Experiment No: 1

Experiment Name: Line Drawing Algorithms (Polynomial, DDA & Bresenham)

Theory

Line drawing algorithms are used to generate straight lines on raster displays.

- Polynomial Method uses the equation of a straight line.
- DDA Algorithm generates points incrementally using floating-point calculations.
- Bresenham's Algorithm uses integer arithmetic, making it faster and more efficient.

Requirements

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

Procedure / Description of Code

1. Initialize graphics mode.
2. Take input coordinates of the line.
3. Apply the selected algorithm.
4. Plot pixels using OpenGL functions.
5. Display the output window.

SOURCE CODE:

```
#include <windows.h>

#include <GL/gl.h>

#include <GL/glut.h>

int x1 = 50, y1 = 50;

int x2 = 300, y2 = 200;
```

```
void drawPixel(int x, int y)
{
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
}
/* DDA Line Algorithm */
void ddaLine()
{
   int dx = x2 - x1;
   int dy = y2 - y1;
   int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);
   float xInc = dx / (float)steps;
   float yInc = dy / (float)steps;
   float x = x1;
   float y = y1;

   for (int i = 0; i <= steps; i++)
   {
      drawPixel((int)(x + 0.5), (int)(y + 0.5));
```

```
        x += xInc;
        y += yInc;
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1, 0, 0);

    ddaLine();   // <-- THIS is Experiment-1

    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("Experiment 1: DDA Line Algorithm");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

```
#include <GL/glut.h>
int x1 = 50, y1 = 50;
int x2 = 300, y2 = 200;
```

2) POLYNOMIAL CODE:

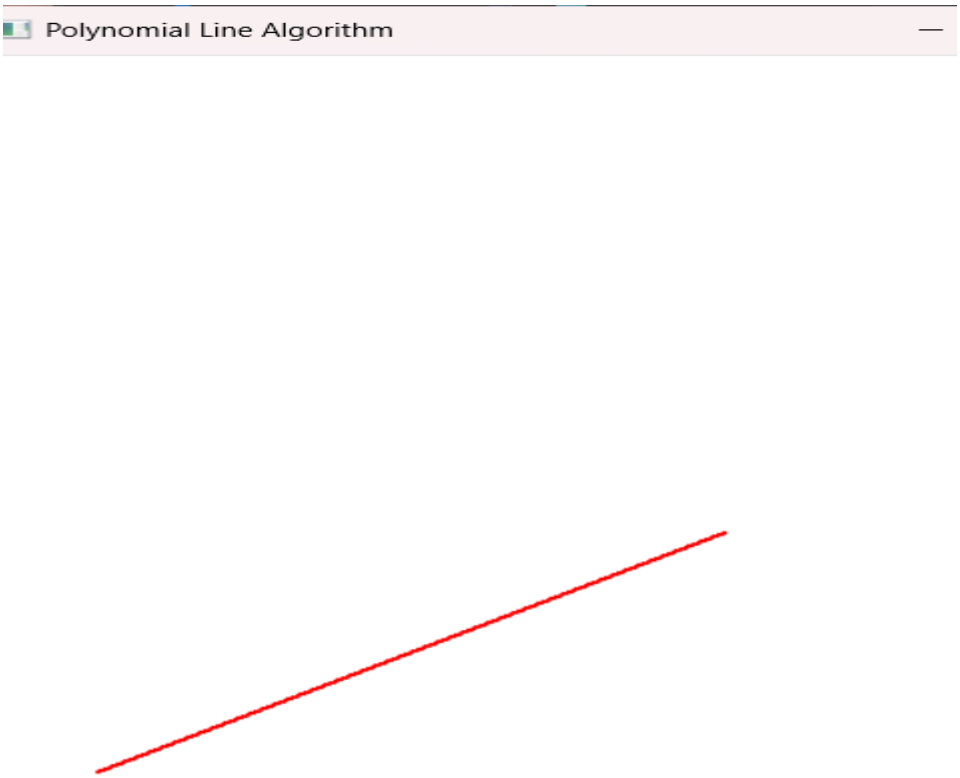
```
#include <windows.h>
```

```
#include <GL/gl.h>
#include <GL/glut.h>
#include <cmath>
int startX = 50;
int startY = 50;
int endX   = 300;
int endY   = 200;
void drawPixel(int x, int y)
{
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
}

/* Polynomial Line Algorithm */
void polynomialLine()
{
   int dx = endX - startX;
   int dy = endY - startY;

   if(dx == 0) // vertical line
   {
      int yStart = (startY < endY) ? startY : endY;
      int yEnd   = (startY > endY) ? startY : endY;
      for(int y = yStart; y <= yEnd; y++)
         drawPixel(startX, y);
      return;
   }
```

```
    float m = (float)dy / dx;
    float c = startY - m * startX;

    if(abs(dx) > abs(dy))
    {
        int xStart = (startX < endX) ? startX : endX;
        int xEnd   = (startX > endX) ? startX : endX;
        for(int x = xStart; x <= xEnd; x++)
        {
            int y = (int)(m*x + c + 0.5);
            drawPixel(x, y);
        }
    }
    else
    {
        int yStart = (startY < endY) ? startY : endY;
        int yEnd   = (startY > endY) ? startY : endY;
        for(int y = yStart; y <= yEnd; y++)
        {
            int x = (int)((y - c)/m + 0.5);
            drawPixel(x, y);
        }
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1,0,0);
    polynomialLine();
```

```
    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("Polynomial Line Algorithm");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

```
#include <GL/glut.h>
int x1 = 50, y1 = 50;
int x2 = 300, y2 = 200;
```



3) BRESENHAM CODE:

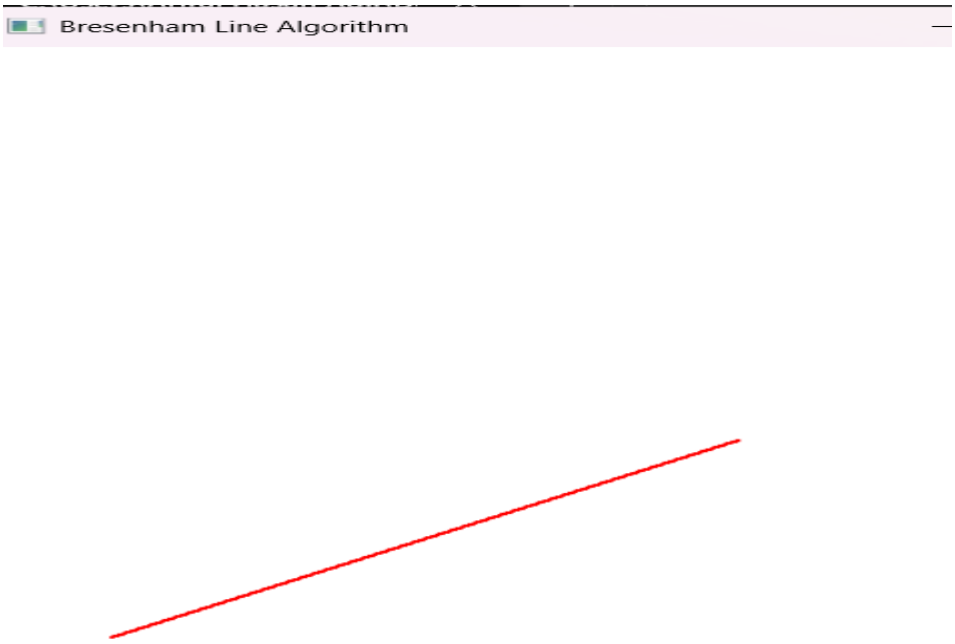
```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>
#include <cmath>  // For abs()
int startX = 50;
int startY = 50;
int endX   = 300;
int endY   = 200;

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

```
}

/* Bresenham Line Drawing Algorithm */
void bresenhamLine()
{
   int dx = abs(endX - startX);
   int dy = abs(endY - startY);

   int sx = (startX < endX) ? 1 : -1;
   int sy = (startY < endY) ? 1 : -1;

   int err = dx - dy;
   int x = startX;
   int y = startY;

   while (true)
   {
      drawPixel(x, y);  // Plot the point

      if (x == endX && y == endY)
         break;

      int e2 = 2 * err;

      if (e2 > -dy)
      {
         err -= dy;
         x += sx;
      }
```

```
        if (e2 < dx)
        {
            err += dx;
            y += sy;
        }
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1, 0, 0);  // Red color line

    bresenhamLine();      // Draw line using Bresenham

    glFlush();
}
void init()
{
    glClearColor(1, 1, 1, 1);  // White background
    glPointSize(2);           // Point size for pixel plotting
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 500, 0, 500); // 2D orthographic projection
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

```
    glutInitWindowSize(500, 500);
    glutCreateWindow("Bresenham Line Algorithm");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

```
#include <GL/glut.h>
int x1 = 50, y1 = 50;
int x2 = 300, y2 = 200;
```



Conclusion / Discussion

In this experiment, different line drawing algorithms were implemented and compared. Bresenham's algorithm was found to be the most efficient due to its use of integer calculations and faster execution.