```h
#ifndef POLIGONO_POLIGONO_H
#define POLIGONO_POLIGONO_H
#pragma once

#include <memory>
#include <vector>
#include "shape.h"
#include "Vertex.h"

#ifdef _WIN32
#include <glad/glad.h>
#else
#include <OpenGL/gl3.h>
#endif

class Poligono;
using PoligonoPtr = std::shared_ptr<Poligono>;

class Poligono final : public Shape {
    GLuint m_vao = 0;
    GLuint m_vbo = 0;
    GLuint m_ebo = 0;
    GLsizei m_index_count = 0;

    explicit Poligono(const std::vector<Vertex> &vertices, const std::vector<uint32_t>& indices);

public:
    static PoligonoPtr Make(const std::vector<Vertex> &vertices, const std::vector<uint32_t>& indices);

    ~Poligono() override;

    void Draw() override;
};

#endif //POLIGONO_POLIGONO_H

```

```cpp
#include "poligono.h"
#include <iostream>

PoligonoPtr Poligono::Make(const std::vector<Vertex> &vertices, const std::vector<uint32_t> &indices) {
    return PoligonoPtr(new Poligono(vertices, indices));
}

Poligono::Poligono(const std::vector<Vertex> &vertices, const std::vector<uint32_t> &indices) {
    if (vertices.size() < 3 || indices.empty()) return;

    m_index_count = static_cast<GLsizei>(indices.size());

    glGenVertexArrays(1, &m_vao);
    glGenBuffers(1, &m_vbo);
    glGenBuffers(1, &m_ebo);

    glBindVertexArray(m_vao);
    glBindBuffer(GL_ARRAY_BUFFER, m_vbo);
    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), vertices.data(), GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_ebo);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(uint32_t), indices.data(), GL_STATIC_DRAW);

    glVertexAttribPointer(
        0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), reinterpret_cast<void *>(offsetof(Vertex, position))
    );
    glEnableVertexAttribArray(0);

    glVertexAttribPointer(
        1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), reinterpret_cast<void *>(offsetof(Vertex, color))
    );
    glEnableVertexAttribArray(1);

    glBindVertexArray(0);
    std::cout << "Poligono robusto criado (VAO ID: " << m_vao << ")" << std::endl;
}

Poligono::~Poligono() {
    std::cout << "Deletando Poligono robusto (VAO ID: " << m_vao << ")" << std::endl;

    // Libera todos os recursos da GPU
    glDeleteBuffers(1, &m_vbo);
    glDeleteBuffers(1, &m_ebo);
    glDeleteVertexArrays(1, &m_vao);
}

void Poligono::Draw() {
    if (m_vao == 0 || m_index_count == 0) return;

    glBindVertexArray(m_vao);
    glDrawElements(GL_TRIANGLES, m_index_count, GL_UNSIGNED_INT, 0);
```

```
52      glBindVertexArray(0);
53  }
54
```

File - C:\Users\gabri\OneDrive\PUC\CompGraf\poligono\main.cpp

```cpp
#ifdef _WIN32
#include <windows.h>
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#else
#include <OpenGL/gl3.h>
#include <GLFW/glfw3.h>
#endif

#include "error.h"
#include "poligono.h"
#include "shader.h"
#include "Vertex.h"

#include <cstdio>

static PoligonoPtr poly;
static ShaderPtr shd;

static void error(const int code, const char *msg) {
    printf("GLFW error %d: %s\n", code, msg);
    glfwTerminate();
    exit(1);
}

static void keyboard(GLFWwindow *window, const int key, int scancode, const int action, int mods) {
    if (key == GLFW_KEY_Q && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, GLFW_TRUE);
}

static void resize(GLFWwindow *win, const int width, const int height) {
    glViewport(0, 0, width, height);
}

static void initialize() {
    glClearColor(0.95f, 0.95f, 0.95f, 1.0f);

    const std::vector<Vertex> vertices = {
        //        Posição (x, y, z)                    Cor (r, g, b)
        {{0.8f, 0.8f, 0.0f}, {1.0f, 1.0f, 0.5f}}, // Vértice 0: Amarelo
        {{-0.7f, 0.7f, 0.0f}, {0.6f, 0.8f, 1.0f}}, // Vértice 1: Azul claro
        {{-0.1f, 0.1f, 0.0f}, {0.9f, 0.5f, 0.5f}}, // Vértice 2: Vermelho
        {{-0.9f, -0.8f, 0.0f}, {0.6f, 1.0f, 0.6f}}, // Vértice 3: Verde
        {{0.9f, -0.6f, 0.0f}, {0.8f, 0.6f, 1.0f}} // Vértice 4: Roxo
    };
    const std::vector<uint32_t> indices = {
        2, 4, 0,
        2, 3, 4,
        2, 1, 3
    };

    poly = Poligono::Make(vertices, indices);

    shd = Shader::Make();
    shd->AttachVertexShader("shaders/vertex.glsl");
    shd->AttachFragmentShader("shaders/fragment.glsl");
    shd->Link();

```

```
    Error::Check("initialize");
}

static void display(GLFWwindow *win) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    shd->UseProgram();
    poly->Draw();

    Error::Check("display");
}


int main() {
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);

    glfwSetErrorCallback(error);

    GLFWwindow *win = glfwCreateWindow(800, 800, "Teste do Poligono", nullptr
, nullptr);
    glfwSetFramebufferSizeCallback(win, resize);
    glfwSetKeyCallback(win, keyboard);

    glfwMakeContextCurrent(win);

#ifdef __glad_h_
    if (!gladLoadGLLoader(reinterpret_cast<GLADloadproc>(glfwGetProcAddress
))) {
        printf("Failed to initialize GLAD OpenGL context\n");
        exit(1);
    }
#endif

    printf("OpenGL version: %s\n", glGetString(GL_VERSION));

    initialize();

    while (!glfwWindowShouldClose(win)) {
        display(win);
        glfwSwapBuffers(win);
        glfwPollEvents();
    }

    glfwTerminate();
    return 0;
}

```