmetry
    """
    x = 0
    y = radius
    d = 1 - radius  # Initial decision parameter

    # Plot initial points
    Circlepoints(x, y, x0, y0)

    # Loop while in first octant (0° to 45°)
    while x < y:
        if d < 0:
            # Move East (E)
            d = d + 2*x + 3
            x = x + 1
        else:
            # Move Southeast (SE)
            d = d + 2*x - 2*y + 5
            x = x + 1
            y = y - 1

        # Plot 8 symmetric points
        Circlepoints(x, y, x0, y0)


def Circlepoints(x, y, x0, y0):
    """
    Exploits 8-way symmetry to plot points in all octants
    """
    draw_points(x + x0, y + y0)      # Octant 1
    draw_points(y + x0, x + y0)      # Octant 2
    draw_points(-y + x0, x + y0)     # Octant 3
    draw_points(-x + x0, y + y0)     # Octant 4
    draw_points(-x + x0, -y + y0)    # Octant 5
    draw_points(-y + x0, -x + y0)    # Octant 6
    draw_points(y + x0, -x + y0)     # Octant 7
    draw_points(x + x0, -y + y0)     # Octant 8


def draw_points(x, y):
    """
    Renders a single pixel at (x, y)
    """
    glPointSize(3)
```

```
    glBegin(GL_POINTS)
    glVertex2f(x, y)
    glEnd()


def showScreen():
    """
    Main drawing function - renders all circles
    """
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    iterate()
    glColor3f(1.0, 1.0, 0.0)  # Yellow color

    # Center coordinates
    x = 250
    y = 250

    # Draw large center circle

    # Draw 8 inside circles (radius 75)
    MidpointCircle(150, x, y)
    MidpointCircle(75, x+75, y)
    MidpointCircle(75, x, y+75)
    MidpointCircle(75, x-75, y)
    MidpointCircle(75, x, y-75)
    MidpointCircle(75, x+53, y+53)
    MidpointCircle(75, x-53, y+53)
    MidpointCircle(75, x-53, y-53)
    MidpointCircle(75, x+53, y-53)

    # Draw 8 surrounding circles (radius 75)
    MidpointCircle(75, x+150, y)        # Right
    MidpointCircle(75, x+106, y+106)    # Top-right diagonal
    MidpointCircle(75, x, y+150)        # Top
    MidpointCircle(75, x-106, y+106)    # Top-left diagonal
    MidpointCircle(75, x-150, y)        # Left
    MidpointCircle(75, x-106, y-106)    # Bottom-left diagonal
    MidpointCircle(75, x, y-150)        # Bottom
    MidpointCircle(75, x+106, y-106)    # Bottom-right diagonal

    glutSwapBuffers()
```

Output 1:



Output 2:



Discussion: Implemented the Mid Point Circle Drawing Algorithm to efficiently render circles using only integer arithmetic and 8-way symmetry, then create a visually interesting pattern using multiple circles.

Solution Approach:

The Mid Point Circle Algorithm is based on the implicit circle equation: $f(x,y) = x^2 + y^2 - r^2 = 0$

Key Implementation Steps:

1. **Decision Parameter Initialization:**
    - Start at (0, r) — the topmost point of the circle
    - Initialize decision parameter: $d = 1 - r$
    - This parameter determines whether the midpoint between E and SE pixels is inside or outside the circle
2. **Iterative Pixel Selection:**
    - Loop while $x < y$ (first octant only, 0° to 45°)
    - At each step, check decision parameter d:
        - If $d < 0$: Midpoint is inside circle → choose East pixel, update $d = d + 2x + 3$
        - If $d \geq 0$: Midpoint is outside circle → choose Southeast pixel, update $d = d + 2x - 2y + 5$
    - This incremental approach uses only addition/subtraction (no multiplication or division)
3. **Eight-way Symmetry Exploitation:**
    - For every point (x, y) computed in the first octant, I generate 8 symmetric points:
        - (±x, ±y) and (±y, ±x)
    - This reduces computation by a factor of 8 compared to computing all octants separately

4. **Pattern Design:**

    - Created a central large circle (radius 150) at viewport center (250, 250)
    - Positioned 8 smaller circles (radius 75) around it:
        - Cardinal positions: ±150 offset in x or y directions (distance = 150)
        - Diagonal positions: ±106 offset in both x and y (distance ≈ 150, using Pythagorean theorem: $\sqrt{(106^2 + 106^2)} \approx 150$)
    - This creates a symmetric "atomic orbital" or flower-like pattern