ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | Щекин С.В. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7 |
| **Освоение работы с программируемыми шейдерами** |
| по дисциплине: Компьютерная графика |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4134к |  |  |  | Костяков Н.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

Цель работы

Выполнить на выбор один из вариантов: - Шейдер с сохранением данных в текстуре, - Процедурный, текстурный шейдер, - Традиционные шейдер, - Фрагментный шейдер, - Вершинный шейдер. Примеры из пакета с компилятором Cg не допускаются

Листинг

Main.cpp

/\*

Выполнить на выбор один из вариантов:

- Шейдер с сохранением данных в текстуре,

- Процедурный, текстурный шейдер,

- Традиционные шейдер,

- Фрагментный шейдер,

- Вершинный шейдер.

Примеры из пакета с компилятором Cg не допускаются

\*/

#include <iostream>

#include <cmath>

#include <GL/glew.h>

#include <GL/glut.h>

#include "shaders.h" // функции для ШЕЙДЕРОВ

#include "move.h" // функции для ПЕРЕДВИЖЕНИЯ

#include "draw.h" // функции для ПРОРИСОВКИ

#include <fstream>

#include <string>

#define TITLE "Shader"

#define W\_WIDTH 1280

#define W\_HEIGHT 720

#define FPS 60

namespace global {

float cam\_xz\_rotate = 0;

float cam\_y\_rotate = 0.2;

float cam\_zoom = 14;

float light\_xy\_rotate = 0;

float light\_y\_rotate = 0;

}

// замена while

void timer(int value) {

glutPostRedisplay();

glutTimerFunc(1000 / FPS, timer, 0);

}

// ну тут уже ОЧЕВИДНО

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DEPTH);

glutInitWindowPosition(150, 50);

glutInitWindowSize(W\_WIDTH, W\_HEIGHT);

glutCreateWindow(TITLE);

GLenum err = glewInit();

if (GLEW\_OK != err) {

std::cout << "GLEW ERROR: " << glewGetErrorString(err) << std::endl;

}

// загружаем шейдер с файла

std::string f\_shader\_str = "";

std::ifstream inp;

inp.open("shader.frag");

while (inp) {

f\_shader\_str += inp.get();

}

inp.close();

f\_shader\_str.at(f\_shader\_str.length() - 1) = '\0';

//

make\_shader(f\_shader\_str);

// передаём функции для прорисовки

glutReshapeFunc(Reshape);

glutDisplayFunc(Display);

//

glutSetKeyRepeat(GLUT\_KEY\_REPEAT\_OFF);

// передаём функции для определения нажатых клавиш

glutKeyboardFunc(keyDown);

glutKeyboardUpFunc(keyUp);

// таймер

glutTimerFunc(0, timer, 0);

// чтоб окно не закрывалось

glutMainLoop();

}

Move.h

#pragma once

/\*

Здесь реализуются функции для ПЕРЕДВИЖЕНИЯ

\*/

#include <GL/glut.h>

#include "shaders.h"

namespace global {

extern float cam\_xz\_rotate;

extern float cam\_y\_rotate;

extern float cam\_zoom;

extern float light\_xy\_rotate;

extern float light\_y\_rotate;

}

bool left = false;

bool right = false;

bool up = false;

bool down = false;

// отпущенные клавишы

void keyUp(unsigned char key, int xx, int yy) {

switch (key) {

case ('a'):

left = false;

break;

case ('d'):

right = false;

break;

case ('w'):

up = false;

break;

case ('s'):

down = false;

break;

// выход

case 27:

exit(0);

break;

}

}

// нажатые клавишы

void keyDown(unsigned char key, int xx, int yy) {

switch (key) {

case ('a'):

left = true;

break;

case ('d'):

right = true;

break;

case ('w'):

up = true;

break;

case ('s'):

down = true;

break;

case ('f'):

switch\_shader();

break;

}

}

float fraction = 0.05;

float cam\_y\_rotate\_max = 1.5;

// функция для рассчётов передвижений

void move() {

if (left) {

global::cam\_xz\_rotate -= fraction;

}

if (right) {

global::cam\_xz\_rotate += fraction;

}

if (up && (global::cam\_y\_rotate + fraction < cam\_y\_rotate\_max)) {

global::cam\_y\_rotate += fraction;

}

if (down && (global::cam\_y\_rotate - fraction > -cam\_y\_rotate\_max)) {

global::cam\_y\_rotate -= fraction;

}

}

float fraction\_zoom = 1.0;

// прокрутка колеса мыши

void MouseWheel(int button, int dir, int x, int y) {

if (dir > 0) {

global::cam\_zoom -= fraction\_zoom;

}

else {

global::cam\_zoom += fraction\_zoom;

}

}

Shaders.h

#pragma once

/\*

Здесь реализуются функции для ШЕЙДЕРОВ

\*/

#include <iostream>

#include <GL/glut.h>

#include <GL/glew.h>

#include <string>

GLuint frag\_shader, prog;

bool shader\_flag = false;

void switch\_shader() {

if (shader\_flag) {

glUseProgram(0);

}

else {

glUseProgram(prog);

}

shader\_flag = !shader\_flag;

}

void make\_shader(std::string f\_shader\_str) {

const char\* f\_shader\_char = f\_shader\_str.c\_str();

// std::cout << f\_shader\_char << std::endl;

frag\_shader = glCreateShader(GL\_FRAGMENT\_SHADER);

glShaderSource(frag\_shader, 1, &f\_shader\_char, 0);

glCompileShader(frag\_shader);

prog = glCreateProgram();

glAttachShader(prog, frag\_shader);

glLinkProgram(prog);

}

Draw.h

#pragma once

/\*

Здесь реализуются функции для ПРОРИСОВКИ

\*/

#include <iostream>

#include <GL/glut.h>

#include "move.h"

#include <string>

#include <cmath>

namespace global {

extern float cam\_xz\_rotate;

extern float cam\_y\_rotate;

extern float cam\_zoom;

extern float light\_xy\_rotate;

}

// метериалы объектов

float mat\_dif\_cube[] = { 1.0f, 1.0f, 1.0f, 1.0f };

float mat\_spec\_cube[] = { 1.0f, 1.0f, 1.0f };

float mat\_amb\_cube[] = { 1.0f, 1.0f, 1.0f };

float mat\_shininess\_cube = 0.1f \* 128;

// настройки света

// материал объекта для визуализации источника

float mat\_dif\_light[] = { 1.0f, 1.0f, 1.0f, 1.0f };

float mat\_spec\_light[] = { 1.0f, 1.0f, 1.0f };

float mat\_amb\_light[] = { 1.0f, 1.0f, 1.0f };

float mat\_shininess\_light = 0.1f \* 128;

// материал света

float matl\_dif\_light[] = { 0.9f, 0.9f, 0.9f, 1.0f };

// перемещение света

float light\_position\_fraction = 0.02;

float light\_position\_radius = 10;

float light\_tick = 0;

// центр вращения XYZ и W (если поставить не 0, то будет точечный источник света)

float light\_position[4] = { 0.0, 5.0, 0.0, 0.0 };

// функция для отображения и включения света

void set\_light(GLenum name, GLfloat\* light\_position) {

// материал света

glLightfv(name, GL\_DIFFUSE, matl\_dif\_light);

glLightfv(name, GL\_POSITION, light\_position);

glLightf(name, GL\_SPOT\_CUTOFF, 360);

// материал объекта который визуализирует источник света

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_amb\_light);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_dif\_light);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_spec\_light);

glMaterialf(GL\_FRONT, GL\_SHININESS, mat\_shininess\_light);

glTranslatef(light\_position[0], light\_position[1], light\_position[2]);

glutSolidSphere(0.2, 32, 32);

glTranslatef(-light\_position[0], -light\_position[1], -light\_position[2]);

// включаем свет

glEnable(name);

}

void Display(void) {

glLoadIdentity();

// рассчитываем передвижение для камеры

move();

// задаём параметры камере

gluLookAt(

cos(global::cam\_y\_rotate) \* sin(global::cam\_xz\_rotate) \* global::cam\_zoom,

sin(global::cam\_y\_rotate) \* global::cam\_zoom,

cos(global::cam\_y\_rotate) \* cos(global::cam\_xz\_rotate) \* global::cam\_zoom,

0, 0, 0,

0.0f, 1.0f, 0.0f

);

// фон

glClearColor(0, 0.0, 0.0, 1);

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

// без этого буквально все элементы становятся 3D (видно сквозь объекты)

glEnable(GL\_DEPTH\_TEST);

// включаем смешивание цвета (для прозрачности)

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

/\* включаем освещение \*/

glEnable(GL\_LIGHTING);

/\* настриваем источники света \*/

// light\_position\_rotate[0] = light\_position[0][0] + (cos(light\_tick) \* light\_position\_radius);

// light\_position\_rotate[1] = light\_position[0][1];

// light\_position\_rotate[2] = light\_position[0][2] + (sin(light\_tick) \* light\_position\_radius);

// set\_light(GL\_LIGHT0, light\_position\_rotate);

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_amb\_cube);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_dif\_cube);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_spec\_cube);

glMaterialf(GL\_FRONT, GL\_SHININESS, mat\_shininess\_cube);

glutSolidCube(2);

// для того чтобы поверхность нормально реагировала на свет

glEnable(GL\_AUTO\_NORMAL);

glPopMatrix();

glFlush();

}

void Reshape(int w, int h) {

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(40.0, (GLfloat)w / h, 1, 100.0);

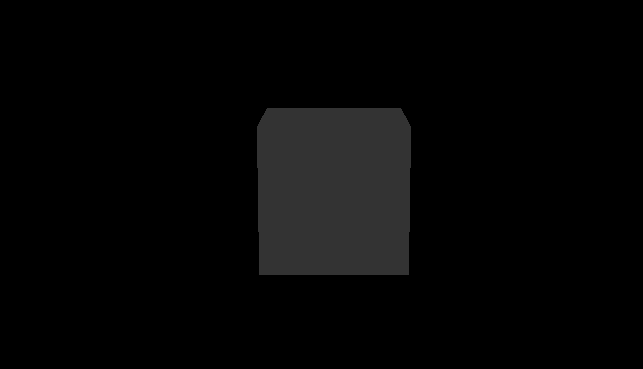
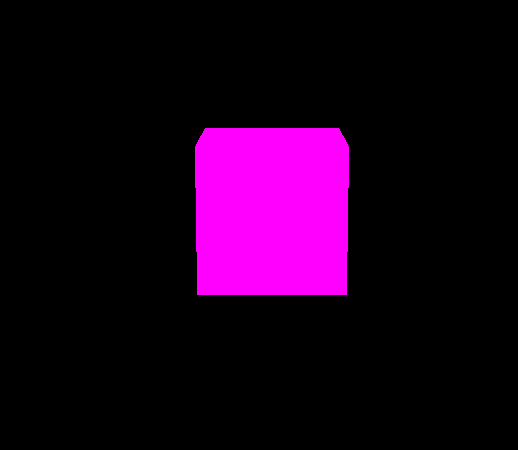
glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

}

Результат работы

Вывод

Я освоил работу с шейдерами