Experiment No: 2, Name: Circle Drawing Algorithms

Objective:

The objective of this experiment is to study and implement different circle drawing algorithms in Computer Graphics. The experiment aims to understand how circles can be generated efficiently on a raster display using mathematical and incremental methods.

Theory:

- Midpoint Circle Algorithm: Uses integer calculations to draw a circle by evaluating the midpoint between two candidate pixels and deciding which pixel is closer to the actual circle.
- Bresenham's Circle Algorithm is an optimized version of the midpoint algorithm. It uses only integer calculations and is one of the fastest algorithms for circle drawing
- Trigonometric Circle Algorithm uses parametric equations x=rcos⁡θx = r\cos\thetax=rcosθ and y=rsin⁡θy = r\sin\thetay=rsinθ. It provides smooth circles but requires floating-point and trigonometric calculations.
- Polynomial Circle Algorithm uses the basic equation of a circle x2+y2=r2x^2 + y^2 = r^2x2+y2=r2. It calculates pixel positions using square root operations, which makes it simple but slower.

Requirements:

- Programming Language: C++
- Graphics Library: OpenGL (GLUT)
- Software: CodeBlocks

Procedure / Description of Code

Steps:

1. Initialize the graphics mode using OpenGL and create a display window.
2. Define the center coordinates (xc,yc)(x_c, y_c)(xc,yc) and radius rrr of the circle.
3. Select a circle drawing algorithm to implement.

4. For each algorithm:
5. Calculate the pixel positions based on its mathematical method.
6. Use the symmetry of the circle to plot points in all eight octants.
7. Plot the calculated points using `glVertex2i()` function.
8. Refresh the display to show the drawn circle.
9. Repeat the process for other circle drawing algorithms.

Source Code with Input/Output:

Midpoint Circle Algorithm

```
#include <windows.h>

#include <GL/gl.h>

#include <GL/glut.h>

#include <cmath>

int centerX = 250; // circle center x

int centerY = 250; // circle center y

int radius  = 100; // circle radius

void drawPixel(int x, int y)

{

   glBegin(GL_POINTS);

   glVertex2i(x, y);

   glEnd();

}
```

```
/* Midpoint Circle Algorithm */
void midpointCircle()
{
    int x = 0;
    int y = radius;
    int p = 1 - radius;

    while (x <= y)
    {
        // Draw all 8 octants
        drawPixel(centerX + x, centerY + y);
        drawPixel(centerX - x, centerY + y);
        drawPixel(centerX + x, centerY - y);
        drawPixel(centerX - x, centerY - y);
        drawPixel(centerX + y, centerY + x);
        drawPixel(centerX - y, centerY + x);
        drawPixel(centerX + y, centerY - x);
        drawPixel(centerX - y, centerY - x);

        x++;
        if (p < 0)
```

```
            p += 2 * x + 1;
        else
        {
            y--;
            p += 2 * (x - y) + 1;
        }
    }
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1, 0, 0);  // Red circle
    midpointCircle();
    glFlush();
}

void init()
{
    glClearColor(1, 1, 1, 1);       // White background
    glPointSize(2);
```

```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 500, 0, 500);
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Midpoint Circle Algorithm");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Polynomial Circle Algorithm

```
void polynomialCircle(int xc, int yc, int r) {
    for (int x = -r; x <= r; x++) {
        int y = sqrt(r * r - x * x);
        glVertex2i(xc + x, yc + y);
        glVertex2i(xc + x, yc - y);
```

```
    }
}
```

Trigonometric Circle Algorithm

```
void trigonometricCircle(int xc, int yc, int r) {
    for (float theta = 0; theta <= 360; theta += 0.5) {
        float rad = theta * 3.14159 / 180;
        int x = r * cos(rad);
        int y = r * sin(rad);
        glVertex2i(xc + x, yc + y);
    }
}
```

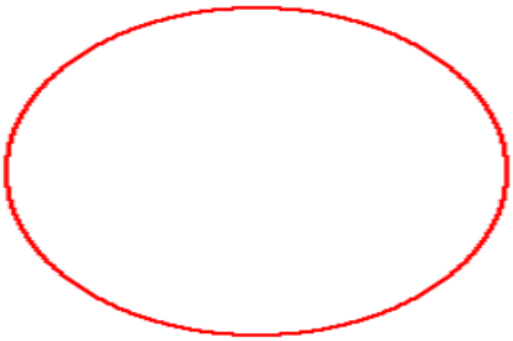
Bresenham's Circle Algorithm

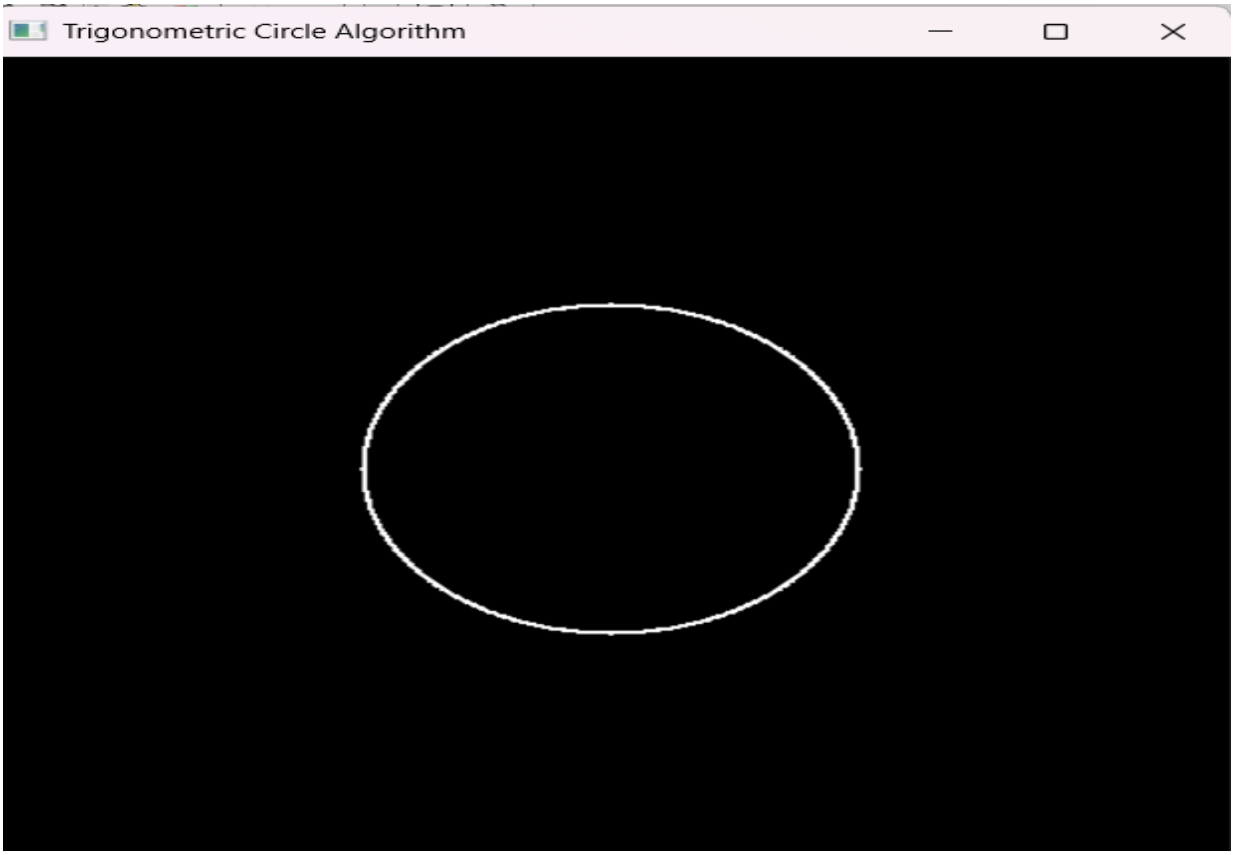
```
void bresenhamCircle(int xc, int yc, int r) {
    int x = 0, y = r;
    int d = 3 - 2 * r;

    while (x <= y) {
        glVertex2i(xc + x, yc + y);
        glVertex2i(xc - x, yc + y);
        glVertex2i(xc + x, yc - y);
        glVertex2i(xc - x, yc - y);
```

```
        glVertex2i(xc + y, yc + x);
        glVertex2i(xc - y, yc + x);
        glVertex2i(xc + y, yc - x);
        glVertex2i(xc - y, yc - x);

        if (d < 0)
            d += 4 * x + 6;
        else {
            d += 4 * (x - y) + 10;
            y--;
        }
        x++;
    }
}
```

```
int centerX = 250; // circle center x
int centerY = 250; // circle center y
int radius  = 100; // circle radius
```


Midpoint Circle Algorithm




Trigonometric Circle Algorithm

Conclusion / Discussion

In this experiment, different circle drawing algorithms were successfully implemented and studied. The Polynomial and Trigonometric algorithms are easy to understand but computationally slower due to the use of floating-point and mathematical functions. The Midpoint and Bresenham's circle algorithms are more efficient and faster as they use integer arithmetic and symmetry properties. Among all, Bresenham's circle algorithm is the most efficient for real-time applications. This experiment helps in understanding the importance of optimized algorithms in computer graphics.