**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 6**

по курсу «Компьютерная графика»

Тема: Создание шейдерных анимационных эффектов.

Студент: Сахарин Н.А.

Группа: 80-308

Преподаватель: Чернышов Л.Н.

Оценка:

Москва, 2018

1. Постановка задачи

Для многогранника из предыдущих лабораторных работ обеспечить выполнение следующего шейдерного эффекта: Анимация, координата Y изменяется по закону Y = Y \* cos(t + Y).

2. Решение задачи

Данная лабораторная работа выполнена мной на языке программирования С++. Для обеспечения необходимого эффекта изображение должно перерисовываться с течением времени, при этом в нем должны происходить какие-то изменения (в моем случае меняется координата Y). Желательно также чтобы перерисовка происходила через достаточно малые интервалы времени. Для этого в программе есть функция, которая по таймеру вызывает перерисовку изображения, предварительно поменяв необходимые параметры.

3. Руководство по использованию программы

Программа запускается, сразу появляется окно с многогранником, который мгновенно начинает изменяться согласно заданному эффекту. Многогранник также можно осветить разными источниками света, повернуть. Окно масштабируется.

4. Листинг программы

#include <GL/glut.h>

#include <iostream>

#include <GL/glu.h> /\* this includes the necessary X headers \*/

#include <GL/gl.h>

#include <cmath>

using namespace std;

#define PI 3.14159265

double rx = 0;

double ry = 0;

double rz = 0;

double t = 0;

int light = 0;

void init()

{

glEnable(GL\_LIGHTING);

glEnable(GL\_COLOR\_MATERIAL);

float ambient[4] = {0.1, 0.1, 0.1, 1};

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambient);

glLightModelf(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE);

glEnable(GL\_NORMALIZE);

}

void dis() {

glClearColor(0.0, 0.0, 0.0, 0.0);

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

glLoadIdentity();

glMatrixMode(GL\_MODELVIEW);

float ambience[4] = {0.3f, 0.3f, 0.3f, 1.0};//установка "мирового" света, когда мы не задаем другое освещение

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

//GLfloat light0\_diffuse[] = {0.4, 0.7, 0.2};

float material\_diffuse[] = {1.0, 1.0, 1.0, 1.0};

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, material\_diffuse);

/\*

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light0\_diffuse);

glLightfv(GL\_LIGHT0, GL\_POSITION, light0\_direction);

\*/

if (light == 1) {

//направленный источник света

glEnable(GL\_LIGHT0);

GLfloat light0\_diffuse[] = {1.0, 1.0, 1.0};

GLfloat light0\_direction[] = {0.0, 0.0, 1.0, 1};

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light0\_diffuse);

glLightfv(GL\_LIGHT0, GL\_POSITION, light0\_direction);

}

if (light == 2) {

//точечный источник света с убыванием интенсивности с расстоянием

float light2\_diffuse[] = {0.4, 0.7, 0.2};

float light2\_position[] = {0.0, 0.0, 1.0, 1.0};

glEnable(GL\_LIGHT2);

glLightfv(GL\_LIGHT2, GL\_DIFFUSE, light2\_diffuse);

glLightfv(GL\_LIGHT2, GL\_POSITION, light2\_position);

glLightf(GL\_LIGHT2, GL\_CONSTANT\_ATTENUATION, 0.0);

glLightf(GL\_LIGHT2, GL\_LINEAR\_ATTENUATION, 0.2);

glLightf(GL\_LIGHT2, GL\_QUADRATIC\_ATTENUATION, 0.4);

}

if (light == 3) {

//прожектор

GLfloat light3\_diffuse[] = {0.4, 0.7, 0.2};

GLfloat light3\_position[] = {0.0, 0.0, 1.0, 1.0};

GLfloat light3\_spot\_direction[] = {0.0, 0.0, -1.0};

glEnable(GL\_LIGHT3);

glLightfv(GL\_LIGHT3, GL\_DIFFUSE, light3\_diffuse);

glLightfv(GL\_LIGHT3, GL\_POSITION, light3\_position);

glLightf(GL\_LIGHT3, GL\_SPOT\_CUTOFF, 30);

glLightfv(GL\_LIGHT3, GL\_SPOT\_DIRECTION, light3\_spot\_direction);

}

glRotatef(rx, 1.0, 0.0, 0.0);

glRotatef(ry, 0.0, 1.0, 0.0);

glRotatef(rz, 0.0, 0.0, 1.0);

glBegin(GL\_TRIANGLES);

glColor3f(1.0, 0.0, 0.0);

glVertex3f(0.0, 0.0, 1.0);

glVertex3f(1, 0.0, 0.0);

glVertex3f(1 \* cos(2\*PI / 5.), (1 \* sin(2\*PI / 5.))\*cos(t), 0.0);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.0, 0.0, 0.0);

glVertex3f(0.0, 0.0, 1.0);

glVertex3f(1 \* cos(2\*PI / 5.), 1 \* sin(2\*PI / 5.) \* cos(t), 0.0);

glVertex3f(1 \* cos(4\*PI / 5.), 1 \* sin(4\*PI / 5.) \* cos (t), 0.0);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.08, 0.0, 0.0);

glVertex3f(0.0, 0.0, 1.0);

glVertex3f(1 \* cos(4\*PI / 5.), 1 \* sin(4\*PI / 5.) \* cos(t), 0.0);

glVertex3f(1 \* cos(6\*PI / 5.), 1 \* sin(6\*PI / 5.) \* cos (t), 0.0);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.07, 0.0, 0.0);

glVertex3f(0.0, 0.0, 1.0);

glVertex3f(1 \* cos(6\*PI / 5.), 1 \* sin(6\*PI / 5.) \* cos(t), 0.0);

glVertex3f(1 \* cos(8\*PI / 5.), 1 \* sin(8\*PI / 5.) \* cos(t), 0.0);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.06, 0.0, 0.0);

glVertex3f(0.0, 0.0, 1.0);

glVertex3f(1 \* cos(8\*PI / 5.), 1 \* sin(8\*PI / 5.) \* cos(t), 0.0);

glVertex3f(1.0, 0.0, 0.0);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.05, 0.0, 0.0);

glVertex3f(0.0, 0.0, 0.0);

glVertex3f(1, 0.0, 0.0);

glVertex3f(1 \* cos(2\*PI / 5.), 1 \* sin(2\*PI / 5.) \* cos(t), 0.0);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.04, 0.0, 0.0);

glVertex3f(0.0, 0.0, 0.0);

glVertex3f(1 \* cos(2\*PI / 5.), 1 \* sin(2\*PI / 5.) \* cos(t), 0.0);

glVertex3f(1 \* cos(4\*PI / 5.), 1 \* sin(4\*PI / 5.) \* cos(t), 0.0);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.03, 0.0, 0.0);

glVertex3f(0.0, 0.0, 0.0);

glVertex3f(1 \* cos(4\*PI / 5.), 1 \* sin(4\*PI / 5.) \* cos(t), 0.0);

glVertex3f(1 \* cos(6\*PI / 5.), 1 \* sin(6\*PI / 5.) \* cos(t), 0.0);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.02, 0.0, 0.0);

glVertex3f(0.0, 0.0, 0.0);

glVertex3f(1 \* cos(6\*PI / 5.), 1 \* sin(6\*PI / 5.) \* cos(t), 0.0);

glVertex3f(1 \* cos(8\*PI / 5.), 1 \* sin(8\*PI / 5.) \* cos(t), 0.0);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.01, 0.0, 0.0);

glVertex3f(0.0, 0.0, 0.0);

glVertex3f(1 \* cos(8\*PI / 5.), 1 \* sin(8\*PI / 5.) \* cos(t), 0.0);

glVertex3f(1.0, 0.0, 0.0);

glEnd();

glDisable(GL\_LIGHT0);

glDisable(GL\_LIGHT2);

glDisable(GL\_LIGHT3);

glFlush();

glutSwapBuffers();

}

void reshape(int w, int h)

{

glViewport( 0, 0, w, h); // сброс текущей области вывода

glMatrixMode( GL\_PROJECTION );

glLoadIdentity();

glMatrixMode( GL\_MODELVIEW );

glLoadIdentity();

}

void key\_board(int key, int x, int y)

{

if (key == GLUT\_KEY\_RIGHT)

{

ry += 15;

}

else if (key == GLUT\_KEY\_LEFT)

{

ry -= 15;

}

else if (key == GLUT\_KEY\_DOWN)

{

rz -= 15;

}

else if (key == GLUT\_KEY\_UP)

{

rz += 15;

}

else if (key == GLUT\_KEY\_INSERT)

{

rx += 15;

}

else if (key == GLUT\_KEY\_F1) {

light = 1;

}

else if (key == GLUT\_KEY\_F2) {

light = 2;

}

else if (key == GLUT\_KEY\_F3) {

light = 3;

}

// Запрос обновления изображения

glutPostRedisplay();

}

void TimeFunction(int value) {

cout<<t<<endl;

glutPostRedisplay();

glutTimerFunc(60,TimeFunction,1);

t += 0.01 \* PI;

if (t == 2 \* PI) {

t = 0;

}

}

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

glutInit(&argcp, argv);

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

glutInitWindowSize(640, 480);

glutInitWindowPosition(0, 0);

glutCreateWindow("Sakharin 80-308");

init();

glEnable(GL\_DEPTH\_TEST);

glutReshapeFunc(reshape);

glutDisplayFunc(dis);

glutSpecialFunc(key\_board);

glutTimerFunc(300,TimeFunction,1);

glutMainLoop();

return 0;

}

Скриншоты:



