DDA Line Drawing algorithm (Basic)

```
#include <GL/freeglut.h>
#include <cmath>
#include <iostream>
#include <vector>

float xStart, yStart, xEnd, yEnd;

// White background, black line
void init() {
   glClearColor(1.0, 1.0, 1.0, 1.0); // White background
   glColor3f(0.0, 0.0, 0.0);       // Black line
   glPointSize(3.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 500, 0, 500);
}

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

void ddaLineDrawing() {
   glClear(GL_COLOR_BUFFER_BIT);

   float dx = xEnd - xStart;
   float dy = yEnd - yStart;
   float steps = std::max(std::abs(dx), std::abs(dy));
   float xInc = dx / steps;
   float yInc = dy / steps;

   float x = xStart;
   float y = yStart;

   for (int i = 0; i <= steps; ++i) {
      drawPixel(round(x), round(y));
      x += xInc;
      y += yInc;
   }

   glFlush();
}

void printMajorCoordsToTerminal() {
   std::cout << "\nMajor 10 Coordinates on the line:\n";
```

```
    float dx = xEnd - xStart;
    float dy = yEnd - yStart;
    float steps = std::max(std::abs(dx), std::abs(dy));
    float xInc = dx / steps;
    float yInc = dy / steps;

    float x = xStart;
    float y = yStart;

    int interval = (int)(steps / 9);  // 10 points including start and end

    for (int i = 0, count = 0; i <= steps && count < 10; i += interval, count++) {
        std::cout << "(" << round(x) << ", " << round(y) << ") ";
        x = xStart + xInc * i;
        y = yStart + yInc * i;
    }
    std::cout << std::endl;
}

int main(int argc, char** argv) {
    std::cout << "Enter xStart yStart: ";
    std::cin >> xStart >> yStart;
    std::cout << "Enter xEnd yEnd: ";
    std::cin >> xEnd >> yEnd;

    printMajorCoordsToTerminal(); // Output coordinates before opening window

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("DDA Line Drawing - FreeGLUT");

    init();
    glutDisplayFunc(ddaLineDrawing);
    glutMainLoop();

    return 0;
}
```

Breshenem Line Drawing Algorithm (Basic)

```
#include <GL/freeglut.h>
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

int xStart, yStart, xEnd, yEnd;
vector<pair<int, int> > majorCoords;

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

   // Store major 10 coordinates
   if (majorCoords.size() < 10)
      majorCoords.push_back(make_pair(x, y));
}

void bresenhamLineDrawing() {
   int dx = abs(xEnd - xStart);
   int dy = abs(yEnd - yStart);

   int x = xStart;
   int y = yStart;

   int sx = (xEnd >= xStart) ? 1 : -1;
   int sy = (yEnd >= yStart) ? 1 : -1;

   bool isSteep = dy > dx;

   if (isSteep) {
      swap(x, y);
      swap(dx, dy);
      swap(sx, sy);
   }

   int p = 2 * dy - dx;

   for (int i = 0; i <= dx; ++i) {
      if (isSteep)
         plot(y, x);
      else
         plot(x, y);
```

```
        x += sx;
        if (p >= 0) {
            y += sy;
            p -= 2 * dx;
        }
        p += 2 * dy;
    }
    glFlush();

    // Print major 10 coordinates
    cout << "Major 10 Coordinates:\n";
    for (int i = 0; i < majorCoords.size(); ++i) {
        cout << "(" << majorCoords[i].first << ", " << majorCoords[i].second << ")\n";
    }
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0); // black line
    bresenhamLineDrawing();
}

void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0); // white background
    gluOrtho2D(0, 500, 0, 500);       // 2D coordinate system
}

int main(int argc, char** argv) {
    cout << "Enter xStart yStart: ";
    cin >> xStart >> yStart;
    cout << "Enter xEnd yEnd: ";
    cin >> xEnd >> yEnd;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Bresenham Line Drawing");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Breshenem Circle Drawing Algorithm (Basic)

```
#include <GL/freeglut.h>
#include <iostream>
#include <cmath>
using namespace std;

int xc, yc, r;

void plotPoints(int x, int y) {
   // Plot the 8 symmetric points
   glBegin(GL_POINTS);
      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);
   glEnd();
}

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

   plotPoints(x, y);

   while (x <= y) {
      x++;
      if (d < 0) {
         d += 4 * x + 6;
      } else {
         y--;
         d += 4 * (x - y) + 10;
      }
      plotPoints(x, y);
   }
   glFlush();
}

void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0, 0, 0); // black
   bresenhamCircle();
```

```
}

void init() {
   glClearColor(1.0, 1.0, 1.0, 1.0);  // white background
   gluOrtho2D(0, 500, 0, 500);       // coordinate system
}

int main(int argc, char** argv) {
   cout << "Enter center of circle (xc, yc): ";
   cin >> xc >> yc;
   cout << "Enter radius: ";
   cin >> r;

   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("Bresenham Circle Drawing");
   init();
   glutDisplayFunc(display);
   glutMainLoop();

   return 0;
}
```

2D Transformations (Basic)

```
#include<GL/glut.h>
#include<cmath>
#include<cstdio>

float vertices[4][2] = {{-0.25, -0.25},{0.25, -0.25},{0.25, 0.25},{-0.25, 0.25}};
float tx = 0, ty = 0, sx = 1, sy = 1, shx = 1, shy = 1, angle = 0;
int option = 0;

void drawPolygon() {
   glBegin(GL_LINE_LOOP);
   for (int i = 0; i < 4; i++) {
      float x = vertices[i][0];
      float y = vertices[i][1];
      float newX, newY, rad;

      switch(option) {
         case 1:
            x = x + tx;
            y = y + ty;
            break;
         case 2:
            rad = angle * 3.14159 / 180;
            newX = x * cos(rad) - y * sin(rad);
            newY = x * sin(rad) + y * cos(rad);
            x = newX;
            y = newY;
            break;
         case 3:
            x *= sx;
            y *= sy;
            break;
         case 4:
            x += shx * y;
            y += shy * x;
            break;
         case 5:
            break;
      }
      glVertex2f(x, y);
   }
   glEnd();
}

void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.5, 0.5, 0.5);
```

```
    glBegin(GL_LINES);
    glVertex2f(-1.0, 0.0);
    glVertex2f(1.0, 0.0);
    glVertex2f(0.0, -1.0);
    glVertex2f(0.0, 1.0);
    glEnd();

    //Original Matrix
    glColor3f(0, 0, 1);
    glBegin(GL_LINE_LOOP);
    for(int i = 0; i < 4; i++) {
        glVertex2f(vertices[i][0], vertices[i][1]);
    }
    glEnd();

    //Transformed Matrix
    glColor3f(1, 0, 0);
    drawPolygon();

    glutSwapBuffers();
}

void keyboard(unsigned char key, int x, int y) {
    switch(key) {
        case '0':
            option = 0;
            break;
        case '1':
            option = 1;
            printf("Enter tx, ty");
            scanf("%f %f",&tx, &ty);
            break;
        case '2':
            option = 2;
            printf("Enter angle");
            scanf("%f", &angle);
            break;
        case '3':
            option = 3;
            printf("Enter sx, sy");
            scanf("%f %f", &sx, &sy);
            break;
        case '4':
            option = 4;
            printf("Enter shx, shy");
            scanf("%f %f", &shx, &shy);
            break;
```

```
        case 27:
            exit(0); //esc
    }
    glutPostRedisplay();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Transformations");

    glClearColor(1.0, 1.0, 1.0, 1.0);

    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);

    printf("Press\n1-Translation\n2-Rotation\n3-Scaling\n4-Shearing\n0-Reset\nESC-Exit\n");

    glutMainLoop();
    return 0;
}
```

Flood fill , Boundary Fill (Basic)

```
#include <GL/glut.h>
#include <unistd.h> // For usleep
#include <iostream>
using namespace std;

int width = 800, height = 600;
int fillOption = 0; // 0: No Fill, 1: Flood, 2: Boundary

float bgColor[3] = {1.0, 1.0, 1.0};
float borderColor[3] = {0.0, 0.0, 0.0};
float floodColor[3] = {1.0, 0.0, 0.0};
float boundaryColor[3] = {0.0, 1.0, 0.0};

void setPixel(int x, int y, float color[3]) {
   glColor3fv(color);
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
   glFlush();
   usleep(300);
}

void getPixelColor(int x, int y, float color[3]) {
   glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, color);
}

bool isSameColor(float a[3], float b[3]) {
   return (abs(a[0] - b[0]) < 0.01 &&
           abs(a[1] - b[1]) < 0.01 &&
           abs(a[2] - b[2]) < 0.01);
}

void floodFill(int x, int y, float oldColor[3], float newColor[3]) {
   float color[3];
   getPixelColor(x, y, color);

   if (isSameColor(color, oldColor) && !isSameColor(color, newColor)) {
      setPixel(x, y, newColor);
      floodFill(x + 1, y, oldColor, newColor);
      floodFill(x - 1, y, oldColor, newColor);
      floodFill(x, y + 1, oldColor, newColor);
      floodFill(x, y - 1, oldColor, newColor);
   }
}

void boundaryFill(int x, int y, float bColor[3], float fillColor[3]) {
```

```
    float color[3];
    getPixelColor(x, y, color);

    if (!isSameColor(color, bColor) && !isSameColor(color, fillColor)) {
        setPixel(x, y, fillColor);
        boundaryFill(x + 1, y, bColor, fillColor);
        boundaryFill(x - 1, y, bColor, fillColor);
        boundaryFill(x, y + 1, bColor, fillColor);
        boundaryFill(x, y - 1, bColor, fillColor);
    }
}

void drawShape() {
    glColor3fv(borderColor);

    // Square
    glBegin(GL_LINE_LOOP);
    glVertex2i(200, 200);
    glVertex2i(300, 200);
    glVertex2i(300, 300);
    glVertex2i(200, 300);
    glEnd();

    // Triangle
    glBegin(GL_LINE_LOOP);
    glVertex2i(400, 200);
    glVertex2i(500, 200);
    glVertex2i(450, 300);
    glEnd();

    glFlush();
}

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    drawShape();
}

void mouse(int btn, int state, int x, int y) {
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        int my = height - y;
        int mx = x;
        float clickedColor[3];
        getPixelColor(mx, my, clickedColor);

        if (fillOption == 1 && isSameColor(clickedColor, bgColor)) {
            floodFill(mx, my, bgColor, floodColor);
```

```
        } else if (fillOption == 2 && !isSameColor(clickedColor, borderColor)) {
            boundaryFill(mx, my, borderColor, boundaryColor);
        }
    }
}

void menu(int option) {
    fillOption = option;
    glutPostRedisplay();
}

void init() {
    glClearColor(bgColor[0], bgColor[1], bgColor[2], 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, width, 0, height);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Flood Fill and Boundary Fill - Dev C++ Compatible");
    init();
    glutDisplayFunc(myDisplay);
    glutMouseFunc(mouse);
    glutCreateMenu(menu);
    glutAddMenuEntry("Flood Fill", 1);
    glutAddMenuEntry("Boundary Fill", 2);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutMainLoop();
    return 0;
}
```

DDA Line Drawing Algorithm (Adv)

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

// Structure to store a line's data
struct Line {
   int x1, y1, x2, y2;
   int type;  // 0 = Simple, 1 = Dotted, 2 = Dashed, 3 = Solid
   float color[3]; // RGB values
};

std::vector<Line> lines;  // Stores drawn lines
int lineType = 0;        // Default: Simple Line
float currentColor[3] = {1.0, 0.0, 0.0}; // Default: Red
int xStart, yStart;      // Mouse click start points
bool isDrawing = false;   // Flag for mouse drag

// Function to draw a line using DDA algorithm
void drawDDALine(int x1, int y1, int x2, int y2, int patternType) {
   float dx = x2 - x1;
   float dy = y2 - y1;
   int steps = std::max(abs(dx), abs(dy));
   float xInc = dx / steps;
   float yInc = dy / steps;

   float x = x1;
   float y = y1;

   glBegin(GL_POINTS);
   for (int i = 0; i <= steps; i++) {
      bool draw = true;
      if (patternType == 1) draw = (i % 5 == 0);          // Dotted
      else if (patternType == 2) draw = (i % 10 < 5);      // Dashed
      if (draw) glVertex2i(round(x), round(y));
      x += xInc;
      y += yInc;
   }
   glEnd();
}

// Function to draw the line based on its type
void drawLine(const Line& line) {
   system("cls");
   glColor3fv(line.color);
```

```
    if (line.type == 0 || line.type == 1 || line.type == 2) {
        drawDDALine(line.x1, line.y1, line.x2, line.y2, line.type);
    } else if (line.type == 3) {  // Solid Bold Line
        glLineWidth(3.0);
        glBegin(GL_LINES);
        glVertex2i(line.x1, line.y1);
        glVertex2i(line.x2, line.y2);
        glEnd();
        glLineWidth(1.0);
    }

    printf("Line drawn from (%d, %d) to (%d, %d)\n", (line.x1 - 400), (line.y1 - 300), (line.x2 -
400), (line.y2 - 300));
}

// Function to display all drawn lines
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);

    // Draw X and Y axes
    glBegin(GL_LINES);
    glVertex2f(800, 300);
    glVertex2f(0, 300);
    glEnd();

    glBegin(GL_LINES);
    glVertex2f(400, 600);
    glVertex2f(400, 0);
    glEnd();

    // Ticks on X and Y axes
    for (int i = 1; i < 40; i++) {
        glBegin(GL_LINES);
        glVertex2f((20 * i), 295);
        glVertex2f((20 * i), 305);
        glEnd();
    }

    for (int i = 1; i < 30; i++) {
        glBegin(GL_LINES);
        glVertex2f(395, (20 * i));
        glVertex2f(405, (20 * i));
        glEnd();
    }
```

```
    // Draw all lines
    for (size_t i = 0; i < lines.size(); i++) {
        drawLine(lines[i]);
    }

    glFlush();
}

// Mouse function to handle line drawing
void mouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        xStart = x;
        yStart = 600 - y;  // Convert to OpenGL's coordinate system
        isDrawing = true;
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
        Line newLine;
        newLine.x1 = xStart;
        newLine.y1 = yStart;
        newLine.x2 = x;
        newLine.y2 = 600 - y;
        newLine.type = lineType;
        newLine.color[0] = currentColor[0];
        newLine.color[1] = currentColor[1];
        newLine.color[2] = currentColor[2];

        lines.push_back(newLine);  // Add line to the list
        isDrawing = false;
        glutPostRedisplay();
    }
}

// Keyboard function for undo feature
void keyboard(unsigned char key, int, int) {
    if (key == 'u' || key == 'U') {
        if (!lines.empty()) {
            lines.pop_back();
            glutPostRedisplay();
        }
    }
}

// Function to handle button clicks
void buttonClick(int choice) {
    if (choice == 4) {
        if (!lines.empty()) lines.pop_back();
    } else {
```

```
        lineType = choice;
    }
    glutPostRedisplay();
}

// Function to handle color selection
void colorMenu(int choice) {
    switch (choice) {
        case 0: currentColor[0] = 1.0; currentColor[1] = 0.0; currentColor[2] = 0.0; break; // Red
        case 1: currentColor[0] = 0.0; currentColor[1] = 0.0; currentColor[2] = 1.0; break; // Blue
        case 2: currentColor[0] = 1.0; currentColor[1] = 1.0; currentColor[2] = 0.0; break; //
Yellow
        case 3: currentColor[0] = 0.0; currentColor[1] = 1.0; currentColor[2] = 1.0; break; // Cyan
        case 4: currentColor[0] = 1.0; currentColor[1] = 0.0; currentColor[2] = 1.0; break; //
Magenta
        case 5: currentColor[0] = 1.0; currentColor[1] = 0.5; currentColor[2] = 0.0; break; //
Orange
    }
    glutPostRedisplay();
}

// Menu setup for line selection
void createMenu() {
    int colorSubMenu = glutCreateMenu(colorMenu);
    glutAddMenuEntry("Red", 0);
    glutAddMenuEntry("Blue", 1);
    glutAddMenuEntry("Yellow", 2);
    glutAddMenuEntry("Cyan", 3);
    glutAddMenuEntry("Magenta", 4);
    glutAddMenuEntry("Orange", 5);

    int mainMenu = glutCreateMenu(buttonClick);
    glutAddMenuEntry("Simple Line", 0);
    glutAddMenuEntry("Dotted Line", 1);
    glutAddMenuEntry("Dashed Line", 2);
    glutAddMenuEntry("Solid Line (Bold)", 3);
    glutAddMenuEntry("Undo", 4);
    glutAddSubMenu("Select Color", colorSubMenu);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

// Initialization
void init() {
    glClearColor(1, 1, 1, 1);
    gluOrtho2D(0, 800, 0, 600);
}
```

```
int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(800, 600);
   glutCreateWindow("Line Drawing with Mouse (DDA)");
   init();
   createMenu();
   glutDisplayFunc(display);
   glutMouseFunc(mouse);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

Breshenem Line Drawing Algorithm (Adv)

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

// Structure to store a line's data
struct Line {
   int x1, y1, x2, y2;
   int type;  // 0 = Simple, 1 = Dotted, 2 = Dashed, 3 = Solid
   float color[3]; // RGB values
};

std::vector<Line> lines;  // Stores drawn lines
int lineType = 0;        // Default: Simple Line
float currentColor[3] = {1.0, 0.0, 0.0}; // Default: Red
int xStart, yStart;      // Mouse click start points
bool isDrawing = false;   // Flag for mouse drag

// Function to draw a line using Bresenham's algorithm with line pattern
void drawBresenhamLine(int x1, int y1, int x2, int y2, int patternType) {
   int dx = abs(x2 - x1);
   int dy = abs(y2 - y1);
   int sx = (x1 < x2) ? 1 : -1;
   int sy = (y1 < y2) ? 1 : -1;
   bool isSteep = dy > dx;
   int err = (isSteep ? dy : dx) / 2;
   int count = 0;

   glBegin(GL_POINTS);

   while (true) {
      bool draw = true;
      if (patternType == 1) draw = (count % 5 == 0);        // Dotted
      else if (patternType == 2) draw = (count % 10 < 5);    // Dashed
      if (draw) glVertex2i(x1, y1);

      if (x1 == x2 && y1 == y2) break;

      if (isSteep) {
         y1 += sy;
         err -= dx;
         if (err < 0) {
            x1 += sx;
            err += dy;
         }
```

```
        } else {
            x1 += sx;
            err -= dy;
            if (err < 0) {
                y1 += sy;
                err += dx;
            }
        }

        count++;
    }

    glEnd();
}

// Function to draw the line based on its type
void drawLine(const Line& line) {
    system("cls");
    glColor3fv(line.color);

    if (line.type == 0 || line.type == 1 || line.type == 2) {
        drawBresenhamLine(line.x1, line.y1, line.x2, line.y2, line.type);
    } else if (line.type == 3) {  // Solid Bold Line
        glLineWidth(3.0);
        glBegin(GL_LINES);
        glVertex2i(line.x1, line.y1);
        glVertex2i(line.x2, line.y2);
        glEnd();
        glLineWidth(1.0);
    }

    printf("Line drawn from (%d, %d) to (%d, %d)\n", (line.x1 - 400), (line.y1 - 300), (line.x2 -
400), (line.y2 - 300));
}

// Function to display all drawn lines
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);

    // Draw X and Y axes
    glBegin(GL_LINES);
    glVertex2f(800, 300);
    glVertex2f(0, 300);
    glEnd();

    glBegin(GL_LINES);
```

```
    glVertex2f(400, 600);
    glVertex2f(400, 0);
    glEnd();

    // Ticks on X and Y axes
    for (int i = 1; i < 40; i++) {
        glBegin(GL_LINES);
        glVertex2f((20 * i), 295);
        glVertex2f((20 * i), 305);
        glEnd();
    }

    for (int i = 1; i < 30; i++) {
        glBegin(GL_LINES);
        glVertex2f(395, (20 * i));
        glVertex2f(405, (20 * i));
        glEnd();
    }

    // Draw all lines
    for (size_t i = 0; i < lines.size(); i++) {
        drawLine(lines[i]);
    }

    glFlush();
}

// Mouse function to handle line drawing
void mouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        xStart = x;
        yStart = 600 - y;  // Convert to OpenGL's coordinate system
        isDrawing = true;
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
        Line newLine;
        newLine.x1 = xStart;
        newLine.y1 = yStart;
        newLine.x2 = x;
        newLine.y2 = 600 - y;
        newLine.type = lineType;
        newLine.color[0] = currentColor[0];
        newLine.color[1] = currentColor[1];
        newLine.color[2] = currentColor[2];

        lines.push_back(newLine);  // Add line to the list
        isDrawing = false;
```

```
        glutPostRedisplay();
    }
}

// Keyboard function for undo feature
void keyboard(unsigned char key, int, int) {
    if (key == 'u' || key == 'U') {
        if (!lines.empty()) {
            lines.pop_back();
            glutPostRedisplay();
        }
    }
}

// Function to handle button clicks
void buttonClick(int choice) {
    if (choice == 4) {
        if (!lines.empty()) lines.pop_back();
    } else {
        lineType = choice;
    }
    glutPostRedisplay();
}

// Function to handle color selection
void colorMenu(int choice) {
    switch (choice) {
        case 0: currentColor[0] = 1.0; currentColor[1] = 0.0; currentColor[2] = 0.0; break; // Red
        case 1: currentColor[0] = 0.0; currentColor[1] = 0.0; currentColor[2] = 1.0; break; // Blue
        case 2: currentColor[0] = 1.0; currentColor[1] = 1.0; currentColor[2] = 0.0; break; //
Yellow
        case 3: currentColor[0] = 0.0; currentColor[1] = 1.0; currentColor[2] = 1.0; break; // Cyan
        case 4: currentColor[0] = 1.0; currentColor[1] = 0.0; currentColor[2] = 1.0; break; //
Magenta
        case 5: currentColor[0] = 1.0; currentColor[1] = 0.5; currentColor[2] = 0.0; break; //
Orange
    }
    glutPostRedisplay();
}

// Menu setup for line selection
void createMenu() {
    int colorSubMenu = glutCreateMenu(colorMenu);
    glutAddMenuEntry("Red", 0);
    glutAddMenuEntry("Blue", 1);
    glutAddMenuEntry("Yellow", 2);
    glutAddMenuEntry("Cyan", 3);
```

```
    glutAddMenuEntry("Magenta", 4);
    glutAddMenuEntry("Orange", 5);

    int mainMenu = glutCreateMenu(buttonClick);
    glutAddMenuEntry("Simple Line", 0);
    glutAddMenuEntry("Dotted Line", 1);
    glutAddMenuEntry("Dashed Line", 2);
    glutAddMenuEntry("Solid Line (Bold)", 3);
    glutAddMenuEntry("Undo", 4);
    glutAddSubMenu("Select Color", colorSubMenu);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

// Initialization
void init() {
    glClearColor(1, 1, 1, 1);
    gluOrtho2D(0, 800, 0, 600);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Line Drawing with Mouse (Bresenham)");
    init();
    createMenu();
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}
```

Breshenem Circle Drawing Algorithm (Adv)

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

// Structure to store a shape's data
struct Shape {
   int x1, y1; // Coordinates for the shape's center
   int type;   // 0 = Simple Circle, 1 = Dotted Circle, 2 = Bold Circle
   float color[3]; // RGB values
   int radius;  // Radius for circle
};

std::vector<Shape> shapes; // Stores drawn shapes
int circleType = 0; // Default: Simple Circle
float currentColor[3] = {1.0, 0.0, 0.0}; // Default: Red
int xStart, yStart; // Mouse click start points
bool isDrawing = false; // Flag for mouse drag

// Function to draw a simple circle using Bresenham's Circle Algorithm
void drawBresenhamCircle(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++;
   }
}

// Function to draw different types of circles
void drawCircle(const Shape& shape) {
   glColor3fv(shape.color);
```

```
    if (shape.type == 0) {
        // Simple Circle or Bold Circle
//        glLineWidth(5.0);
        glBegin(GL_POINTS);
        drawBresenhamCircle(shape.x1, shape.y1, shape.radius);
        glEnd();
        glLineWidth(1.0);
    }
    else if (shape.type == 1) {
        // Dotted Circle
        glBegin(GL_POINTS);
        int x = 0, y = shape.radius;
        int d = 3 - 2 * shape.radius;
        int count = 0;
        while (x <= y) {
            if (count % 2 == 0) {
                glVertex2i(shape.x1 + x, shape.y1 + y);
                glVertex2i(shape.x1 - x, shape.y1 + y);
                glVertex2i(shape.x1 + x, shape.y1 - y);
                glVertex2i(shape.x1 - x, shape.y1 - y);
                glVertex2i(shape.x1 + y, shape.y1 + x);
                glVertex2i(shape.x1 - y, shape.y1 + x);
                glVertex2i(shape.x1 + y, shape.y1 - x);
                glVertex2i(shape.x1 - y, shape.y1 - x);
            }
            if (d < 0) {
                d += 4 * x + 6;
            } else {
                d += 4 * (x - y) + 10;
                y--;
            }
            x++;
            count++;
        }
        glEnd();
    }
    system("cls");
    printf("Circle drawn with centre (%d, %d) and radius %d", (shape.x1 - 400), (shape.y1 -
300), shape.radius);

}

// Function to display all drawn shapes
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_LINES);
```

```
    glVertex2f(800, 300);
    glVertex2f(0, 300);
    glEnd();
    glBegin(GL_LINES);
    glVertex2f(400, 600);
    glVertex2f(400, 0);
    glEnd();
    for (int i = 1; i < 40; i++){
        glBegin(GL_LINES);
        glVertex2f((20 * i), 295);
        glVertex2f((20 * i), 305);
        glEnd();
    }
    for (int i = 1; i < 30; i++){
        glBegin(GL_LINES);
        glVertex2f(395, (20 * i));
        glVertex2f(405, (20 * i));
        glEnd();
    }
    for (size_t i = 0; i < shapes.size(); i++) {
        drawCircle(shapes[i]);
    }
    glFlush();
}

// Mouse function to handle shape drawing
void mouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        xStart = x;
        yStart = 600 - y;
        isDrawing = true;
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
        // Calculate radius for the circle
        if (circleType >= 0) {
            int radius = sqrt(pow(x - xStart, 2) + pow((600 - y) - yStart, 2));
            shapes.push_back({xStart, yStart, circleType, {currentColor[0], currentColor[1],
currentColor[2]}, radius});
        }
        isDrawing = false;
        glutPostRedisplay();
    }
}

// Keyboard function for undo feature
void keyboard(unsigned char key, int, int) {
    if (key == 'u' || key == 'U') {
```

```
        if (!shapes.empty()) {
            shapes.pop_back();
            glutPostRedisplay();
        }
    }
}

// Function to handle button clicks for shape type selection
void buttonClick(int choice) {
    if (choice == 3) {
        if (!shapes.empty()) shapes.pop_back();
    } else {
        circleType = choice;
    }
    glutPostRedisplay();
}

// Function to handle color selection
void colorMenu(int choice) {
    switch (choice) {
        case 0: currentColor[0] = 1.0; currentColor[1] = 0.0; currentColor[2] = 0.0; break; // Red
        case 1: currentColor[0] = 0.0; currentColor[1] = 0.0; currentColor[2] = 1.0; break; // Blue
        case 2: currentColor[0] = 1.0; currentColor[1] = 1.0; currentColor[2] = 0.0; break; //
Yellow
        case 3: currentColor[0] = 0.0; currentColor[1] = 1.0; currentColor[2] = 0.0; break; //
Green
        case 4: currentColor[0] = 1.0; currentColor[1] = 0.0; currentColor[2] = 1.0; break; //
Magenta
        case 5: currentColor[0] = 1.0; currentColor[1] = 0.5; currentColor[2] = 0.0; break; //
Orange
        case 6: currentColor[0] = 0.0; currentColor[1] = 0.0; currentColor[2] = 0.0; break; //
Black
    }
    glutPostRedisplay();
}

// Menu setup for circle type and color selection
void createMenu() {
    int colorSubMenu = glutCreateMenu(colorMenu);
    glutAddMenuEntry("Red", 0);
    glutAddMenuEntry("Blue", 1);
    glutAddMenuEntry("Yellow", 2);
    glutAddMenuEntry("Green", 3);
    glutAddMenuEntry("Magenta", 4);
    glutAddMenuEntry("Orange", 5);
    glutAddMenuEntry("Black", 6);
```

```
    int mainMenu = glutCreateMenu(buttonClick);
    glutAddMenuEntry("Simple Circle", 0);
    glutAddMenuEntry("Dotted Circle", 1);
//    glutAddMenuEntry("Bold Circle", 2);
    glutAddMenuEntry("Undo", 3);
    glutAddSubMenu("Select Color", colorSubMenu);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

// OpenGL initialization
void init() {
    glClearColor(1, 1, 1, 1); // White background
    gluOrtho2D(0, 800, 0, 600); // Coordinate system
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Circle Drawing with Mouse");
    init();
    createMenu();
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    glutMainLoop();
    return 0;
}
```

2D Transformations (Adv)

```
#include <stdio.h>
#include <windows.h>
#include <GL/glut.h>
#include <vector>
#include <iostream>
#include <cmath>

using namespace std;

vector<vector<int> > vertices; // Store user input vertices
vector<vector<int> > result;   // Store transformed vertices

void takeInputShape() {
   int no_of_vertices;
   cout << "Enter the number of vertices: ";
   cin >> no_of_vertices;

   vertices.resize(no_of_vertices, vector<int>(3, 1));
   for (int i = 0; i < no_of_vertices; i++) {
      cout << "Enter coordinates (x, y) for vertex " << i + 1 << ": ";
      cin >> vertices[i][0] >> vertices[i][1];

      // Convert to OpenGL coordinates
      vertices[i][0] += 400;
      vertices[i][1] += 300;
   }
   result = vertices;
}

void translate() {
   int tx, ty;
   cout << "Enter translation values (tx ty): ";
   cin >> tx >> ty;

   for (int i = 0; i < vertices.size(); i++) {
      result[i][0] = result[i][0] + tx;
      result[i][1] = result[i][1] + ty;
   }
   glutPostRedisplay();
}

void rotate() {
   int px, py;
   double angle;
   cout << "Enter rotation point (px py): ";
```

```
    cin >> px >> py;
    cout << "Enter rotation angle (degrees): ";
    cin >> angle;

    angle = angle * M_PI / 180.0; // Convert to radians

    // Translate the point to origin
    for (int i = 0; i < result.size(); i++) {
        result[i][0] -= (px + 400);
        result[i][1] -= (py + 300);
    }

    // Apply rotation around the origin
    for (int i = 0; i < result.size(); i++) {
        int x = result[i][0];
        int y = result[i][1];
        result[i][0] = x * cos(angle) - y * sin(angle);
        result[i][1] = x * sin(angle) + y * cos(angle);
    }

    // Translate the point back to its original position
    for (int i = 0; i < result.size(); i++) {
        result[i][0] += (px + 400);
        result[i][1] += (py + 300);
    }

    glutPostRedisplay();
}

void scale() {
    float sx, sy;
    cout << "Enter scaling factors (sx sy): ";
    cin >> sx >> sy;

    for (int i = 0; i < result.size(); i++) {
        result[i][0] = 400 + (result[i][0] - 400) * sx;
        result[i][1] = 300 + (result[i][1] - 300) * sy;
    }
    glutPostRedisplay();
}

void shear() {
    float shx, shy;
    cout << "Enter shear values (shx shy): ";
    cin >> shx >> shy;

    for (int i = 0; i < result.size(); i++) {
```

```
        int x = result[i][0] - 400;
        int y = result[i][1] - 300;
        result[i][0] = 400 + x + shx * y;
        result[i][1] = 300 + y + shy * x;
    }
    glutPostRedisplay();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw X and Y axes
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2f(800, 300);
    glVertex2f(0, 300);
    glVertex2f(400, 600);
    glVertex2f(400, 0);
    glEnd();

    // Draw grid ticks
    for (int i = 1; i < 40; i++) {
        glBegin(GL_LINES);
        glVertex2f((20 * i), 295);
        glVertex2f((20 * i), 305);
        glEnd();
    }
    for (int i = 1; i < 30; i++) {
        glBegin(GL_LINES);
        glVertex2f(395, (20 * i));
        glVertex2f(405, (20 * i));
        glEnd();
    }

    // Draw original shape (Red)
    if (!vertices.empty()) {
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < vertices.size(); i++) {
            glVertex2f(vertices[i][0], vertices[i][1]);
        }
        glEnd();
    }

    // Draw transformed shape (Blue)
    if (!result.empty()) {
        glColor3f(0.0, 0.0, 1.0);
```

```
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < result.size(); i++) {
            glVertex2f(result[i][0], result[i][1]);
        }
        glEnd();
    }

    glFlush();
}

void initOpenGL() {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    gluOrtho2D(0, 800, 0, 600);
}

void keyboard(unsigned char key, int x, int y) {
    if (key == 't' || key == 'T') translate();
    else if (key == 'r' || key == 'R') rotate();
    else if (key == 's' || key == 'S') scale();
    else if (key == 'h' || key == 'H') shear();
}

int main(int argc, char **argv) {
    takeInputShape();

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("2D Transformations");

    initOpenGL();
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);

    glutMainLoop();
    return 0;
}
```

Flood fill Algorithm (Adv)

```
#include <GL/glut.h>
#include <iostream>

int windowWidth = 500, windowHeight = 500;

// Set the pixel size (for drawing)
void setPixel(int x, int y, float r, float g, float b) {
   glColor3f(r, g, b);
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
   glFlush();
}

// Get the color of a pixel
void getPixelColor(int x, int y, float* color) {
   glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, color);
}

// Compare two colors (helper function)
bool isSameColor(float* color1, float* color2) {
   return (color1[0] == color2[0] &&
         color1[1] == color2[1] &&
         color1[2] == color2[2]);
}

// Flood fill function (4-connected)
void floodFill(int x, int y, float* oldColor, float* newColor) {
   float currentColor[3];
   getPixelColor(x, y, currentColor);

   if (isSameColor(currentColor, oldColor)) {
      setPixel(x, y, newColor[0], newColor[1], newColor[2]);
      floodFill(x + 1, y, oldColor, newColor);
      floodFill(x - 1, y, oldColor, newColor);
      floodFill(x, y + 1, oldColor, newColor);
      floodFill(x, y - 1, oldColor, newColor);
   }
}

// Mouse callback
void mouse(int button, int state, int x, int y) {
   if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
      float oldColor[3];
      float newColor[3] = {1.0f, 0.0f, 0.0f}; // red
```

```
        getPixelColor(x, windowHeight - y, oldColor);
        if (!isSameColor(oldColor, newColor)) {
            floodFill(x, windowHeight - y, oldColor, newColor);
        }
    }
}

// Display callback (draw boundary shape)
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw a square boundary (black)
    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINE_LOOP);
    glVertex2i(100, 100);
    glVertex2i(200, 100);
    glVertex2i(200, 200);
    glVertex2i(100, 200);
    glEnd();

    glFlush();
}

// Initialization
void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0); // white background
    glColor3f(0.0, 0.0, 0.0);
    gluOrtho2D(0, windowWidth, 0, windowHeight);
}

// Main
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow("Flood Fill in OpenGL");

    init();
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    glutMainLoop();
    return 0;
}
```

 # Boundary Fill Algorithm (Adv)

```
#include <GL/glut.h>
#include <iostream>

int windowWidth = 500, windowHeight = 500;

// Draw a single pixel
void setPixel(int x, int y, float r, float g, float b) {
   glColor3f(r, g, b);
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
   glFlush();
}

// Get color of a pixel
void getPixelColor(int x, int y, float* color) {
   glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, color);
}

// Check if two colors match
bool isSameColor(float* color1, float* color2) {
   return (color1[0] == color2[0] &&
         color1[1] == color2[1] &&
         color1[2] == color2[2]);
}

// Boundary fill algorithm (4-connected)
void boundaryFill(int x, int y, float* fillColor, float* boundaryColor) {
   float currentColor[3];
   getPixelColor(x, y, currentColor);

   if (!isSameColor(currentColor, boundaryColor) &&
```

```
        !isSameColor(currentColor, fillColor)) {
        setPixel(x, y, fillColor[0], fillColor[1], fillColor[2]);

        boundaryFill(x + 1, y, fillColor, boundaryColor);
        boundaryFill(x - 1, y, fillColor, boundaryColor);
        boundaryFill(x, y + 1, fillColor, boundaryColor);
        boundaryFill(x, y - 1, fillColor, boundaryColor);
    }
}

// Mouse click callback
void mouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        float boundaryColor[3] = {0.0f, 0.0f, 0.0f}; // Black boundary
        float fillColor[3] = {1.0f, 0.0f, 0.0f};     // Red fill

        int fx = x;
        int fy = windowHeight - y;  // Flip Y-axis

        boundaryFill(fx, fy, fillColor, boundaryColor);
    }
}

// Draw shape to be filled
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw a square boundary (black)
    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINE_LOOP);
    glVertex2i(100, 100);
    glVertex2i(300, 100);
    glVertex2i(300, 300);
    glVertex2i(100, 300);
    glEnd();

    glFlush();
}

// OpenGL init
void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0); // White background
    glColor3f(0.0, 0.0, 0.0);         // Default draw color
    gluOrtho2D(0, windowWidth, 0, windowHeight);
}

// Main
```

```
int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(windowWidth, windowHeight);
   glutCreateWindow("Boundary Fill in OpenGL");

   init();
   glutDisplayFunc(display);
   glutMouseFunc(mouse);
   glutMainLoop();

   return 0;
}
```

Boundary and Flood Fill Combined (Adv)

```
#include <GL/glut.h>
#include <iostream>

int windowWidth = 600, windowHeight = 600;

// Set pixel color at (x, y)
void setPixel(int x, int y, float r, float g, float b) {
   glColor3f(r, g, b);
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
   glFlush();
}

// Get pixel color at (x, y)
void getPixelColor(int x, int y, float* color) {
   glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, color);
}

// Check if two colors are the same
bool isSameColor(float* c1, float* c2) {
   return (c1[0] == c2[0] && c1[1] == c2[1] && c1[2] == c2[2]);
}

// Boundary Fill (4-connected)
void boundaryFill(int x, int y, float* fillColor, float* boundaryColor) {
   float current[3];
   getPixelColor(x, y, current);
   if (!isSameColor(current, boundaryColor) && !isSameColor(current, fillColor)) {
      setPixel(x, y, fillColor[0], fillColor[1], fillColor[2]);
      boundaryFill(x + 1, y, fillColor, boundaryColor);
      boundaryFill(x - 1, y, fillColor, boundaryColor);
      boundaryFill(x, y + 1, fillColor, boundaryColor);
      boundaryFill(x, y - 1, fillColor, boundaryColor);
   }
}

// Flood Fill (4-connected)
void floodFill(int x, int y, float* oldColor, float* newColor) {
   float current[3];
   getPixelColor(x, y, current);
   if (isSameColor(current, oldColor)) {
      setPixel(x, y, newColor[0], newColor[1], newColor[2]);
      floodFill(x + 1, y, oldColor, newColor);
      floodFill(x - 1, y, oldColor, newColor);
      floodFill(x, y + 1, oldColor, newColor);
```

```
        floodFill(x, y - 1, oldColor, newColor);
    }
}

// Mouse click callback
void mouse(int button, int state, int x, int y) {
    if (state == GLUT_DOWN) {
        int fx = x;
        int fy = windowHeight - y;

        float black[3] = {0.0f, 0.0f, 0.0f};       // Boundary color (black)
        float red[3] = {1.0f, 0.0f, 0.0f};         // Boundary fill color (red)
        float green[3] = {0.0f, 1.0f, 0.0f};       // Flood fill color (green)
        float clickedColor[3];

        getPixelColor(fx, fy, clickedColor);

        if (button == GLUT_LEFT_BUTTON) {
            floodFill(fx, fy, clickedColor, green); // Flood fill on left click
        } else if (button == GLUT_RIGHT_BUTTON) {
            boundaryFill(fx, fy, red, black); // Boundary fill on right click
        }
    }
}

// Draw shapes
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw square (for boundary fill)
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(100, 100);
    glVertex2i(200, 100);
    glVertex2i(200, 200);
    glVertex2i(100, 200);
    glEnd();

    // Draw triangle (for flood fill)
    glBegin(GL_LINE_LOOP);
    glVertex2i(300, 100);
    glVertex2i(400, 100);
    glVertex2i(350, 200);
    glEnd();

    glFlush();
}
```

```
// Setup OpenGL
void init() {
   glClearColor(1.0, 1.0, 1.0, 1.0); // white background
   gluOrtho2D(0, windowWidth, 0, windowHeight);
}

// Main function
int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(windowWidth, windowHeight);
   glutCreateWindow("Flood Fill (Left Click) & Boundary Fill (Right Click)");

   init();
   glutDisplayFunc(display);
   glutMouseFunc(mouse);
   glutMainLoop();

   return 0;
}
```