**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 2**

Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий.

Студент: Лошманов Юрий Андреевич

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата: 05.10.2021

Оценка:

Москва, 2021

1. Постановка задачи

Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна.

Вариант многогранника:

4. Клин

1. Описание программы

Программа представляет собой окно, в виде игры, курсор автоматически захватывается. Реализована камеру, которую можно вращать и перемещать.

Управление:

W - вперёд

S - назад

A - влево

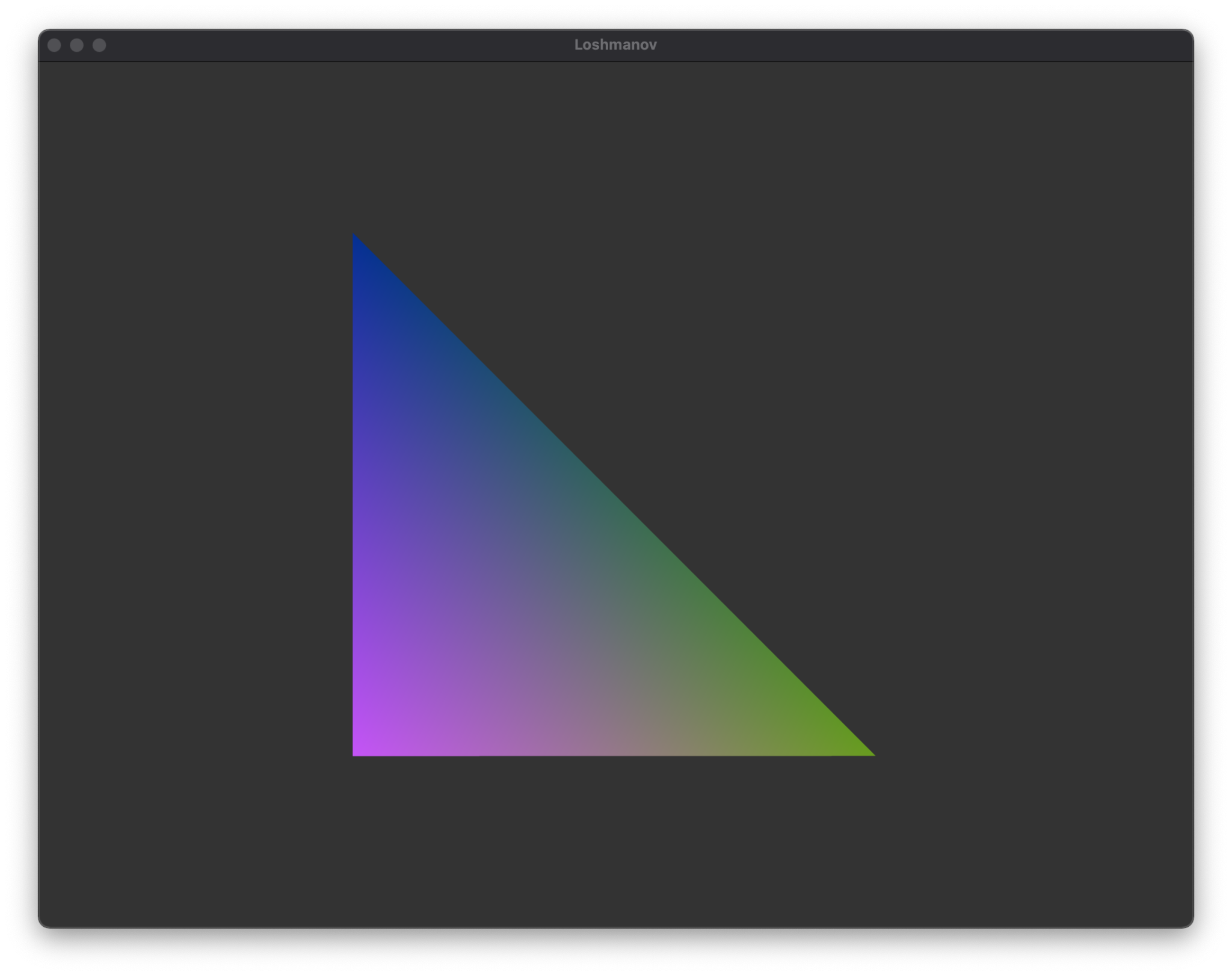
D - вправо

Space - вверх

Shift - вниз

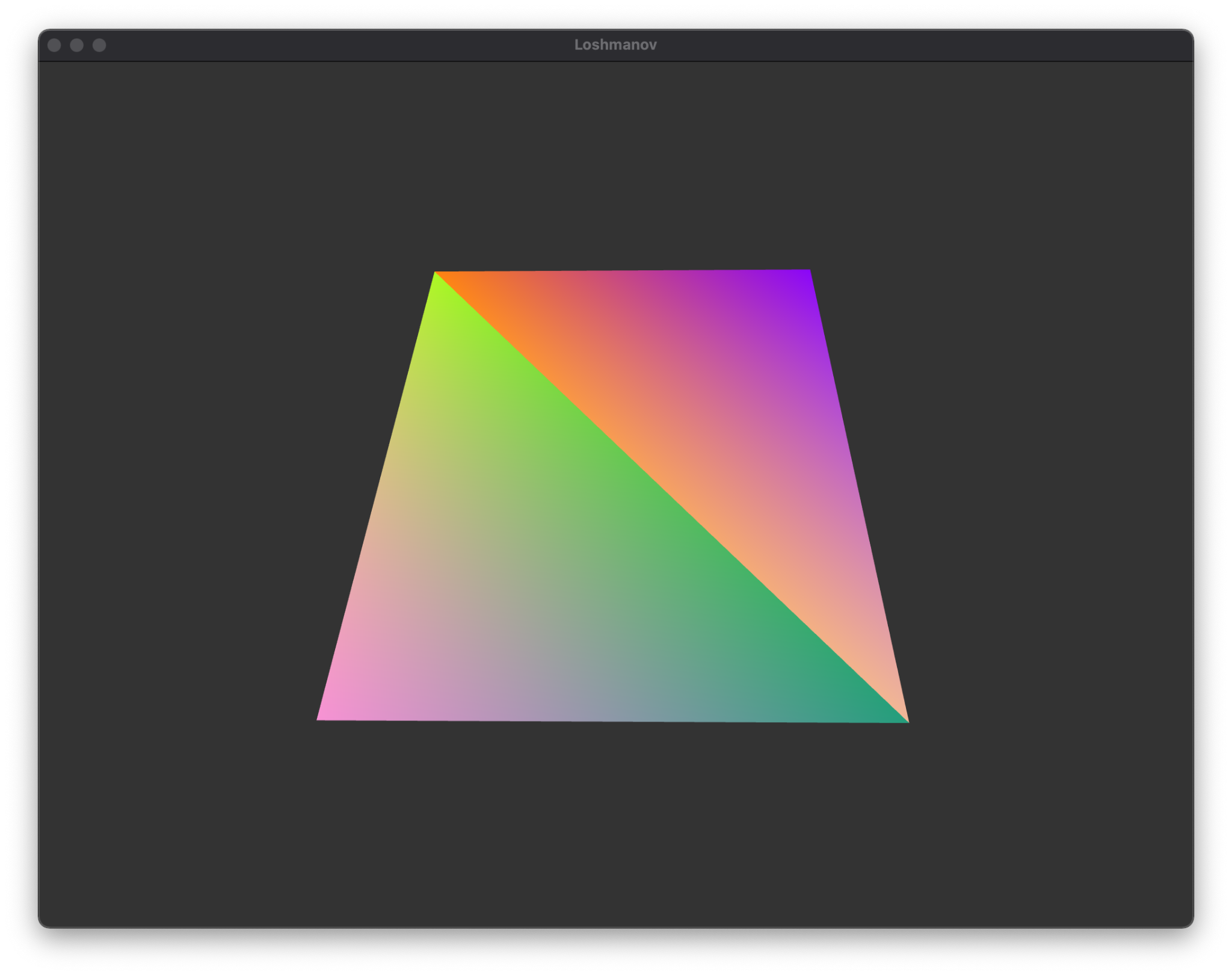
1. Набор тестов
2. Вид сбоку
3. Вид спереди
4. Вид с верхнего угла
5. Вид с нижнего угла
6. Результаты выполнения тестов

1. Вид сбоку



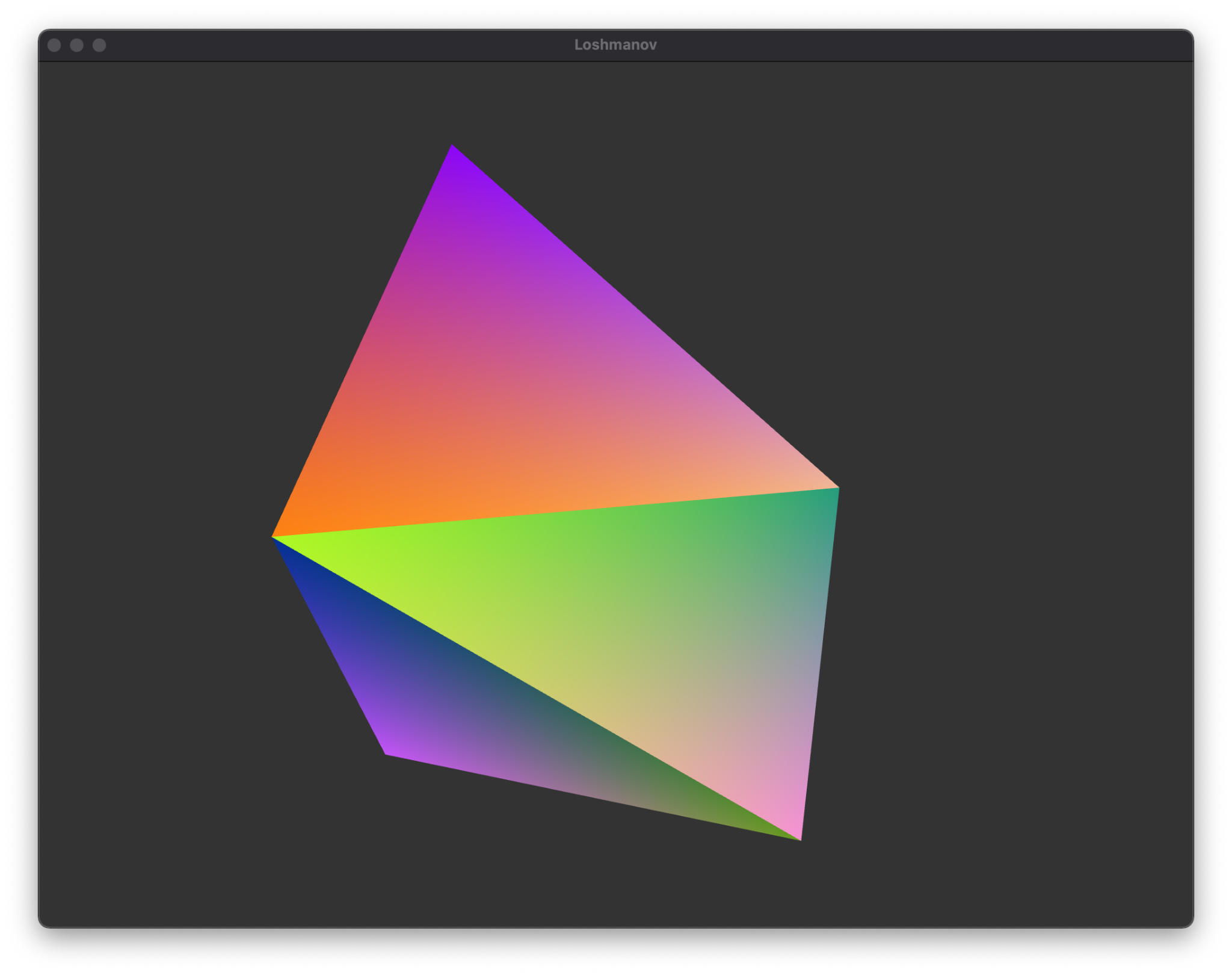
Видно только один треугольник. Невидимые треугольники удалены.

2. Вид спереди



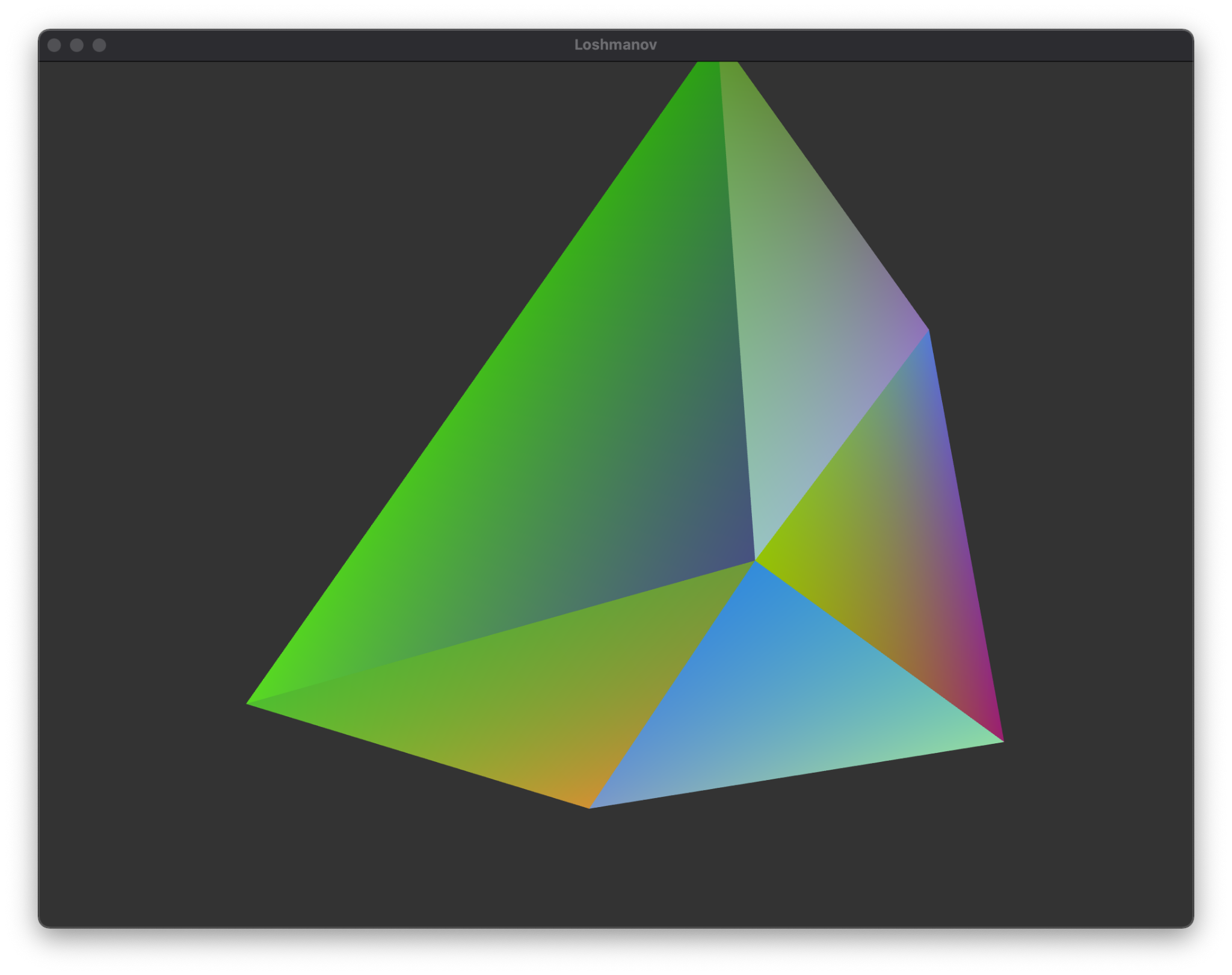
Видно 2 треугольника, составляющий одну плоскость. Невидимые треугольники удалены.

3. Вид с верхнего угла



Видны треугольники 2 сторон. Невидимые треугольники удалены.

4. Вид с нижнего угла



Видны треугольники 3 сторон. Невидимые треугольники удалены.

1. Листинг программы

**App.hpp**

//

// Created by Yury Loshmanov on 30.09.2021.

// Group М8О-206Б-19

//

#ifndef LAB2\_APP\_HPP

#define LAB2\_APP\_HPP

#include <algorithm>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <vector>

#include <cstdio>

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

#include <glm/glm.hpp>

#include <GL/glew.h>

#include <GLFW/glfw3.h>

namespace kg {

class App {

public:

static GLuint loadShaders(const std::string& vertexFilePath, const std::string& fragmentFilePath);

void computeMatricesFromInputs();

int run();

protected:

GLFWwindow \*window;

glm::mat4 viewMatrix;

glm::mat4 projectionMatrix;

glm::vec3 position = glm::vec3(0, 0, 5);

float horizontalAngle = 3.14f;

float verticalAngle = 0.0f;

float initialFoV = 45.0f;

float speed = 3.0f;

float mouseSpeed = 0.005f;

};

}

#endif //LAB2\_APP\_HPP

**App.cpp**

//

// Created by Yury Loshmanov on 30.09.2021.

// Group М8О-206Б-19

//

#include "../include/App.hpp"

GLuint kg::App::loadShaders(const std::string& vertexFilePath, const std::string& fragmentFilePath) {

GLuint vertexShaderId = glCreateShader(GL\_VERTEX\_SHADER);

GLuint fragmentShaderId = glCreateShader(GL\_FRAGMENT\_SHADER);

std::string vertexShaderCode;

std::ifstream vertexShaderStream(vertexFilePath, std::ios::in);

if (vertexShaderStream.is\_open()) {

std::stringstream ss;

ss << vertexShaderStream.rdbuf();

vertexShaderCode = ss.str();

vertexShaderStream.close();

} else {

std::cerr << "Impossible to open " << vertexFilePath << std::endl;

return 1;

}

std::string fragmentShaderCode;

std::ifstream fragmentShaderStream(fragmentFilePath, std::ios::in);

if (fragmentShaderStream.is\_open()) {

std::stringstream ss;

ss << fragmentShaderStream.rdbuf();

fragmentShaderCode = ss.str();

fragmentShaderStream.close();

}

GLint result = GL\_FALSE;

int infoLogLength;

std::cout << "Compiling shader : " << vertexFilePath << std::endl;

auto vertexSourcePointer = vertexShaderCode.c\_str();

glShaderSource(vertexShaderId, 1, &vertexSourcePointer, nullptr);

glCompileShader(vertexShaderId);

glGetShaderiv(vertexShaderId, GL\_COMPILE\_STATUS, &result);

glGetShaderiv(vertexShaderId, GL\_INFO\_LOG\_LENGTH, &infoLogLength);

if (infoLogLength > 0) {

std::vector<char> vertexShaderErrorMessage(infoLogLength + 1);

glGetShaderInfoLog(vertexShaderId, infoLogLength, nullptr, &vertexShaderErrorMessage[0]);

std::cout << &vertexShaderErrorMessage[0] << std::endl;

}

std::cout << "Compiling shader : " << fragmentFilePath << std::endl;

auto fragmentSourcePointer = fragmentShaderCode.c\_str();

glShaderSource(fragmentShaderId, 1, &fragmentSourcePointer, nullptr);

glCompileShader(fragmentShaderId);

glGetShaderiv(fragmentShaderId, GL\_COMPILE\_STATUS, &result);

glGetShaderiv(fragmentShaderId, GL\_INFO\_LOG\_LENGTH, &infoLogLength);

if (infoLogLength > 0) {

std::vector<char> FragmentShaderErrorMessage(infoLogLength + 1);

glGetShaderInfoLog(fragmentShaderId, infoLogLength, nullptr, &FragmentShaderErrorMessage[0]);

std::cout << &FragmentShaderErrorMessage[0] << std::endl;

}

std::cout << "Linking program\n" << std::endl;

GLuint programId = glCreateProgram();

glAttachShader(programId, vertexShaderId);

glAttachShader(programId, fragmentShaderId);

glLinkProgram(programId);

glGetProgramiv(programId, GL\_LINK\_STATUS, &result);

glGetProgramiv(programId, GL\_INFO\_LOG\_LENGTH, &infoLogLength);

if (infoLogLength > 0) {

std::vector<char> ProgramErrorMessage(infoLogLength + 1);

glGetProgramInfoLog(programId, infoLogLength, nullptr, &ProgramErrorMessage[0]);

std::cout << &ProgramErrorMessage[0] << std::endl;

}

glDetachShader(programId, vertexShaderId);

glDetachShader(programId, fragmentShaderId);

glDeleteShader(vertexShaderId);

glDeleteShader(fragmentShaderId);

return programId;

}

void kg::App::computeMatricesFromInputs() {

static auto lastTime = glfwGetTime();

auto currentTime = glfwGetTime();

auto deltaTime = static\_cast<float>(currentTime - lastTime);

double xMousePos, yMousePos;

glfwGetCursorPos(window, &xMousePos, &yMousePos);

glfwSetCursorPos(window, 1024.0f / 2.0f, 768.0f / 2.0f);

auto verticalAngleInc = mouseSpeed \* static\_cast<float>(768.0f / 2.0f - yMousePos);

static auto termAngle = glm::pi<float>() / -2;

if (verticalAngle + verticalAngleInc <= termAngle || verticalAngle + verticalAngleInc >= -termAngle) {

return;

}

horizontalAngle += mouseSpeed \* static\_cast<float>(1024.0f / 2.0f - xMousePos);

verticalAngle += mouseSpeed \* static\_cast<float>(768.0f / 2.0f - yMousePos);

glm::vec3 direction(

cos(verticalAngle) \* sin(horizontalAngle),

sin(verticalAngle),

cos(verticalAngle) \* cos(horizontalAngle)

);

glm::vec3 right = glm::vec3(

sin(horizontalAngle - 3.14f / 2.0f),

0,

cos(horizontalAngle - 3.14f / 2.0f)

);

auto up = glm::cross(right, direction);

// Move forward

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

position += direction \* deltaTime \* speed;

}

// Move backward

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

position -= direction \* deltaTime \* speed;

}

// Strafe right

if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS) {

position += right \* deltaTime \* speed;

}

// Strafe left

if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS) {

position -= right \* deltaTime \* speed;

}

// Move up

if (glfwGetKey(window, GLFW\_KEY\_SPACE) == GLFW\_PRESS) {

position.y += deltaTime \* speed;

}

// Move down

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

position.y -= deltaTime \* speed;

}

auto FoV = initialFoV;

projectionMatrix = glm::perspective(glm::radians(FoV), 4.0f / 3.0f, 0.1f, 100.0f);

viewMatrix = glm::lookAt(position, position + direction, up);

lastTime = currentTime;

}

int kg::App::run() {

if (!glfwInit()) {

std::cerr << "Failed to initialize GLFW" << std::endl;

return 1;

}

glfwWindowHint(GLFW\_SAMPLES, 4);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE);

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

window = glfwCreateWindow(1024, 768, "Loshmanov", nullptr, nullptr);

if (window == nullptr) {

std::cerr << "Failed to open GLFW window." << std::endl;

glfwTerminate();

return 2;

}

glfwMakeContextCurrent(window);

glewExperimental = true;

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

std::cerr << "Failed to initialize GLEW" << std::endl;

glfwTerminate();

return 3;

}

glfwSetInputMode(window, GLFW\_STICKY\_KEYS, GL\_TRUE);

glfwSetInputMode(window, GLFW\_CURSOR, GLFW\_CURSOR\_DISABLED);

glfwPollEvents();

glfwSetCursorPos(window, 1024.0f / 2.0f, 768.0f / 2.0f);

glClearColor(0.2f, 0.2f, 0.2f, 0.0f);

glEnable(GL\_DEPTH\_TEST);

glDepthFunc(GL\_LESS);

glEnable(GL\_CULL\_FACE);

GLuint VertexArrayID;

glGenVertexArrays(1, &VertexArrayID);

glBindVertexArray(VertexArrayID);

auto programID = loadShaders("TransformVertexShader.vertexshader", "ColorFragmentShader.fragmentshader");

GLuint matrixId = glGetUniformLocation(programID, "MVP");

GLuint textureId = glGetUniformLocation(programID, "myTextureSampler");

static const GLfloat g\_vertex\_buffer\_data[] = {

-1.0f, -1.0f, -1.0f,

-1.0f, -1.0f, 1.0f,

-1.0f, 1.0f, 1.0f,

1.0f, -1.0f, 1.0f,

-1.0f, -1.0f, -1.0f,

1.0f, -1.0f, -1.0f,

-1.0f, -1.0f, -1.0f,

-1.0f, 1.0f, 1.0f,

-1.0f, 1.0f, -1.0f,

1.0f, -1.0f, 1.0f,

-1.0f, -1.0f, 1.0f,

-1.0f, -1.0f, -1.0f,

-1.0f, 1.0f, 1.0f,

-1.0f, -1.0f, 1.0f,

1.0f, -1.0f, 1.0f,

-1.0f, 1.0f, 1.0f,

1.0f, -1.0f, 1.0f,

1.0f, -1.0f, -1.0f,

-1.0f, 1.0f, 1.0f,

1.0f, -1.0f, -1.0f,

-1.0f, 1.0f, -1.0f,

-1.0f, -1.0f, -1.0f,

-1.0f, 1.0f, -1.0f,

1.0f, -1.0f, -1.0f

};

static const GLfloat g\_color\_buffer\_data[] = {

0.583f, 0.771f, 0.014f,

0.609f, 0.115f, 0.436f,

0.327f, 0.483f, 0.844f,

0.822f, 0.569f, 0.201f,

0.435f, 0.602f, 0.223f,

0.310f, 0.747f, 0.185f,

0.597f, 0.770f, 0.761f,

0.559f, 0.436f, 0.730f,

0.359f, 0.583f, 0.152f,

0.483f, 0.596f, 0.789f,

0.559f, 0.861f, 0.639f,

0.195f, 0.548f, 0.859f,

0.014f, 0.184f, 0.576f,

0.771f, 0.328f, 0.970f,

0.406f, 0.615f, 0.116f,

0.676f, 0.977f, 0.133f,

0.971f, 0.572f, 0.833f,

0.140f, 0.616f, 0.489f,

0.997f, 0.513f, 0.064f,

0.945f, 0.719f, 0.592f,

0.543f, 0.021f, 0.978f,

0.279f, 0.317f, 0.505f,

0.167f, 0.620f, 0.077f,

0.347f, 0.857f, 0.137f,

};

GLuint vertexBuffer;

glGenBuffers(1, &vertexBuffer);

glBindBuffer(GL\_ARRAY\_BUFFER, vertexBuffer);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(g\_vertex\_buffer\_data), g\_vertex\_buffer\_data, GL\_STATIC\_DRAW);

GLuint colorBuffer;

glGenBuffers(1, &colorBuffer);

glBindBuffer(GL\_ARRAY\_BUFFER, colorBuffer);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(g\_color\_buffer\_data), g\_color\_buffer\_data, GL\_STATIC\_DRAW);

do {

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

glUseProgram(programID);

computeMatricesFromInputs();

glm::mat4 modelMatrix = glm::mat4(1.0);

glm::mat4 MVP = projectionMatrix \* viewMatrix \* modelMatrix;

glUniformMatrix4fv(static\_cast<GLint>(matrixId), 1, GL\_FALSE, &MVP[0][0]);

glActiveTexture(GL\_TEXTURE0);

glUniform1i(static\_cast<GLint>(textureId), 0);

glEnableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, vertexBuffer);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 0, nullptr);

glEnableVertexAttribArray(1);

glBindBuffer(GL\_ARRAY\_BUFFER, colorBuffer);

glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 0, nullptr);

glDrawArrays(GL\_TRIANGLES, 0, 8 \* 3);

glDisableVertexAttribArray(0);

glDisableVertexAttribArray(1);

glfwSwapBuffers(window);

glfwPollEvents();

} while (glfwGetKey(window, GLFW\_PRESS && glfwWindowShouldClose(window) == 0);

glDeleteBuffers(1, &vertexBuffer);

glDeleteBuffers(1, &colorBuffer);

glDeleteProgram(programID);

glDeleteTextures(1, &textureId);

glDeleteVertexArrays(1, &VertexArrayID);

glfwTerminate();

return 0;

}

**main.cpp**

//

// Created by Yury Loshmanov on 30.09.2021.

// Group М8О-206Б-19

//

#include "../include/App.hpp"

int main() {

kg::App app;

app.run();

return 0;

}

1. Выводы

Выполнив данную лабораторную работу, я научился пользоваться библиотекой OpenGL, реализовав компьютерную графику на C++, изучил проекцию 3D изображения на 2D экран и реализовал камеру для наблюдения за клином.

ЛИТЕРАТУРА

1. Туториал по OpenGL[Электронный ресурс]URL: <https://www.opengl-tutorial.org> (Дата обращения: 21.09.2021).