

# Problem Statement: Geometric shape generation using different shape generating algorithms

## Introduction

Geometric shape generating algorithms are computational methods that create various shapes, such as polygons, circles, and curves, using mathematical rules and parameters. These algorithms are fundamental in computer graphics, design, and engineering, enabling the generation of precise and visually appealing shapes. This assignment focuses on applying these algorithms to draw and color the flag of Bangladesh.

## Problem Scenario

As a graphic designer in a company, the task is to develop a system to generate and color the flag of Bangladesh in different sizes, selected by the user each time, using algorithms such as the Digital Differential Analyzer (DDA) line drawing algorithm, Bresenham line drawing algorithm, and midpoint circle drawing algorithm.

## Tasks and Solutions

### Task 1: Selection of Algorithms

For drawing the flag, the DDA line drawing algorithm will be used to create the rectangular background, and the midpoint circle drawing algorithm will be used to draw the red circular disc. For coloring the flag, the flood fill algorithm will be employed to fill the rectangle with green and the circle with red.

### Task 2: Code to Create the Flag Shape

A detailed pseudocode implementation to create the flag shape is provided below.

```
ALGORITHM DrawFlag(width, height)
    // Draw rectangle using DDA
    DrawDDA(0, 0, width, 0)    // Top
    DrawDDA(0, 0, 0, height)   // Left
    DrawDDA(width, 0, width, height) // Right
    DrawDDA(0, height, width, height) // Bottom

    // Draw circle using Midpoint algorithm
    DrawMidpointCircle(width/4, height/2, height/4)
END ALGORITHM

ALGORITHM DrawDDA(x1, y1, x2, y2)
    dx ← x2 - x1
    dy ← y2 - y1
```

```

    steps ← maximum of |dx| and |dy|
    xIncrement ← dx / steps
    yIncrement ← dy / steps
    x ← x1
    y ← y1
    PlotPixel(x, y)
    FOR k ← 0 TO steps - 1 DO
        x ← x + xIncrement
        y ← y + yIncrement
        PlotPixel(round(x), round(y))
    END FOR
END ALGORITHM

ALGORITHM DrawMidpointCircle(xCenter, yCenter, radius)
    x ← 0
    y ← radius
    d ← 1 - radius
    WHILE y > x DO
        PlotPixel(xCenter + x, yCenter + y)
        PlotPixel(xCenter + y, yCenter + x)
        PlotPixel(xCenter - x, yCenter + y)
        PlotPixel(xCenter - y, yCenter + x)
        PlotPixel(xCenter - x, yCenter - y)
        PlotPixel(xCenter - y, yCenter - x)
        PlotPixel(xCenter + x, yCenter - y)
        PlotPixel(xCenter + y, yCenter - x)
        IF d < 0 THEN
            d ← d + 2 * x + 3
        ELSE
            d ← d + 2 * (x - y) + 5
            y ← y - 1
        END IF
        x ← x + 1
    END WHILE
END ALGORITHM

```

### Task 3: System to Color the Flag

The pseudocode implementation for the flood fill algorithm to color the flag is detailed below.

```

ALGORITHM FloodFill(x, y, newColor, oldColor)
    IF GetPixel(x, y) ≠ oldColor OR GetPixel(x, y) = newColor THEN
        RETURN
    
```

**END IF**

SetPixel(x, y, newColor)

FloodFill(x + 1, y, newColor, oldColor)

FloodFill(x - 1, y, newColor, oldColor)

FloodFill(x, y + 1, newColor, oldColor)

FloodFill(x, y - 1, newColor, oldColor)

**END ALGORITHM**

**ALGORITHM** ColorFlag(width, height)

FloodFill(1, 1, GREEN, GetPixel(1, 1)) // Fill rectangle with green

FloodFill(width/4, height/2, RED, GetPixel(width/4, height/2)) // Fill circle with red

**END ALGORITHM**

## Results

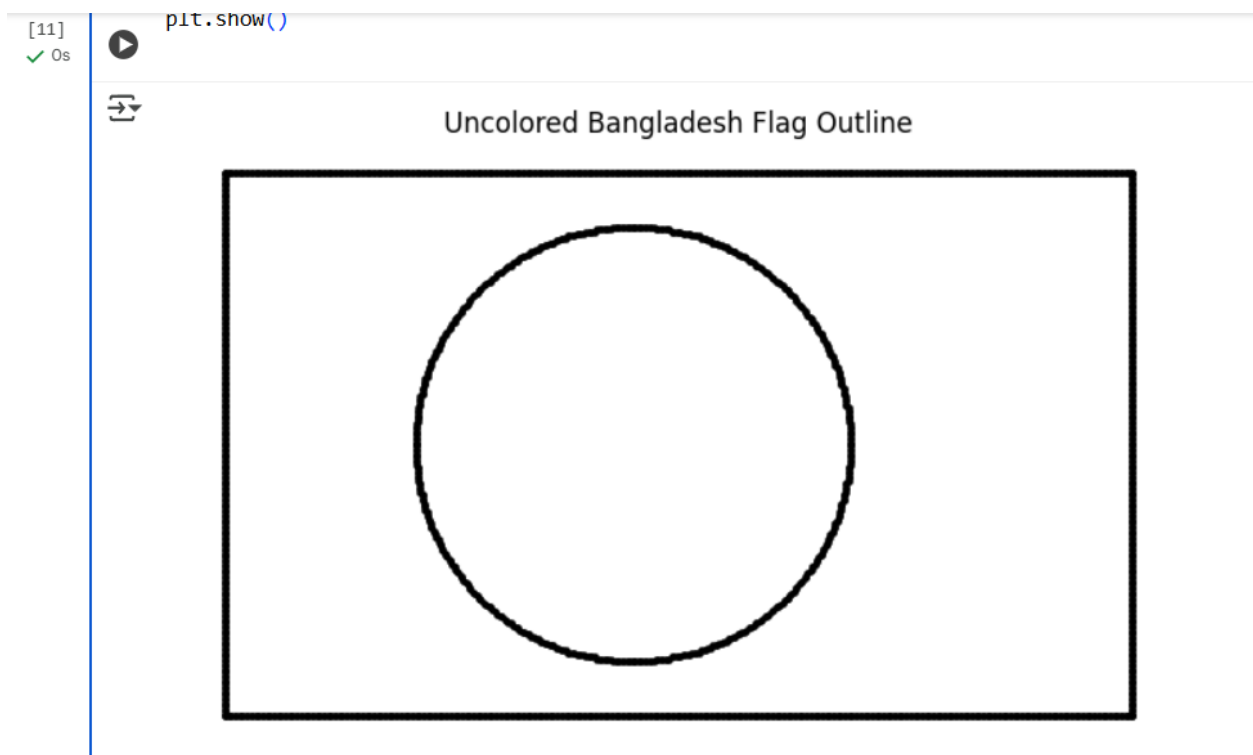


Figure 1: Shape of The Given Flag (Uncolored)



Figure 2: Shape of The Given Flag (Colored)

#### Task 4: Reflection on Advantages and Limitations

- **Advantages:** - DDA provides high accuracy for line drawing with smooth interpolation. - Midpoint circle algorithm is efficient, requiring minimal computation for circle generation. - Flood fill ensures complete and uniform coloring of enclosed areas. - **Limitations:** - DDA can be computationally intensive due to floating-point operations. - Midpoint circle algorithm may introduce minor rounding errors. - Flood fill can be slow and memory-intensive for large areas due to recursive calls, potentially leading to stack overflow.

#### Complex Problem-Solving Questions

- Question:** Does the solution need in-depth engineering knowledge?  
**Answer:** Yes, a deep understanding of graphics algorithms, including their mathematical underpinnings and practical implementation, is crucial for crafting an effective solution.
- Question:** Does the solution involve wide-ranging or conflicting technical, engineering, or other issues?  
**Answer:** Generally, no major conflicts emerge, but optimizing the algorithms for varying screen resolutions and performance might pose a challenge that requires thoughtful adjustments.
- Question:** Is the solution well-known, or does it require abstract thinking and analysis to formulate?

**Answer:** The algorithms like DDA, Midpoint Circle, and Flood Fill are well-established in computer graphics, yet tailoring them to accommodate dynamic flag sizes demands analytical creativity.

- d. Question:** Does the solution involve infrequently encountered issues?

**Answer:** No, these are common graphics challenges frequently addressed in both academic and industry contexts.

- e. Question:** Does the solution need adherence to standards and codes of practice?

**Answer:** Yes, adherence to basic coding standards and graphics conventions, such as accurate color representation and pixel precision, is necessary.

- f. Question:** Does the solution involve stakeholders with conflicting technical requirements?

**Answer:** Not directly, though managing diverse client preferences for flag dimensions and color fidelity could present minor hurdles.

- g. Question:** Does the solution involve interdependence between sub-problems or parts?

**Answer:** Absolutely, the successful coloring of the flag relies on the precise completion of drawing the rectangle and circle beforehand.