|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кафедра |  | O7 |  | Информационные системы и программная инженерия |
|  |  | шифр |  | наименование кафедры, по которой выполняется работа |
| Дисциплина |  | Компьютерная геометрия и графика | | |
|  |  | наименование дисциплины | | |

|  |  |  |
| --- | --- | --- |
| ЛАБОРАТОРНАЯ РАБОТА | 5 |  |
|  | номер (при наличии) |  |
| Ложные трёхмерные объекты. | | |
| Камера. | | |
|  | | |

при наличии указать тему лабораторной работы и (или) номер варианта

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ОБУЧАЮЩИЙСЯ** | | | | | | |
| группы | | | |  | | И913Б |
|  |  | Кульга Н.И | | | | |
| подпись |  | фамилия и инициалы | | | | |
|  | | | | |
| дата сдачи | | | | |
| **ПРОВЕРИЛ** | | | | | | |
|  | | | | | | | |
| ученая степень, ученое звание, должность | | | | | | | |
|  |  | Мажайцев Е.А | | | | | |
| подпись |  | фамилия и инициалы | | | | | |
| Оценка / балльная оценка | | |  | | | |
|  | | | | |
| дата проверки | | | | |

Задание:

Задание 1.

Установите перспективу в соответствии с вариантом.

Задание 2.

С помощью загрузки моделей или готовых объектов, изобразите рисунок в соответствии с вариантом.

Задание 3. Установите камеру таким образом, чтобы она была направлена в центр вашего изображения.

Задание 4. Реализуйте перемещение камеры относительно объекта:

● Объезд камеры вокруг объекта;

● Приближение/удаление камеры относительно объекта;

Задание 5. Реализуйте движение объекта относительно камеры:

● Движение объекта влево/вправо относительно камеры;

● Вращение объекта вокруг своей оси.

Результат работы программы:

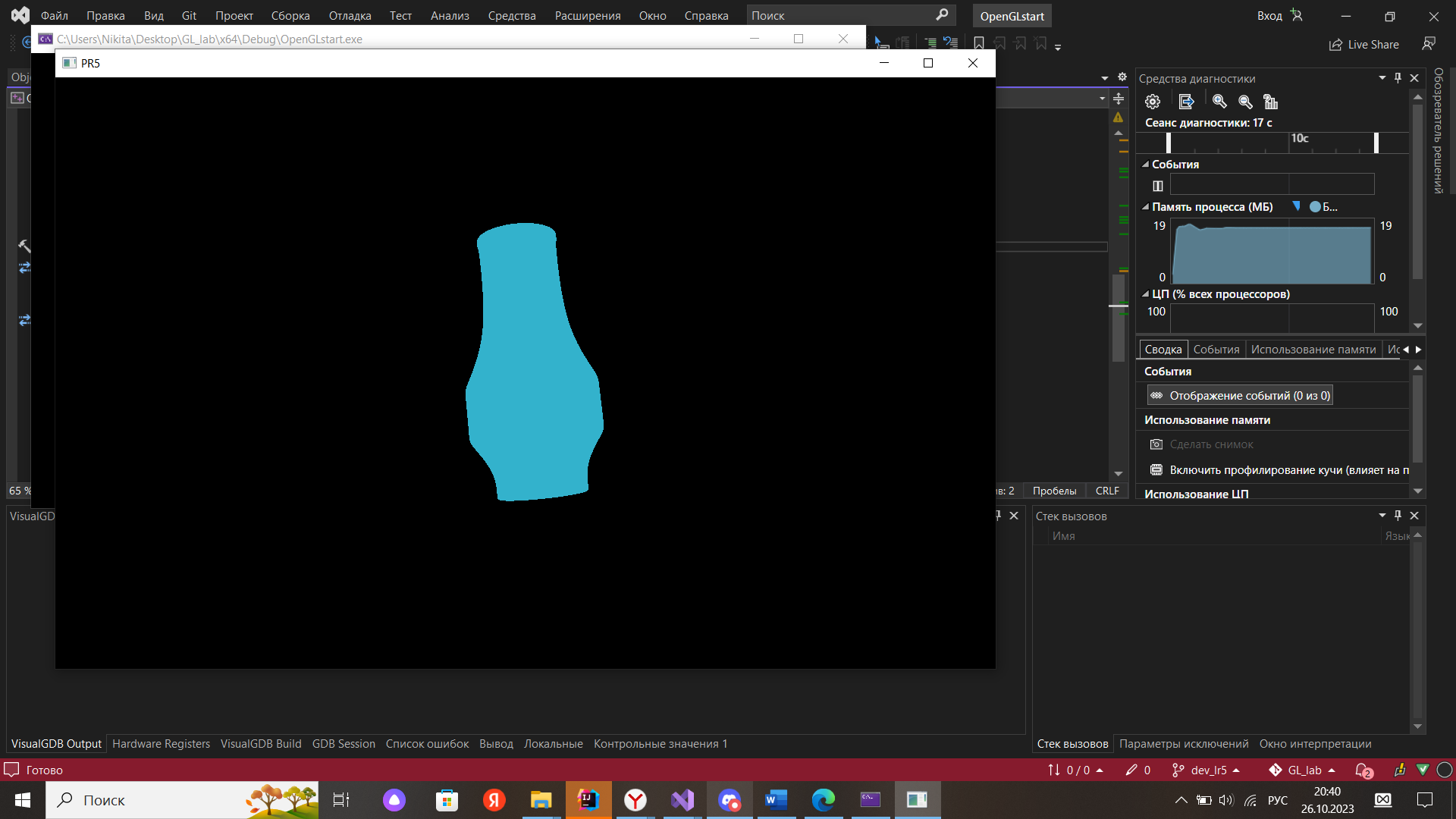


Рисунок 1 – Результат работы.

Код программы:

Main.cpp:

#include "glew.h"

#include <glfw3.h>

#include <iostream>

#include "ShaderFuncs.h"

#include "glm/glm.hpp"

#include "glm/gtc/matrix\_transform.hpp"

#include "dataPoint.h"

#include "Object.h"

#include <cmath>

int WinWidth = 1240;

int WinHeight = 780;

void glfw\_window\_size\_callback(GLFWwindow\* window, int width, int height);

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mods);

int chse;

GLfloat dist;

GLfloat ang;

GLfloat mv;

GLfloat rt;

int main() {

dist = 35.0f;

ang = 0.0f;

mv = 0.0f;

glfwInit();

GLFWwindow\* window = glfwCreateWindow(WinWidth, WinHeight, "PR5",

NULL, NULL);

glfwMakeContextCurrent(window);

glewInit();

glfwSetWindowSizeCallback(window, glfw\_window\_size\_callback);

glfwSetKeyCallback(window, key\_callback);

GLuint vs = LoadShader("VertexShader.glsl", GL\_VERTEX\_SHADER);

GLuint fs = LoadShader("FragmentShader.glsl", GL\_FRAGMENT\_SHADER);

GLuint shader\_programme = glCreateProgram();

glAttachShader(shader\_programme, vs);

glAttachShader(shader\_programme, fs);

glBindAttribLocation(shader\_programme, 0, "vertex\_position");

glBindAttribLocation(shader\_programme, 1, "vertex\_colour");

glLinkProgram(shader\_programme);

chse = 0;

OpenGLENG::Object\* pl = OpenGLENG::Object::LoadFromOBJ("v.obj", { 0.2f,0.7f,0.8f });

while (!glfwWindowShouldClose(window)) {

glfwPollEvents();

glViewport(0, 0, WinWidth, WinHeight);

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

glUseProgram(shader\_programme);

glm::mat4 trm = glm::mat4(1.0f);

trm = glm::translate(trm, { 0.0f,-5.0f,mv });

trm = glm::rotate(trm, glm::radians(0.0f), glm::vec3(1.0, 1.0, 0.0));

trm = glm::rotate(trm, rt, glm::vec3(0.0, 0.0, 1.0));

glm::vec3 scale = { 0.2f,0.2f,0.2f };

trm = glm::scale(trm, scale); // уменьшение

glm::mat4 prsp = glm::perspective(glm::radians(35.0f), (float)WinWidth / WinHeight, 0.0f, 70.0f);

glm::mat4 cam = glm::lookAt(glm::vec3(dist \* cos(ang), 0.5f, dist \* sin(ang)), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.1f, 0.0f));

pl->draw(prsp \* cam \* trm, shader\_programme);

glfwSwapBuffers(window);

}

glDeleteShader(vs);

glDeleteShader(fs);

glDeleteShader(shader\_programme);

delete pl;

glfwTerminate();

return 0;

}

void glfw\_window\_size\_callback(GLFWwindow\* window, int width, int height) {

WinWidth = width;

WinHeight = height;

}

void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mods)

{

if (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS)

glfwSetWindowShouldClose(window, 1);

else if (key == GLFW\_KEY\_DOWN && action == GLFW\_REPEAT)

dist += 0.05f;

else if (key == GLFW\_KEY\_UP && action == GLFW\_REPEAT)

dist -= 0.05f;

else if (key == GLFW\_KEY\_LEFT && action == GLFW\_REPEAT)

ang -= 0.01f;

else if (key == GLFW\_KEY\_RIGHT && action == GLFW\_REPEAT)

ang += 0.01f;

else if (key == GLFW\_KEY\_A && action == GLFW\_REPEAT)

mv += 0.01f;

else if (key == GLFW\_KEY\_D && action == GLFW\_REPEAT)

mv -= 0.01f;

else if (key == GLFW\_KEY\_W && action == GLFW\_REPEAT)

rt += 0.01f;

else if (key == GLFW\_KEY\_S && action == GLFW\_REPEAT)

rt -= 0.01f;

}

\\..........................

Shaderfuncs.h:

#pragma once

#include <iostream>

#include <string>

#include <GL/glew.h>

#include <GLFW/glfw3.h>

GLuint LoadShader(std::string filename, GLenum type);

Shaderfuncs.cpp:

#include "ShaderFuncs.h"

#include <fstream>

#include <vector>

GLuint LoadShader(std::string filename, GLenum type)

{

std::ifstream fin;

fin.open(filename);

if (fin) {

std::string textshader;

char c= fin.get();

while (c!=-1) {

textshader += c;

c = fin.get();

}

fin.close();

GLuint s = glCreateShader(type);

char\* buf = new GLchar[textshader.length() + 1];

strcpy\_s(buf, textshader.length()+1 , textshader.c\_str());

glShaderSource(s, 1, &buf, NULL);

glCompileShader(s);

GLint isCompiled = 0;

glGetShaderiv(s, GL\_COMPILE\_STATUS, &isCompiled);

if (isCompiled == GL\_FALSE)

{

GLint maxLength = 0;

glGetShaderiv(s, GL\_INFO\_LOG\_LENGTH, &maxLength);

std::vector<GLchar> errorLog(maxLength);

glGetShaderInfoLog(s, maxLength, &maxLength, &errorLog[0]);

for (GLchar c : errorLog)

std::cout << c;

}

delete[] buf;

return s;

}

else {

std::cout << "ERROR: No such file";

return -1;

}

}

FragmentShader.glsl:

#version 400

in vec3 colour;

out vec4 frag\_colour;

void main () {

frag\_colour = vec4 (colour, 1.0);

}

VertexShaner.glsl:

#version 400

in vec3 vertex\_position;

in vec3 vertex\_colour;

uniform mat4 transform;

out vec3 colour;

void main () {

colour = vertex\_colour;

gl\_Position = transform \* vec4 (vertex\_position, 1.0);

}

OBJloader.cpp

#include "OBJloader.h"

bool OBJLoad(

const char\* path,

std::vector<glm::vec3>& outVertices, //Выходной массив координат

std::vector<glm::vec2>& outTextures, //Выходной массив текстурных координат

std::vector<glm::vec3>& outNormals //Выходной массив нормалей

)

{

//Временные массивы для хранения индексов и значений из файла

std::vector< unsigned int > vertexIndices, vtIndices, normalIndices;

std::vector< glm::vec3 > temp\_vertices;

std::vector< glm::vec2 > temp\_textures;

std::vector< glm::vec3 > temp\_normals;

//Строка для хранения токенов из файла

std::string buf;

//Буфер, хранящий содержимое файла

std::stringstream sstr;

//Переменная для корректной работы с неизвестным числомвершин в грани

bool cont = false;

//Считывание файла

std::ifstream file(path, std::ios::in);

if (file.is\_open()) {

sstr << file.rdbuf();

file.close();

}

else

return false;

//Очистка выходных массивов

outNormals.clear();

outVertices.clear();

outTextures.clear();

//Попытка считать токен из файла

while (cont || sstr >> buf) {

cont = false;

//Обработка координат

if (buf == "v") {

glm::vec3 vertex;

sstr >> vertex.x;

sstr >> vertex.y;

sstr >> vertex.z;

temp\_vertices.push\_back(vertex);

}

//Обработка текстурных координат

else if (buf == "vt") {

glm::vec2 texture;

sstr >> texture.x;

sstr >> texture.y;

temp\_textures.push\_back(texture);

}

//Обработка нормалей

else if (buf == "vn") {

glm::vec3 normal;

sstr >> normal.x;

sstr >> normal.y;

sstr >> normal.z;

temp\_normals.push\_back(normal);

}

//Обработка граней

else if (buf == "f") {

//Временные массивы для хранения информации о грани

std::vector<std::string> vertexes;

std::vector<int> vertindexes;

while (!cont)

{

//Считывается токен

sstr >> buf;

//Если токен содержит данные о вершине - она записываетсяв массив

if (buf.find('/') != std::string::npos)

vertexes.push\_back(buf);

//Если токен - начало следующей строки, он сохраняется иначинается обработка грани

else

{

//Для каждой из сохранённых вершин производитсяпарсинг данных

for (std::string vert : vertexes)

{

std::replace(vert.begin(), vert.end(), '/', ' ');

std::stringstream tmpstream(vert);

int v, vt, n;

tmpstream >> v;

tmpstream >> vt;

tmpstream >> n;

//Индексы заносятся в временный массив

vertindexes.push\_back(v);

vertindexes.push\_back(vt);

vertindexes.push\_back(n);

}

//Первые три вершины заносятся вмассивы индексов

for (int i = 0; i < 3; i++)

{

vertexIndices.push\_back(vertindexes[i \* 3 + 0]);

vtIndices.push\_back(vertindexes[i \* 3 + 1]);

normalIndices.push\_back(vertindexes[i \* 3 + 2]);

}

//Дальше сложнее - если ещё остались вершины, надо и их занести

//Но надо преобразовать из веера треугольников в набор треугольников

size\_t tmpsize = vertexes.size();

if (tmpsize > 3)

{

//Для каждой из вершин добавляются три вершины, образующие треугольник

for (int i = 3; i < tmpsize; i++)

{

vertexIndices.push\_back(vertindexes[0]);

vtIndices.push\_back(vertindexes[1]);

normalIndices.push\_back(vertindexes[2]);

vertexIndices.push\_back(vertindexes[(i - 1) \* 3 + 0]);

vtIndices.push\_back(vertindexes[(i - 1) \* 3 + 1]);

normalIndices.push\_back(vertindexes[(i - 1) \* 3 + 2]);

vertexIndices.push\_back(vertindexes[i \* 3 + 0]);

vtIndices.push\_back(vertindexes[i \* 3 + 1]);

normalIndices.push\_back(vertindexes[i \* 3 + 2]);

}

}

//Чтобы не потерялся считанный токен, ставится флаг о том что он сохранён

cont = true;

}

}

}

}

// Теперь обработка массивов индексов и создание выходных массивов

// Для каждой вершины в массиве значения, соответсвующиевершине

// под указанным индексом заносятся в выходной массив

size\_t tmpsize = vertexIndices.size();

for (unsigned int i = 0; i < tmpsize; i++)

{

//Надо учесть что индексы в файле начинаются с 1!

glm::vec3 vertex = temp\_vertices[vertexIndices[i] - 1];

outVertices.push\_back(vertex);

glm::vec3 normal = temp\_normals[normalIndices[i] - 1];

outNormals.push\_back(normal);

glm::vec2 vt = temp\_textures[vtIndices[i] - 1];

outTextures.push\_back(vt);

}

return true;

}