ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| асп. |  |  |  | Д.А. Смолиенко |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №6 |
| ВЫВОД ТРЕХМЕРНЫХ ОБЪЕКТОВ С ДИНАМИЧЕСКИМ  РАСЧЕТОМ ПРОЕКЦИОННЫХ ТЕНЕЙ |
| по дисциплине:  КОМПЬЮТЕРНАЯ ГРАФИКА |
|  |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4631 |  |  |  | С.А. Гришин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2018

**Задание на лабораторную работу**

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

**Текст программы**

#include <GL/glut.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

static int tick = -1;

static int moving = 1;

static float materialColor[8][4] = {

{0.8, 0.8, 0.8, 1.0}, {0.8, 0.0, 0.0, 1.0}, {0.0, 0.8, 0.0, 1.0},

{0.0, 0.0, 0.8, 1.0}, {0.0, 0.8, 0.8, 1.0}, {0.8, 0.0, 0.8, 1.0},

{0.8, 0.8, 0.0, 1.0}, {0.0, 0.0, 0.0, 0.6},

};

static float lightPos[4] = {2.0, 4.0, 2.0, 1.0};

static float lightAmb[4] = {0.2, 0.2, 0.2, 1.0};

static float lightDiff[4] = {0.8, 0.8, 0.8, 1.0};

static float lightSpec[4] = {0.4, 0.4, 0.4, 1.0};

static float groundPlane[4] = {0.0, 1.0, 0.0, 1.499};

static float backPlane[4] = {0.0, 0.0, 1.0, 0.899};

static float fogColor[4] = {0.0, 0.0, 0.0, 0.0};

static float fogIndex[1] = {0.0};

static void drawCube(int color) {

int i;

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE,

&materialColor[color][0]);

for (i = 0; i < 6; ++i) {

glNormal3fv(&cube\_normals[i][0]);

glBegin(GL\_POLYGON);

glVertex4fv(&cube\_vertexes[i][0][0]);

glVertex4fv(&cube\_vertexes[i][1][0]);

glVertex4fv(&cube\_vertexes[i][2][0]);

glVertex4fv(&cube\_vertexes[i][3][0]);

glEnd();

}

}

static void myShadowMatrix(float ground[4], float light[4]) {

float dot;

float shadowMat[4][4];

dot = ground[0] \* light[0] + ground[1] \* light[1] + ground[2] \* light[2] +

ground[3] \* light[3];

shadowMat[0][0] = dot - light[0] \* ground[0];

shadowMat[1][0] = 0.0 - light[0] \* ground[1];

shadowMat[2][0] = 0.0 - light[0] \* ground[2];

shadowMat[3][0] = 0.0 - light[0] \* ground[3];

shadowMat[0][1] = 0.0 - light[1] \* ground[0];

shadowMat[1][1] = dot - light[1] \* ground[1];

shadowMat[2][1] = 0.0 - light[1] \* ground[2];

shadowMat[3][1] = 0.0 - light[1] \* ground[3];

shadowMat[0][2] = 0.0 - light[2] \* ground[0];

shadowMat[1][2] = 0.0 - light[2] \* ground[1];

shadowMat[2][2] = dot - light[2] \* ground[2];

shadowMat[3][2] = 0.0 - light[2] \* ground[3];

shadowMat[0][3] = 0.0 - light[3] \* ground[0];

shadowMat[1][3] = 0.0 - light[3] \* ground[1];

shadowMat[2][3] = 0.0 - light[3] \* ground[2];

shadowMat[3][3] = dot - light[3] \* ground[3];

glMultMatrixf((const GLfloat \*)shadowMat);

}

void idle(void) {

tick += 1;

glutPostRedisplay();

}

void display(void) {

GLfloat cubeXform[4][4];

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

glPushMatrix();

glTranslatef(-0.5, -0.5, -0.9);

glScalef(2.0, 2.0, 2.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, &materialColor[0][0]);

glBegin(GL\_POLYGON);

glVertex4fv(&koord\_flat[0][0]);

glVertex4fv(&koord\_flat[1][0]);

glVertex4fv(&koord\_flat[2][0]);

glVertex4fv(&koord\_flat[3][0]);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslatef(0.0, 1, 0.0);

glScalef(0.1, 0.1, 0.1);

glRotatef((360.0 / (30 \* 10)) \* tick \* 0.1, 1, 1, 1);

glScalef(1.0, 2.0, 1.0);

glGetFloatv(GL\_MODELVIEW\_MATRIX, (GLfloat \*)cubeXform);

drawCube(3);

glutSolidCube(2);

glPopMatrix();

glDepthMask(GL\_FALSE);

glEnable(GL\_BLEND);

glPushMatrix();

myShadowMatrix(backPlane, lightPos);

glTranslatef(0.0, 0.0, 2.0);

glMultMatrixf((const GLfloat \*)cubeXform);

drawCube(7);

glPopMatrix();

glDepthMask(GL\_TRUE);

glDisable(GL\_BLEND);

glutSwapBuffers();

}

void visible(int state) {

if (state == GLUT\_VISIBLE) {

if (moving)

glutIdleFunc(idle);

} else {

if (moving)

glutIdleFunc(NULL);

}

}

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

glutInitWindowSize(600, 600);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutCreateWindow("Lab 6");

glutDisplayFunc(display);

glutVisibilityFunc(visible);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluLookAt(-0.3, 0.5, 0.5, 1.0, 4.0, -3.0, 1.0, 1.0, -1.0);

glFrustum(-1.0, 1.0, -1.0, 1.0, 1.0, 3.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0, 0.0, -2.0);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPos);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, lightAmb);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightDiff);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, lightSpec);

glEnable(GL\_NORMALIZE);

glEnable(GL\_CULL\_FACE);

glCullFace(GL\_BACK);

glShadeModel(GL\_SMOOTH);

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

glPolygonStipple((const GLubyte \*)shadowPattern);

glClearColor(0.0, 0.0, 0.0, 1);

glClearIndex(0);

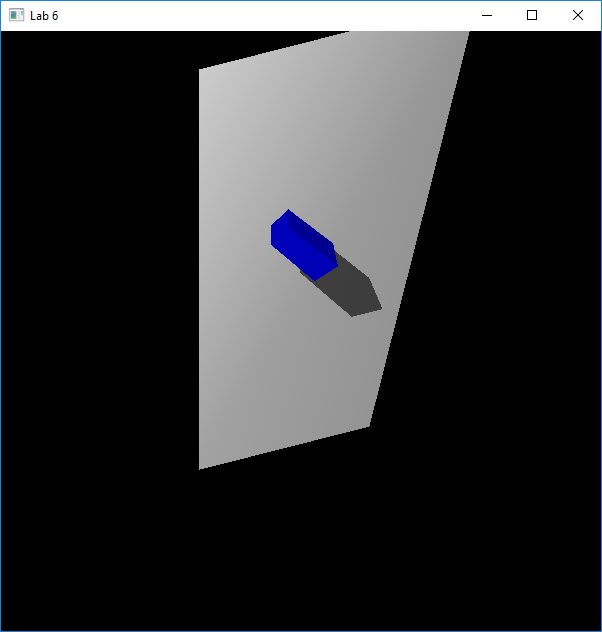
glClearDepth(1);

glutMainLoop();

return 0;

}

**Результат работы программы**



**Выводы**

В результате лабораторной работы разработано приложение выводящее трехмерную сцену с движущимся объектом, который отбрасывает тень на другой объект - плоскость. Тень должна перемещаться вместе с движением объекта исходя из взаимного положения источника света, объекта, который отбрасывает тень и объектов, на которые тень проецируется.