/\*

#include <iostream>

#include <fstream>

#include <vector>

#include <GL/glew.h>

#include <GLFW/glfw3.h>

#include <glm/glm.hpp>

#include <glm/gtc/matrix\_transform.hpp>

#include <glm/gtc/type\_ptr.hpp>

// Структура для хранения сечений

struct Section {

int Z; // координата Z сечения

std::vector<std::pair<int, int>> points; // координаты X и Y всех точек в сечении

};

// Структура для заголовка файла

struct Header {

uint16\_t version;

int N; // количество сечений

uint16\_t pointsPerSection;

uint16\_t logIndex;

double datetime;

uint16\_t frontDiameter;

uint16\_t middleDiameter;

uint16\_t backDiameter;

uint16\_t tipDiameter;

uint16\_t logLength;

uint8\_t curvature;

int16\_t curvatureDirection;

int16\_t taper;

int16\_t taperBase;

float physicalVolume;

uint16\_t flags;

float encoderPulsePrice;

// Резерв: байты с 130 до конца шапки.

};

// Глобальные переменные для управления

float rotationX = 0.0f;

float rotationY = 0.0f;

float scale = 1.0f;

std::vector<Section> sections; // Все сечения

// Функция для чтения заголовка из файла

Header readHeader(std::ifstream& file) {

Header header;

file.read(reinterpret\_cast<char\*>(&header.version), sizeof(header.version));

file.read(reinterpret\_cast<char\*>(&header.N), sizeof(header.N));

file.read(reinterpret\_cast<char\*>(&header.pointsPerSection), sizeof(header.pointsPerSection));

file.read(reinterpret\_cast<char\*>(&header.logIndex), sizeof(header.logIndex));

file.read(reinterpret\_cast<char\*>(&header.datetime), sizeof(header.datetime));

file.read(reinterpret\_cast<char\*>(&header.frontDiameter), sizeof(header.frontDiameter));

file.read(reinterpret\_cast<char\*>(&header.middleDiameter), sizeof(header.middleDiameter));

file.read(reinterpret\_cast<char\*>(&header.backDiameter), sizeof(header.backDiameter));

file.read(reinterpret\_cast<char\*>(&header.tipDiameter), sizeof(header.tipDiameter));

file.read(reinterpret\_cast<char\*>(&header.logLength), sizeof(header.logLength));

file.read(reinterpret\_cast<char\*>(&header.curvature), sizeof(header.curvature));

file.read(reinterpret\_cast<char\*>(&header.curvatureDirection), sizeof(header.curvatureDirection));

file.read(reinterpret\_cast<char\*>(&header.taper), sizeof(header.taper));

file.read(reinterpret\_cast<char\*>(&header.taperBase), sizeof(header.taperBase));

file.read(reinterpret\_cast<char\*>(&header.physicalVolume), sizeof(header.physicalVolume));

file.read(reinterpret\_cast<char\*>(&header.flags), sizeof(header.flags));

file.read(reinterpret\_cast<char\*>(&header.encoderPulsePrice), sizeof(header.encoderPulsePrice));

// Пропускаем резервные байты

file.seekg(130, std::ios::cur);

return header;

}

// Функция для чтения сечений из файла

std::vector<Section> readSections(std::ifstream& file, int N) {

std::vector<Section> sections;

for (int i = 0; i < N; ++i) {

Section section;

file.read(reinterpret\_cast<char\*>(&section.Z), sizeof(section.Z));

uint16\_t M;

file.read(reinterpret\_cast<char\*>(&M), sizeof(M)); // Количество точек в сечении

for (int j = 0; j < M; ++j) {

int16\_t x, y;

file.read(reinterpret\_cast<char\*>(&x), sizeof(x));

file.read(reinterpret\_cast<char\*>(&y), sizeof(y));

section.points.emplace\_back(x, y);

}

sections.push\_back(section);

}

return sections;

}

// Шейдеры

const char\* vertexShaderSource = R"(

#version 330 core

layout(location = 0) in vec3 aPos;

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

void main() {

gl\_Position = projection \* view \* model \* vec4(aPos, 1.0);

}

)";

const char\* fragmentShaderSource = R"(

#version 330 core

out vec4 FragColor;

void main() {

FragColor = vec4(1.0, 1.0, 1.0, 1.0);

}

)";

// Обработка ввода для вращения и масштабирования

void processInput(GLFWwindow\* window) {

if (glfwGetKey(window, GLFW\_KEY\_UP) == GLFW\_PRESS)

rotationX += 0.01f;

if (glfwGetKey(window, GLFW\_KEY\_DOWN) == GLFW\_PRESS)

rotationX -= 0.01f;

if (glfwGetKey(window, GLFW\_KEY\_LEFT) == GLFW\_PRESS)

rotationY += 0.01f;

if (glfwGetKey(window, GLFW\_KEY\_RIGHT) == GLFW\_PRESS)

rotationY -= 0.01f;

if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)

scale += 0.01f;

if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS)

scale -= 0.01f;

}

// Функция для компиляции шейдеров

GLuint compileShader(GLenum type, const char\* source) {

GLuint shader = glCreateShader(type);

glShaderSource(shader, 1, &source, nullptr);

glCompileShader(shader);

return shader;

}

int main() {

// Открытие бинарного файла

std::ifstream file("model.lprf", std::ios::binary);

if (!file.is\_open()) {

std::cerr << "Не удалось открыть файл.\n";

return -1;

}

// Чтение заголовка

Header header = readHeader(file);

std::cout << "Файл версии: " << header.version << "\n";

std::cout << "Количество сечений: " << header.N << "\n";

// Чтение данных сечений

sections = readSections(file, header.N);

file.close();

// Инициализация GLFW

if (!glfwInit()) {

std::cerr << "Failed to initialize GLFW\n";

return -1;

}

GLFWwindow\* window = glfwCreateWindow(800, 600, "3D Object Viewer", nullptr, nullptr);

if (!window) {

std::cerr << "Failed to create GLFW window\n";

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

// Инициализация GLEW

if (glewInit() != GLEW\_OK) {

std::cerr << "Failed to initialize GLEW\n";

return -1;

}

// Здесь создаются VBO и VAO на основе считанных данных

// Основной цикл

while (!glfwWindowShouldClose(window)) {

processInput(window);

// Очистка экрана

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Рендеринг модели бревна на основе данных из файла

// Обновляем экран

glfwSwapBuffers(window);

glfwPollEvents();

}

glfwTerminate();

return 0;

}

\*/