ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

|  |
| --- |
| ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6 |
| **Вывод трехмерных объектов с динамическим расчетом проекционных теней** |
| по дисциплине: Компьютерная графика |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

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

Санкт-Петербург 2022

Цель работы

Вывести трехмерную сцену с движущимся объектом, который отбрасывает тень на другой объект (напр. плоскость). Тень должна перемещаться вместе с движением объекта исходя из взаимного положения источника света, объекта, который отбрасывает тень и объектов, на которые тень проецируется

Листинг программы

Main.cpp

/\*

Вывести трехмерную сцену с движущимся объектом, который отбрасывает тень

на другой объект (напр. плоскость). Тень должна перемещаться вместе с

движением объекта исходя из взаимного положения источника света, объекта,

который отбрасывает тень и объектов, на которые тень проецируется

\*/

#include <iostream>

#include <cmath>

#include <GL/glut.h>

#include "models.h"

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

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

#define TITLE "Shadows"

#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;

float obj\_pos[3] = { 0, 0, 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);

init\_surface();

// 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();

}

Draw.h

#pragma once

/\*

Здесь реализуются функции для ПРОРИСОВКИ

\*/

#include <iostream>

#include <GL/glut.h>

#include <cmath>

#include "move.h"

#include "models.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;

extern float obj\_pos[3];

}

// метериалы объектов

float mat\_dif\_grass[] = { 0.7f, 0.7f, 0.7f, 1.0f };

float mat\_spec\_grass[] = { 0.7f, 0.7f, 0.7f };

float mat\_amb\_grass[] = { 0.7f, 0.7f, 0.7f };

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

float mat\_dif\_Sphere[] = { 0.7f, 0.7f, 0.7f ,1.0f };

float mat\_spec\_Sphere[] = { 0.7f, 0.7f, 0.7f };

float mat\_amb\_Sphere[] = { 0.7f, 0.7f, 0.7f };

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

float mat\_dif\_shadow[] = { 0.0f, 0.0f, 0.0f, 0.8f };

float mat\_spec\_shadow[] = { 0.0f, 0.0f, 0.0f };

float mat\_amb\_shadow[] = { 0.0f, 0.0f, 0.0f };

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

// настройки света

// материал объекта для визуализации источника

float mat\_dif\_light[] = { 0.9f, 0.9f, 0.0f, 1.0f };

float mat\_spec\_light[] = { 0.9f, 0.9f, 0.0f };

float mat\_amb\_light[] = { 0.9f, 0.9f, 0.0f };

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, 4.0, 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();

glRotated(90, 0, 0, 1);

// 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, 0.3, 0.3, 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 \*/

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

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

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

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

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

glEnable(GL\_AUTO\_NORMAL);

draw\_nurb();

// материал шара

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

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

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

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

// шар

glTranslatef(global::obj\_pos[0], 2, global::obj\_pos[2]);

glutSolidCube(1);

// материал тени

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

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

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

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

// измененгие позиции тени

glTranslatef((global::obj\_pos[0] - light\_position[0]) / light\_position[1], -1.9, (global::obj\_pos[2] - light\_position[2]) / light\_position[1]);

// маштабирование тени в зависимости от положения источника оп высоте

glScalef(1 + abs(global::obj\_pos[0] - light\_position[0]) / (light\_position[1] \* 5), 0.02, 1 + abs(global::obj\_pos[2] - light\_position[2]) / (light\_position[1] \* 5));

// отрисовка тени

glRotated(90, 0, 0, 1);

glutSolidCube(1);

// 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);

}

Move.h

#pragma once

/\*

Здесь реализуются функции для ПЕРЕДВИЖЕНИЯ

\*/

#include <GL/glut.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;

extern float obj\_pos[3];

}

bool left = false;

bool right = false;

bool up = false;

bool down = false;

bool obj\_left = false;

bool obj\_right = false;

bool obj\_up = false;

bool obj\_down = false;

// отпущенные клавишы

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

switch (key) {

// camera

case ('a'):

left = false;

break;

case ('d'):

right = false;

break;

case ('w'):

up = false;

break;

case ('s'):

down = false;

break;

// объект

case ('h'):

obj\_left = false;

break;

case ('k'):

obj\_right = false;

break;

case ('u'):

obj\_up = false;

break;

case ('j'):

obj\_down = false;

break;

// выход

case 27:

break;

}

}

// нажатые клавишы

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

switch (key) {

// camera

case ('a'):

left = true;

break;

case ('d'):

right = true;

break;

case ('w'):

up = true;

break;

case ('s'):

down = true;

break;

// объект

case ('h'):

obj\_left = true;

break;

case ('k'):

obj\_right = true;

break;

case ('u'):

obj\_up = true;

break;

case ('j'):

obj\_down = true;

break;

}

}

float fraction = 0.05;

float cam\_y\_rotate\_max = 1.5;

float obj\_fraction = 0.1;

// функция для рассчётов передвижений

void move() {

// camera

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;

}

// объект

if (obj\_left) {

global::obj\_pos[2] -= obj\_fraction;

}

if (obj\_right) {

global::obj\_pos[2] += obj\_fraction;

}

if (obj\_up) {

global::obj\_pos[0] += obj\_fraction;

}

if (obj\_down) {

global::obj\_pos[0] -= obj\_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;

}

}

Models.h

#pragma once

/\*

Здесь храянтся модели

\*/

#include <iostream>

#include <GL/glut.h>

const int size\_numb\_x = 4;

const int size\_numb\_y = 4;

GLfloat ctlpoints[size\_numb\_x][size\_numb\_y][3];

GLfloat knots[size\_numb\_x + size\_numb\_y] = {

0.0, 0.0, 0.0, 0.0,

1.0, 1.0, 1.0, 1.0

};

GLUnurbsObj\* theNurb;

// создание nurb поверхности

void init\_surface(void) {

int u, v;

int m = 20; // 2

for (u = 0; u < size\_numb\_y; u++) {

for (v = 0; v < size\_numb\_x; v++) {

ctlpoints[u][v][0] = m \* ((GLfloat)u - 1.5);

ctlpoints[u][v][1] = m \* ((GLfloat)v - 1.5);

}

}

theNurb = gluNewNurbsRenderer();

gluNurbsProperty(theNurb, GLU\_SAMPLING\_TOLERANCE, 10.0);

gluNurbsProperty(theNurb, GLU\_DISPLAY\_MODE, GLU\_FILL);

gluNurbsProperty(theNurb, GLU\_AUTO\_LOAD\_MATRIX, false);

}

// отрисовка nurb поверхности

void draw\_nurb() {

gluBeginSurface(theNurb);

glRotatef(-90.0, 1.0, 0.0, 0.0);

gluNurbsSurface(theNurb,

size\_numb\_x + size\_numb\_y, knots, size\_numb\_x + size\_numb\_y, knots,

size\_numb\_x \* 3, 3, &ctlpoints[0][0][0],

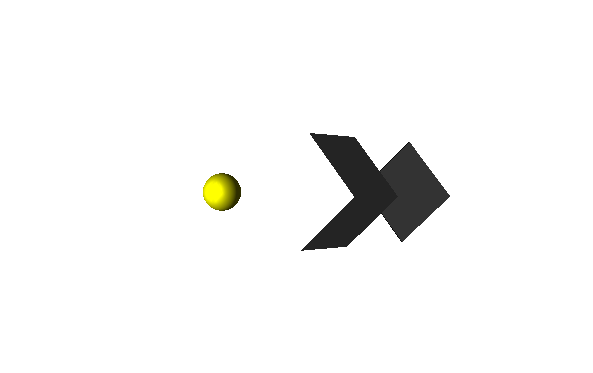
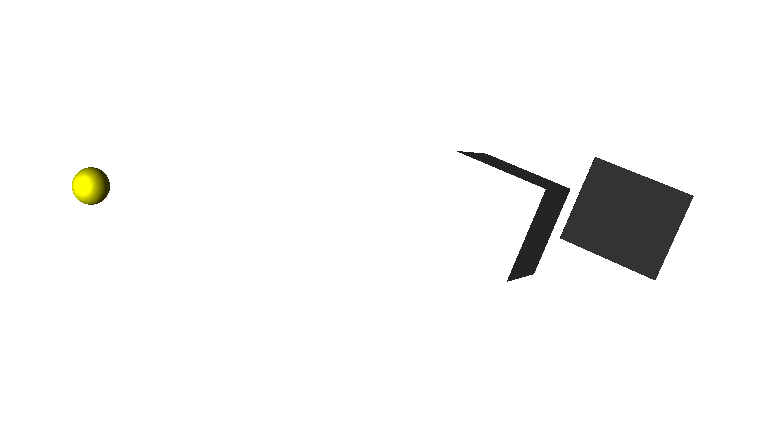
4, 4, GL\_MAP2\_VERTEX\_3);

gluEndSurface(theNurb);

glRotatef(90.0, 1.0, 0.0, 0.0);

}

Результат работы

Вывод

Я научился создавать динамические тени