LAB:- Impementation of OPENGL (Bsc.CSIT 3rd CG) Submission date: 20th April: 2025

Setup OpenGL in Visual Studio 2022 for C/C++ Development

https://www.youtube.com/watch?v=HzFatL3WT6g&list=PLCQCs1yWTdjamLKbgn_qpzTGuQ2egLCfx&index=27

LAB: To be familiar with OpenGL, implement the following.

1) 2D- Geometric primitive
   a. Color commands
   b. Callback function

```
#include <GLFW/glfw3.h>
#include <iostream>

// Vertex data for a triangle
float vertices[] = {
    0.0f, 0.5f, 0.0f,  // Top vertex
   -0.5f, -0.5f, 0.0f,  // Bottom-left vertex
    0.5f, -0.5f, 0.0f   // Bottom-right vertex
};
// Callback function for window resize
void framebuffer_size_callback(GLFWwindow* window, int width, int height) {
    glViewport(0, 0, width, height);
}

int main() {
    // Initialize GLFW
    if (!glfwInit()) {
        std::cerr << "Failed to initialize GLFW" << std::endl;
        return -1;
    }

    // Create a windowed mode window and its OpenGL context
    GLFWwindow* window = glfwCreateWindow(640, 480, "Simple Triangle", nullptr, nullptr);
    if (!window) {
        std::cerr << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }

    // Make the window's context current
    glfwMakeContextCurrent(window);

    // Set OpenGL viewport size
    glViewport(0, 0, 640, 480);

    // Main loop
    while (!glfwWindowShouldClose(window)) {
        // Clear the screen with a dark color
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        // Start drawing a triangle
        glBegin(GL_TRIANGLES);
        glColor3f(1.0f, 0.0f, 0.0f);  // Red color
        glVertex2f(vertices[0], vertices[1]);  // Top vertex
        glVertex2f(vertices[3], vertices[4]);  // Bottom-left vertex
        glVertex2f(vertices[6], vertices[7]);  // Bottom-right vertex
        glEnd();

        // Swap the front and back buffers
        glfwSwapBuffers(window);

        // Poll for events
        glfwPollEvents();
    }

    // Clean up and terminate
    glfwDestroyWindow(window);
    glfwTerminate();
    return 0;
}
```

LAB2: To be familiar with OpenGL, implement the breshanham algorithm.

```
#include <GLFW/glfw3.h>
#include <iostream>

// Bresenham's Line Drawing Algorithm
void drawLine(int x1, int y1, int x2, int y2) {
    int dx = x2 - x1;
    int dy = y2 - y1;
    int dx2 = 2 * dx;
    int dy2 = 2 * dy;

    int x = x1;
    int y = y1;

    if (dx == 0) {
        if (y1 > y2) std::swap(y1, y2);
        for (int i = y1; i <= y2; i++) {
            glBegin(GL_POINTS);
            glVertex2f(x / 640.0f * 2 - 1, i / 480.0f * 2 - 1);  // Normalize the coordinates
            glEnd();
        }
        return;
    }

    if (dy == 0) {
        if (x1 > x2) std::swap(x1, x2);
        for (int i = x1; i <= x2; i++) {
            glBegin(GL_POINTS);
            glVertex2f(i / 640.0f * 2 - 1, y / 480.0f * 2 - 1);  // Normalize the coordinates
            glEnd();
        }
        return;
    }

    bool steep = abs(dy) > abs(dx);

    // Swap x and y if the line is steep
    if (steep) {
```

```cpp
        std::swap(x, y);
        std::swap(dx, dy);
        std::swap(dx2, dy2);
    }

    int xEnd = x2;
    int yEnd = y2;
    if (x1 > x2) {
        std::swap(x1, x2);
        std::swap(y1, y2);
    }

    int p = dx2 - dy;
    for (int i = x1; i <= x2; i++) {
        int currentX = x;
        int currentY = y;

        if (steep) {
            glBegin(GL_POINTS);
            glVertex2f(currentY / 640.0f * 2 - 1, currentX / 480.0f * 2 - 1);  // Normalize the coordinates
            glEnd();
        }
        else {
            glBegin(GL_POINTS);
            glVertex2f(currentX / 640.0f * 2 - 1, currentY / 480.0f * 2 - 1);  // Normalize the coordinates
            glEnd();
        }

        if (p > 0) {
            if (steep)
                y += (y2 > y1 ? 1 : -1);
            else
                x += (x2 > x1 ? 1 : -1);
            p -= 2 * dy;
        }
        if (p <= 0) {
            if (steep)
                x += (x2 > x1 ? 1 : -1);
            else
                y += (y2 > y1 ? 1 : -1);
            p += 2 * dx;
        }
    }
}

int main() {
    // Initialize GLFW
    if (!glfwInit()) {
        std::cerr << "Failed to initialize GLFW" << std::endl;
        return -1;
    }

    // Create a windowed mode window and its OpenGL context
    GLFWwindow* window = glfwCreateWindow(640, 480, "Bresenham's Line Drawing", nullptr, nullptr);
    if (!window) {
        std::cerr << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
```

```
glfwMakeContextCurrent(window);

// Set the OpenGL viewport size
glViewport(0, 0, 640, 480);

// Set the background color to black
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

// Set the color for the line (red)
glColor3f(1.0f, 0.0f, 0.0f);

// Main loop
while (!glfwWindowShouldClose(window)) {
    // Clear the screen
    glClear(GL_COLOR_BUFFER_BIT);

    // Call Bresenham algorithm to draw a line
    drawLine(100, 100, 500, 400);  // Start and end points of the line

    // Swap buffers and poll events
    glfwSwapBuffers(window);
    glfwPollEvents();
}

// Clean up and terminate
glfwDestroyWindow(window);
glfwTerminate();
return 0;
```