Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Чувашский государственный университет имени И.Н. Ульянова»

ФАКУЛЬТЕТ ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедравычислительной техники

АГС.

Лабораторная работа №2.

«Использование uniform-переменных.

Преобразование вершин в GLSL шейдерах.»

Выполнил: студент гр. ИВТ-41-15

Лазарев Дмитрий

Проверил:

Ст. преп. Галибин С.В.

Чебоксары – 2018

**Задание к лабораторной работе.**

В лабораторной работе №2 необходимо:

1. Внести изменения в класс для работы с шейдером CShader, добавив вышеописанные методы для установки uniform-переменных. Для оптимизации методы должны использовать ассоциативный контейнер map.

2. В отдельном модуле реализовать класс для работы с камерой. Класс должен включать методы для получения матриц проекции и наблюдения, а так же методы для перемещения камеры. Перемещение камеры должно осуществляться по вышеописанным правилам и в соответствии с приведенным примером.

3. Реализовать вывод нескольких (не менее 5) кубиков расположенных в разных точках сцены, каждый кубик описывается своим собственным цветом и матрицей модели. Реализовывать отдельные классы или структуры для представления объектов на данном этапе не требуется.

**Текст класса для работы с камерой.**

#pragma once

#include <windows.h>

#include "stdio.h"

#include <fstream>

#include <iostream>

#include <vector>

#include "glew.h"

#include "GL/freeglut.h"

#include <glm/glm.hpp>

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

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

using namespace glm;

const float PI = 3.14;

const float THETA\_MAX = radians(85.0);

const float THETA\_MIN = radians(5.0);

class CCamera

{

// матрица проекции

mat4 ProjectionMatrix;

// матрица наблюдения

mat4 ViewMatrix;

vec3 Eye ;

vec3 Center;

vec3 Up;

float radius;

float fi;

float theta;

float Speed = 5.0;

public:

// конструктор по умолчанию

CCamera(void);

// установка матрицы проекции

void SetProjectionMatrix(float fovy, float aspect, float zNear, float zFar);

// получение матрицы проекции

mat4 GetProjectionMatrix(void);

// получить матрицу камеры (наблюдения)

mat4 GetViewMatrix(void);

// передвинуть камеру и точку наблюдения в горизонтальной плоскости oXZ

void MoveOXZ(float dForward, float dRight);

// вращение в горизонтальной и вертикальной плоскости

void Rotate(float dHorizAngle, float dVertAngle);

// приближение/удаление

void GetUp();

void Zoom(float dR);

void Saving();

};

#include "CCamera.h"

CCamera::CCamera(void)

{

std::ifstream file("Camera.txt");

if (!file.is\_open())

{

radius = 20;

theta = radians(5.0);

fi = 0;

Center = vec3(0, 0, 0);

}

else

{

file >> radius >> theta >> fi >>Center.x>>Center.y>>Center.z;

file.close();

}

Eye.x = radius \* sin(theta)\* cos(fi);

Eye.y = radius \* cos(theta);

Eye.z = radius \* sin(theta)\* sin(fi);

Up = vec3(0, 1, 0);

ViewMatrix = lookAt(Eye + Center, Center, Up);

}

mat4 CCamera::GetViewMatrix()

{

return ViewMatrix;

}

void CCamera::MoveOXZ(float dForward, float dRight)

{

dForward = dForward \* Speed;

dRight = dRight \* Speed;

vec3 VForward = normalize(-Eye);

vec3 DeltaF = vec3(VForward.x \* dForward, 0, VForward.z \* dForward);

vec3 DeltaR = normalize(cross(VForward, Up));

DeltaR = vec3(DeltaR.x \* dRight, 0, DeltaR.z \* dRight);

Center = Center + DeltaF + DeltaR;

ViewMatrix = lookAt(Eye + Center, Center, Up);

}

void CCamera::Zoom(float dR)

{

if ((0 > dR) & (radius < 40)) radius = radius - dR;

if ((0 < dR) & (radius > 3)) radius = radius - dR;

Eye.x = radius \* sin(theta) \* cos(fi);

Eye.y = radius \* cos(theta);

Eye.z = radius \* sin(theta) \* sin(fi);

ViewMatrix = lookAt(Eye + Center, Center, Up);

}

void CCamera::Rotate(float dHorizAngle, float dVertAngle)

{

bool rButton = GetAsyncKeyState(VK\_RBUTTON);

if (rButton)

{

if ((dVertAngle > 0) & (theta < THETA\_MAX)) theta += dVertAngle / 360;

if ((dVertAngle < 0) & (theta > THETA\_MIN)) theta += dVertAngle / 360;

fi += dHorizAngle / 360;

if (fi > 2 \* PI) fi -= 2 \* PI;

if (fi < 0) fi += 2 \* PI;

if (theta < THETA\_MIN) theta = THETA\_MIN;

if (theta > THETA\_MAX) theta = THETA\_MAX;

Eye.x = radius \* sin(theta) \* cos(fi);

Eye.y = radius \* cos(theta);

Eye.z = radius \* sin(theta) \* sin(fi);

ViewMatrix = lookAt(Eye + Center, Center, Up);

}

}

void CCamera::SetProjectionMatrix(float fovy, float aspect, float zNear, float zFar)

{

ProjectionMatrix = perspective(fovy, aspect, zNear, zFar);

}

mat4 CCamera::GetProjectionMatrix(void)

{

return ProjectionMatrix;

}

void CCamera::Saving(void)

{

std::ofstream file("Camera.txt");

file << radius << ' ' << theta << ' ' << fi << ' ' << Center.x << ' ' << Center.y << ' ' << Center.z;

file.close();

}

**Текст методов для установки uniform-переменных**.

ы

void CShader::SetUniform(std::string name, vec4 &value)

{

int uniformID;

auto variable = UniformVariables.find(name);

if (variable != UniformVariables.end())

{

uniformID = variable->second;

}

else

{

uniformID = glGetUniformLocation(Program, name.c\_str());

UniformVariables.insert(std::pair<std::string, int>(name, uniformID));

}

if (uniformID < 0) return;

glUseProgram(Program);

glUniform4fv(uniformID, 1, value\_ptr(value));

}

void CShader::SetUniform(std::string name, mat4 &value)

{

int uniformID;

auto variable = UniformVariables.find(name);

if (variable != UniformVariables.end())

{

uniformID = variable->second;

}

else

{

uniformID = glGetUniformLocation(Program, name.c\_str());

UniformVariables.insert(std::pair<std::string, int>(name, uniformID));

}

if (uniformID < 0) return;

glUseProgram(Program);

glUniformMatrix4fv(uniformID, 1, GL\_FALSE, value\_ptr(value));

}