Experiment No: 3

Experiment Name: Draw Ellipse using Polynomial & Trigonometric Algorithm

---

Objective:

- To draw an **ellipse** using **Polynomial (Cartesian) algorithm** and **Trigonometric (Parametric) algorithm**.
- To understand **pixel plotting for 2D curves** in computer graphics.

Theory:

- **Polynomial / Cartesian Algorithm:**
  Uses the ellipse equation:

  Points are calculated along the **x-axis and y-axis** and plotted using pixels.

- **Trigonometric / Parametric Algorithm:**
  Uses parametric form:

  Iterates θ from 0° to 360° to generate the ellipse points.

Requirements / Equipment

- **Software:** Code::Blocks
- **Libraries:** OpenGL, GLUT

Procedure / Description of Code

Steps:

1. Initialize OpenGL window using glutInit and set 2D orthographic projection using gluOrtho2D.
2. Define a drawPixel() function to plot points with GL_POINTS.

3. **Polynomial Algorithm:**
    - Iterate x from -a to +a, compute y from ellipse equation.
    - Plot symmetric points along x and y axes.
4. **Trigonometric Algorithm:**
    - Iterate θ from 0° to 360°, compute x and y using cos and sin.
    - Plot each point to draw the ellipse.
5. Run the program and observe the ellipse drawn on the screen.

Source Code with Input / Output:

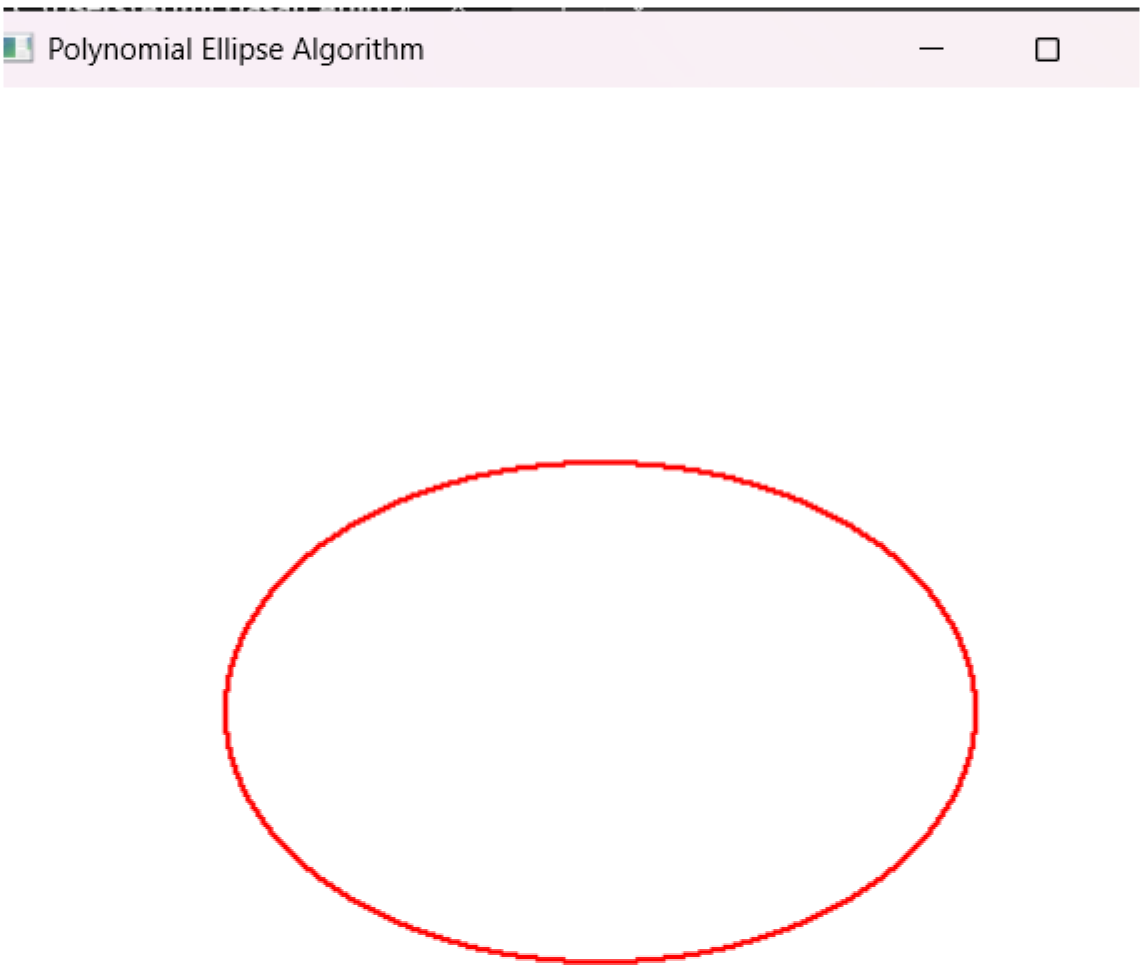
Polynomial Algorithm

```
#include <windows.h>

#include <GL/gl.h>

#include <GL/glut.h>

#include <cmath>

int centerX = 250;

int centerY = 250;

int a = 150; // X-axis radius

int b = 100; // Y-axis radius

void drawPixel(int x, int y)

{

   glBegin(GL_POINTS);

   glVertex2i(x, y);

   glEnd();

}

/* Trigonometric / Parametric Ellipse */

void trigonometricEllipse()

{

   int steps = 360; // more steps = smoother ellipse

   for(int i = 0; i <= steps; i++)

   {
```

```
        float theta = 2 * 3.14159265 * i / steps;
        int x = centerX + (int)(a * cos(theta) + 0.5);
        int y = centerY + (int)(b * sin(theta) + 0.5);
        drawPixel(x, y);
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0, 0, 1); // blue ellipse
    trigonometricEllipse();
    glFlush();
}
void init()
{
    glClearColor(1, 1, 1, 1);
    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("Trigonometric Ellipse Algorithm");
    init();
```

```
    glutDisplayFunc(display);

    glutMainLoop();

    return 0;

}
```

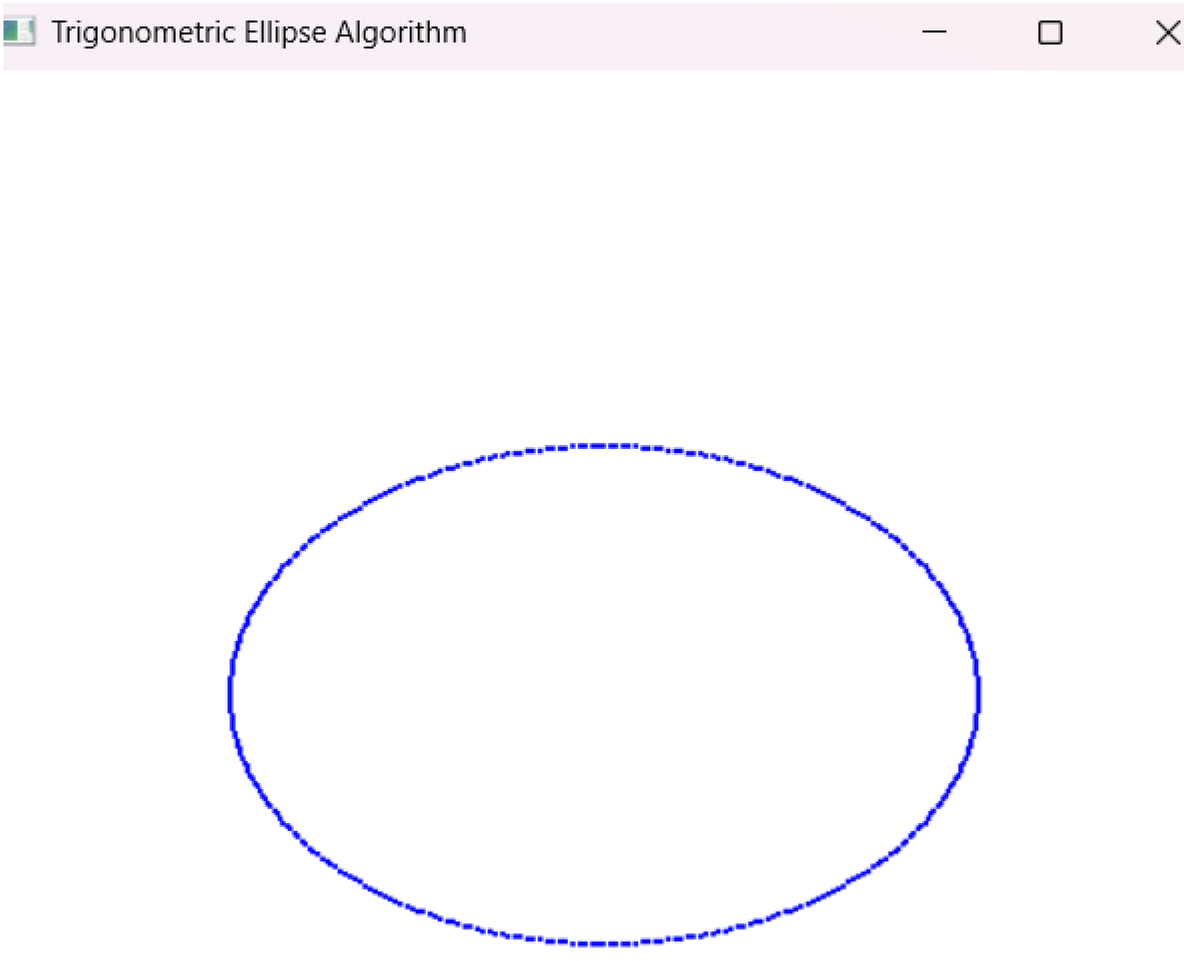
```
// Ellipse center and radii
int centerX = 250;
int centerY = 250;
int a = 150; // radius along X-axis
int b = 100; // radius along Y-axis
```



Trigonometric / Parametric Algorithm

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>
#include <cmath>

int centerX = 250;
int centerY = 250;
int a = 150; // X-axis radius
int b = 100; // Y-axis radius

void drawPixel(int x, int y)
{
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
}

/* Trigonometric / Parametric Ellipse */
void trigonometricEllipse()
{
   int steps = 360; // more steps = smoother ellipse
   for(int i = 0; i <= steps; i++)
   {
      float theta = 2 * 3.14159265 * i / steps;
      int x = centerX + (int)(a * cos(theta) + 0.5);
      int y = centerY + (int)(b * sin(theta) + 0.5);
      drawPixel(x, y);
   }
```

```
}

void display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0, 0, 1); // blue ellipse
   trigonometricEllipse();
   glFlush();
}

void init()
{
   glClearColor(1, 1, 1, 1);
   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("Trigonometric Ellipse Algorithm");
   init();
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```



Conclusion / Discussion

- Both Polynomial and Trigonometric algorithms successfully draw an ellipse.
- Polynomial Algorithm: faster, uses integer calculations, good for pixel-based plotting.
- Trigonometric Algorithm: easier to implement, produces smoother ellipse when `steps` is large.
- Experiment demonstrates symmetry and pixel plotting in 2D graphics.
- Choosing the algorithm depends on speed vs smoothness requirements.