|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кафедра |  | О7 |  | Информационные системы и технологии |
|  |  | шифр |  | наименование кафедры, по которой выполняется работа |
| Дисциплина |  | Компьютерная геометрия и графика | | |
|  |  | наименование дисциплины | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УЧЕБНО-ПРАКТИЧЕСКАЯ РАБОТА | | | | |
|  | 1 |  |  |  |
|  | номер задания (при наличии) |  |  |  |
| Общий алгоритм работы с OpenGL. Создание | | | |  |
| оконного приложения. Создание и отображение изображений. | | | |  |

при наличии указать тему учебно-практической работы и (или) номер варианта

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ОБУЧАЮЩИЙСЯ** | | | | | | |  |
| группы | | | |  | | О711Б |  |
|  |  | Шилов В.И. | | | | |  |
| подпись |  | фамилия и инициалы | | | | |  |
|  | | | | |  |  |  |
| дата сдачи | | | | |  |  |  |
| **ПРОВЕРИЛ** | | | | | | |  |
|  | | | | | | | |
| ученая степень, ученое звание, должность | | | | | | | |
|  |  | Вальштейн К.В. | | | | | |
| подпись |  | фамилия и инициалы | | | | | |
| Оценка / балльная оценка | | |  | | | |  |
|  | | | | |  |  |  |
| дата проверки | | | | |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Санкт-Петербург | | | | | | |
|  |  |  | 20 | 23 | г. |  |

Цель работы.

Создание программы-заготовки для работы с API OpenGL.

Порядок выполнения работы

1. Создайте проект в выбранной среде разработки и подключите к нему выбранные библиотеки для работы с API OpenGL

2. Создать окно для вывода изображения

3. С использованием шейдерных программ отобразить на окне три треугольника разных цветов

4. Созданный проект будет использован в дальнейшем при выполнении следующих работ.

Написанная программа:

#include <GL/glew.h> // подключение GLEW

#include <GLFW/glfw3.h> // подключение GLFW

#include <stdio.h> //Для простого консольного вывода ошибок

#include <glm/glm.hpp>

#include <iostream>

GLfloat\* get\_random\_points(int number\_of\_points)

{

GLfloat\* arr = new GLfloat[number\_of\_points];

for (int i=0;i<number\_of\_points;i++)

{

arr[i] = ((float)rand()) / (RAND\_MAX / 2) - 1;

}

return arr;

}

GLfloat\* get\_random\_colors(int number\_of\_colors)

{

GLfloat\* arr = new GLfloat[number\_of\_colors];

for (int i=0;i<number\_of\_colors;i++)

{

arr[i] = ((float)rand()) / RAND\_MAX;

}

return arr;

}

GLuint get\_vao(GLfloat points[], GLfloat colors[], int coord\_number)

{

GLuint points\_vbo = 0;

glGenBuffers(1, &points\_vbo);

glBindBuffer(GL\_ARRAY\_BUFFER, points\_vbo);

glBufferData(GL\_ARRAY\_BUFFER, coord\_number \* sizeof(GLfloat), points, GL\_STATIC\_DRAW);

GLuint colors\_vbo = 0;

glGenBuffers(1, &colors\_vbo);

glBindBuffer(GL\_ARRAY\_BUFFER, colors\_vbo);

glBufferData(GL\_ARRAY\_BUFFER, coord\_number \* sizeof(GLfloat), colors, GL\_STATIC\_DRAW);

GLuint vao = 0;

glGenVertexArrays(1, &vao);

glBindVertexArray(vao);

glBindBuffer(GL\_ARRAY\_BUFFER, points\_vbo);

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

glBindBuffer(GL\_ARRAY\_BUFFER, colors\_vbo);

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

glEnableVertexAttribArray(0);

glEnableVertexAttribArray(1);

return vao;

}

int main()

{

// Инициализация

if (!glfwInit())

{

fprintf(stderr, "ERROR: could not start GLFW3\n");

return 1;

}

GLFWwindow\* window = glfwCreateWindow(640, 480, "Triangle", NULL, NULL);

if (!window)

{

fprintf(stderr, "ERROR: could not open window with GLFW3\n");

glfwTerminate();

return 1;

}

glfwMakeContextCurrent(window); //Созданное окно выбирается как текущее

glewExperimental = GL\_TRUE; // Подключение новейшей из доступных версий OpenGL

glewInit(); //Включение GLEW

const GLubyte\* renderer = glGetString(GL\_RENDERER); // информация о графической карте

const GLubyte\* version = glGetString(GL\_VERSION); // информация о версии OpenGL

printf("Renderer: %s\n", renderer);

printf("OpenGL version supported %s\n", version);

glEnable(GL\_DEPTH\_TEST); // Включение буфера глубины

glDepthFunc(GL\_LESS); // А вот тут можно управлять его работой. Сейчас установлен режим по умолчанию

// Определение координат

int coord\_number = 9; // Количество координат вершин

GLfloat points[coord\_number] =

{

-0.5f, -0.5f, 0.0f,

0.0f, 0.5f, 0.0f,

0.5f, -0.5f, 0.0f,

};

GLfloat colors[coord\_number] =

{

1.0f, 0.0f, 0.0f,

0.0f, 1.0f, 0.0f,

0.0f, 1.0f, 0.0f,

};

GLfloat points1[coord\_number] =

{

-0.8f, 0.8f, 1.0f,

0.0f, 0.0f, 0.0f,

0.1f, -0.2f, 0.0f,

};

GLfloat colors1[coord\_number] =

{

1.0f, 1.0f, 0.0f,

0.0f, 0.0f, 1.0f,

1.0f, 0.0f, 0.0f,

};

GLuint vao = get\_vao(get\_random\_points(coord\_number), get\_random\_colors(coord\_number), coord\_number);

GLuint vao1 = get\_vao(get\_random\_points(coord\_number), get\_random\_colors(coord\_number), coord\_number);

GLuint vao2 = get\_vao(get\_random\_points(coord\_number), get\_random\_colors(coord\_number), coord\_number);

// Шейдеры

const char\* vertex\_shader\_text =

"#version 400\n"

"layout (location = 0) in vec3 vector\_position;" //Входные данные. В данном случае это координаты одной точки

"layout (location = 1) in vec3 vector\_color;"

"out vec3 colour;"

"void main() {" //Основная функция

" gl\_Position = vec4(vector\_position, 1.0);" //В данном случае простое преобразование без вычисления сложной проекции

" colour = vector\_color;"

"}";

//Фрагмент - это по сути один пиксель на экране

const char\* fragment\_shader\_text = //Фрагментный шейдер отвечает за окрашивание каждого фрагмента

"#version 400\n" //Данный шейдер должен возвращать цвет пикселя

"in vec3 colour;"

"out vec4 frag\_color;" //Цвет из 4-х координат - RGBA

"void main() {"

" frag\_color = vec4(colour, 1.0);"

"}";

// Загрузка шейдеров

GLuint vertex\_shader = glCreateShader(GL\_VERTEX\_SHADER);

glShaderSource(vertex\_shader, 1, &vertex\_shader\_text, NULL);

glCompileShader(vertex\_shader);

GLuint fragment\_shader = glCreateShader(GL\_FRAGMENT\_SHADER);

glShaderSource(fragment\_shader, 1, &fragment\_shader\_text, NULL);

glCompileShader(fragment\_shader);

GLuint shader\_programme = glCreateProgram();

glAttachShader(shader\_programme, vertex\_shader);

glAttachShader(shader\_programme, fragment\_shader);

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

glBindAttribLocation(shader\_programme, 1, "vector\_color");

glLinkProgram(shader\_programme); //Программа компонуется

// Вывод на экран

while (!glfwWindowShouldClose(window)) {

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

glUseProgram(shader\_programme); //Подключается шейдерная программа

glBindVertexArray(vao); //Подключается массив вершин

glDrawArrays(GL\_TRIANGLES, 0, coord\_number / 3); //Из массива рисуются три элемента начиная с элемента с индексом 0

glBindVertexArray(vao1);

glDrawArrays(GL\_TRIANGLES, 0, coord\_number / 3);

glBindVertexArray(vao2);

glDrawArrays(GL\_TRIANGLES, 0, coord\_number / 3);

glfwPollEvents(); //Обработка очереди событий

glfwSwapBuffers(window); //Использование двойной буферизации

}

glfwTerminate();

return 0;

}