| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |
| --- | --- |

**ФАКУЛЬТЕТ** ***ИУК "Информатика и управление"***

**КАФЕДРА** ***ИУК4 "Программная инженерия"***

**ЛАБОРАТОРНАЯ РАБОТА №3**

**«РАБОТА С МАТРИЦАМИ И ПРЕОБРАЗОВАНИЯМИ В OPENGL»**

**ДИСЦИПЛИНА: «Компьютерная графика»**

| Выполнил: студент гр.ИУК5-41Б | | \_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_Шиндин А.О.\_\_\_\_)  (Подпись) (Ф.И.О.) |
| --- | --- | --- |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_Широкова Е.В.\_\_\_)  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2024

**Цели:** формирование практических навыков по работе с матричными преобразованиями для изменения сцены в целом и отдельных еѐ объектов, закрепление знаний о способах проецирования, изучения методов работы со стеком матриц средствами OpenGL, создание ряда многообъектных сцен и их преобразование с использованием переноса, поворота и масштабирования.

**Задачи:** сформировать понимание преобразований наблюдения, модели и проектирования, познакомиться с концепцией матриц преобразования, выяснить назначение единичной матрицы, научиться создавать приложения OpenGL с использованием матриц преобразования и ранее изученных объектов.

**Ход работы:**

Задание №1:Создать приложение, выводящее на экран вращающуюся пирамиду, каждая грань которой имеет свой собственный цвет.

Листинг программы:

#include "GL/glut.h"

#include "GL/gl.h"

double rotate\_y = 0;

double rotate\_x = 0;

void display() {

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

// Поворачиваем на угол

glRotatef(rotate\_x, 1, 0, 0);

glRotatef(rotate\_y, 0, 1, 0);

glBegin(GL\_POLYGON);

glColor3f(1.0, 0.5, 0.0);

glVertex3f(0.0, 0.5, 0.0);

glVertex3f(0.5, -0.5, -0.5);

glVertex3f(0.5, -0.5, 0.5);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(0.0, 0.5, 1.0);

glVertex3f(0.0, 0.5, 0.0);

glVertex3f(0.5, -0.5, -0.5);

glVertex3f(-0.5, -0.5, -0.5);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(0.0, 1.0, 0.0);

glVertex3f(0.0, 0.5, 0.0);

glVertex3f(-0.5, -0.5, -0.5);

glVertex3f(-0.5, -0.5, 0.5);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(1, 1, 0);

glVertex3f(0.0, 0.5, 0.0);

glVertex3f(-0.5, -0.5, 0.5);

glVertex3f(0.5, -0.5, 0.5);

glEnd();

glBegin(GL\_POLYGON);

glColor3f(1, 0, 0.5);

glVertex3f(0.5, -0.5, 0.5);

glVertex3f(0.5, -0.5, -0.5);

glVertex3f(-0.5, -0.5, -0.5);

glVertex3f(-0.5, -0.5, 0.5);

glEnd();

glutSwapBuffers();

}

void TimerFunction(int value) {

rotate\_x = 1;

rotate\_y = 1;

glutPostRedisplay(); // Пометка окна на перерисовку

glutTimerFunc(50, TimerFunction, 0); // 10 милисикунд

}

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

glutInit(&argc, argv); // Инициализация GLUT

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

glutInitWindowPosition(500, 100);

glutInitWindowSize(800, 800);

glutCreateWindow("Cube");

glEnable(GL\_DEPTH\_TEST); // Включение буфера глубины для правильного отображения глубины объектов

glutDisplayFunc(display);

glutTimerFunc(50, TimerFunction, 0); // Установка функции таймера

glutMainLoop();

return 0;

}

Результат:

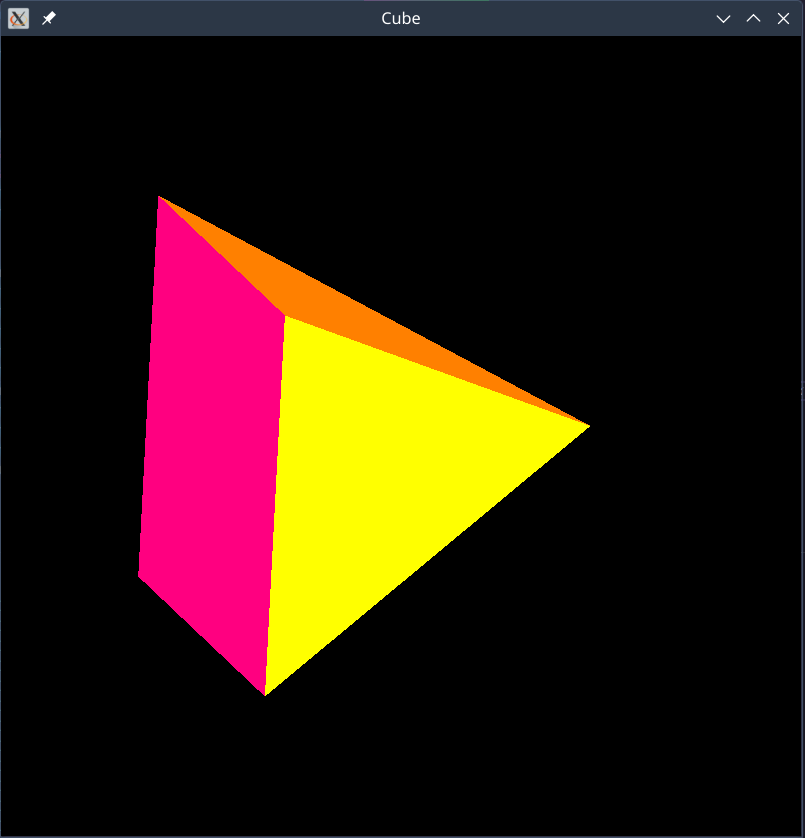


Рисунок 1 - результат выполнения первого задания

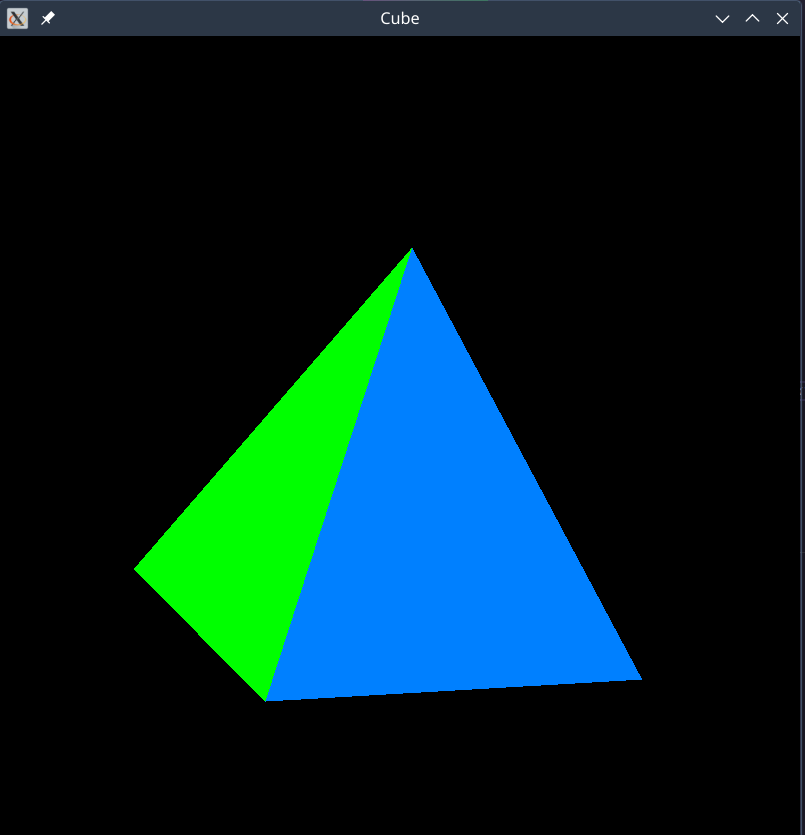


Рисунок 2 - результат выполнения первого задания

Задание №2: Для Примера 1 реализовать изменения согласно варианту, добавить один электрон на максимально большое расстояние от ядра:

Пример 1:

#include "glew.h"

#include "glut.h"

#include <math.h>

// Вызывается для рисования сцены

void RenderScene(void)

{

// Угол поворота вокруг ядра

static GLfloat fElect1 = 0.0f;

// Очищаем окно текущим цветом очистки

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

//Обновляем матрицу наблюдения модели

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

//Транслируем всю сцену в поле зрения

//Это исходное преобразование наблюдения

glTranslatef(0.0f, 0.0f, -100.0f);

// Красное ядро

glColor3ub(255, 255, 255);

glutSolidSphere(10.0f, 150, 150);

// Желтые электроны

glColor3ub(255, 255, 0);

// Орбита первого электрона

// Записываем преобразование наблюдения

glPushMatrix();

glRotatef(fElect1, 0.0f, 1.0f, 0.0f);

glTranslatef(90.0f, 0.0f, 0.0f);

// Рисуем электрон

glutSolidSphere(6.0f, 150, 150);

// Восстанавливаем преобразование наблюдения

glPopMatrix();

//Орбита второго электрона

glPushMatrix();

glRotatef(45.0f, 0.0f, 0.0f, 1.0f);

glRotatef(fElect1, 0.0f, 1.0f, 0.0f);

glTranslatef(-70.0f, 0.0f, 0.0f);

glutSolidSphere(6.0f, 15, 15);

glPopMatrix();

// Орбита третьего электрона

glPushMatrix();

glRotatef(360.0f - 45.0f, 0.0f, 0.0f, 1.0f);

glRotatef(fElect1, 0.0f, 1.0f, 0.0f);

glTranslatef(0.0f, 0.0f, 60.0f);

glutSolidSphere(6.0f, 15, 15);

glPopMatrix();

// Увеличиваем угол поворота

fElect1 += 10.0f;

if (fElect1 > 360.0f)

fElect1 = 0.0f;

// Показываем построенное изображение

glutSwapBuffers();

}

// Функция выполняет необходимую инициализацию

// в контексте визуализации

void SetupRC()

{

glEnable(GL\_DEPTH\_TEST); // Удаление скрытых поверхностей

glFrontFace(GL\_CCW); // Полигоны с обходом против

//часовой стрелки направлены наружу

glEnable(GL\_CULL\_FACE); //Внутри пирамиды расчеты не //производятся

// Черный фон

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

}

void TimerFunc(int value)

{

glutPostRedisplay();

glutTimerFunc(100, TimerFunc, 1);

}

void ChangeSize(int w, int h)

{

GLfloat nRange = 100.0f;

// Предотвращение деления на ноль

if (h == 0)

h = 1;

// Устанавливает поле просмотра по размерам окна

glViewport(0, 0, w, h);

// Обновляет стек матрицы проектирования

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Устанавливает объем отсечения с помощью отсекающих

// плоскостей (левая, правая, нижняя, верхняя,

// ближняя, дальняя)

if (w <= h)

glOrtho(-nRange, nRange, nRange \* h / w, -nRange \* h / w, -nRange \* 2.0f, nRange \*

2.0f);

else

glOrtho(-nRange \* w / h, nRange \* w / h, nRange, -nRange, -nRange \* 2.0f, nRange \*

2.0f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

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

glutInitWindowSize(800, 600);

glutCreateWindow("OpenGL Atom");

glutReshapeFunc(ChangeSize);

glutDisplayFunc(RenderScene);

glutTimerFunc(500, TimerFunc, 1);

SetupRC();

glutMainLoop();

return 0;

}

Листинг программы:

#include "GL/glew.h"

#include "GL/glut.h"

#include "GL/gl.h"

#include <math.h>

// Вызывается для рисования сцены

void RenderScene(void)

{

// Угол поворота вокруг ядра

static GLfloat fElect1 = 0.0f;

// Очищаем окно текущим цветом очистки

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

// Обновляем матрицу наблюдения модели

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

// Транслируем всю сцену в поле зрения

// Это исходное преобразование наблюдения

glTranslatef(0.0f, 0.0f, -100.0f);

// Красное ядро

glColor3ub(255, 255, 255);

glutSolidSphere(10.0f, 150, 150);

// Желтый электрон

glColor3ub(255, 255, 0);

// Орбита первого электрона

// Записываем преобразование наблюдения

glPushMatrix();

glRotatef(fElect1, 0.0f, 1.0f, 0.0f);

glTranslatef(90.0f, 0.0f, 0.0f);

// Рисуем электрон

glutSolidSphere(6.0f, 150, 150);

// Восстанавливаем преобразование наблюдения

glPopMatrix();

// ,,, электрон

glColor3ub(255, 123, 123);

// Орбита второго электрона

glPushMatrix();

glRotatef(45.0f, 0.0f, 0.0f, 1.0f);

glRotatef(fElect1, 0.0f, 1.0f, 0.0f);

glTranslatef(-70.0f, 0.0f, 0.0f);

glutSolidSphere(6.0f, 15, 15);

glPopMatrix();

// ,,, электрон

glColor3ub(123, 123, 255);

// Орбита третьего электрона

glPushMatrix();

glRotatef(360.0f - 45.0f, 0.0f, 0.0f, 1.0f);

glRotatef(fElect1, 0.0f, 1.0f, 0.0f);

glTranslatef(0.0f, 0.0f, 60.0f);

glutSolidSphere(6.0f, 15, 15);

glPopMatrix();

// Орбита третьего электрона

glColor3ub(255, 0, 255);

glPushMatrix();

glRotatef(270.0f, 0.0f, 0.0f, 1.0f);

glRotatef(fElect1, 0.0f, 1.0f, 0.0f);

glTranslatef(0.0f, -10.0f, 95.f);

glutSolidSphere(6.0f, 15, 15);

glPopMatrix();

// Увеличиваем угол поворота

fElect1 += 10.0f;

if (fElect1 > 360.0f)

fElect1 = 0.0f;

// Показываем построенное изображение

glutSwapBuffers();

}

// Функция выполняет необходимую инициализацию

// в контексте визуализации

void SetupRC()

{

glEnable(GL\_DEPTH\_TEST); // Удаление скрытых поверхностей

glFrontFace(GL\_CCW); // Полигоны с обходом против

// часовой стрелки направлены наружу

glEnable(GL\_CULL\_FACE); // Внутри пирамиды расчеты не // производятся

// Черный фон

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

}

void TimerFunc(int value)

{

glutPostRedisplay();

glutTimerFunc(50, TimerFunc, 1);

}

void ChangeSize(int w, int h)

{

GLfloat nRange = 100.0f;

// Предотвращение деления на ноль

if (h == 0)

h = 1;

// Устанавливает поле просмотра по размерам окна

glViewport(0, 0, w, h);

// Обновляет стек матрицы проектирования

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Устанавливает объем отсечения с помощью отсекающих

// плоскостей (левая, правая, нижняя, верхняя,

// ближняя, дальняя)

if (w <= h)

glOrtho(-nRange, nRange, nRange \* h / w, -nRange \* h / w, -nRange \* 2.0f, nRange \* 2.0f);

else

glOrtho(-nRange \* w / h, nRange \* w / h, nRange, -nRange, -nRange \* 2.0f, nRange \* 2.0f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

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

glutInitWindowPosition(500, 100);

glutInitWindowSize(800, 800);

glutCreateWindow("OpenGL Atom");

glutReshapeFunc(ChangeSize);

glutDisplayFunc(RenderScene);

glutTimerFunc(100, TimerFunc, 1);

SetupRC();

glutMainLoop();

return 0;

}

Результат:

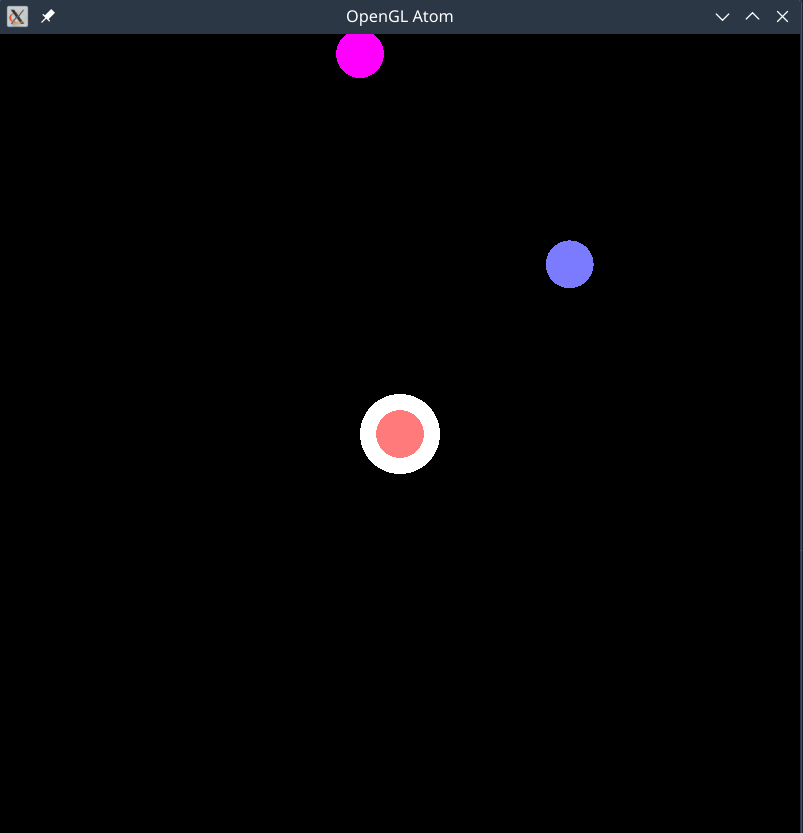


Рисунок 4 - к примеру добавлен фиолетовый электрон

Задание №3: Для Примера 2 реализовать изменения согласно варианту: Определим угол альфа как угол между горизонтальной плоскостью и новой плоскостью вращения (будущей плоскостью вращения) объекта в том виде, в котором мы видим его на экране, изменить плоскость вращения Земли и Луны, скорость и направление вращения Земли.

Пример 1:

#include "GL/glew.h"

#include "GL/glut.h"

#include <math.h>

// Параметры освещения

GLfloat whiteLight[] = { 0.2f, 0.2f, 0.2f, 1.0f };

GLfloat sourceLight[] = { 0.8f, 0.8f, 0.8f, 1.0f };

GLfloat lightPos[] = { 0.0f, 0.0f, 0.0f, 1.0f };

// Вызывается для рисования сцены

void RenderScene(void)

{

// Угол поворота системы Земля/Луна

static float fMoonRot = 0.0f;

static float fEarthRot = 0.0f;

// Очищаем окно текущим цветом очистки

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

// Save the matrix state and do the rotations

glMatrixMode(GL\_MODELVIEW);

glPushMatrix();

// Транслируем всю сцену в поле зрения

glTranslatef(0.0f, 0.0f, -300.0f);

// Устанавливаем цвет материала красным

// Солнце

glDisable(GL\_LIGHTING);

glColor3ub(255, 255, 0);

glutSolidSphere(15.0f, 30, 17);

glEnable(GL\_LIGHTING);

// Движение источника света, после прорисовки солнца!

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

// Поворот системы координат

glRotatef(fEarthRot, 0.0f, 1.0f, 0.0f);

// Прорисовка Земли

glColor3ub(0, 0, 255);

glTranslatef(105.0f, 0.0f, 0.0f);

glutSolidSphere(15.0f, 30, 17);

// Поворот в системе координат, связанной с Землей

// и изображение Луны

glColor3ub(200, 200, 200);

glRotatef(fMoonRot, 0.0f, 1.0f, 0.0f);

glTranslatef(30.0f, 0.0f, 0.0f);

fMoonRot += 15.0f;

if (fMoonRot > 360.0f)

fMoonRot = 0.0f;

glutSolidSphere(6.0f, 30, 17);

// Восстанавливается состояние матрицы

glPopMatrix(); // Матрица наблюдения модели

// Шаг по орбите Земли равен пяти градусам

fEarthRot += 5.0f;

if (fEarthRot > 360.0f)

fEarthRot = 0.0f;

// Показывается построенное изображение

glutSwapBuffers();

}

// Функция выполняет всю необходимую инициализацию в контексте

// визуализации

void SetupRC()

{

// Параметры света и координаты

glEnable(GL\_DEPTH\_TEST); // Удаление скрытых поверхностей

glFrontFace(GL\_CCW); // Многоугольники с обходом против часовой стрелки направлены наружу

glEnable(GL\_CULL\_FACE); // Расчеты внутри самолета не выполняются

// Активация освещения

glEnable(GL\_LIGHTING);

// Устанавливается и активизируется источник света 0

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

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

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

glEnable(GL\_LIGHT0);

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

// Свойства материалов соответствуют кодам glColor

glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);

// Темно-синий фон

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

}

void TimerFunc(int value)

{

glutPostRedisplay();

glutTimerFunc(100, TimerFunc, 1);

}

void ChangeSize(int w, int h)

{

GLfloat fAspect;

// Предотвращает деление на ноль

if (h == 0)

h = 1;

// Размер поля просмотра устанавливается равным размеру окна

glViewport(0, 0, w, h);

// Расчет соотношения сторон окна

fAspect = (GLfloat)w / (GLfloat)h;

// Устанавливаем перспективную систему координат

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Поле обзора равно 45 градусов, ближняя и дальняя плоскости

// проходят через 1 и 425

gluPerspective(45.0f, fAspect, 1.0, 425.0);

// Обновляем матрицу наблюдения модели

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

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

glutInitWindowSize(800, 600);

glutCreateWindow("Earth/Moon/Sun System");

glutReshapeFunc(ChangeSize);

glutDisplayFunc(RenderScene);

glutTimerFunc(250, TimerFunc, 1);

SetupRC();

glutMainLoop();

return 0;

}

Листинг программы:

#include "GL/glew.h"

#include "GL/glut.h"

#include <math.h>

// Параметры освещения

GLfloat whiteLight[] = { 0.2f, 0.2f, 0.2f, 1.0f };

GLfloat sourceLight[] = { 0.8f, 0.8f, 0.8f, 1.0f };

GLfloat lightPos[] = { 0.0f, 0.0f, 0.0f, 1.0f };

// Вызывается для рисования сцены

void RenderScene(void)

{

// Угол поворота системы Земля/Луна

static float fMoonRot = 0.0f;

static float fEarthRot = 0.0f;

// Очищаем окно текущим цветом очистки

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

// Save the matrix state and do the rotations

// glMatrixMode(GL\_MODELVIEW);

glPushMatrix();

// Транслируем всю сцену в поле зрения

glTranslatef(0.0f, 10.0f, -300.0f);

// \* изменить плоскость вращения Земли и Луны, скорость и направление вращения Земли.

// Устанавливаем цвет материала красным

// Солнце

glDisable(GL\_LIGHTING);

glColor3ub(255, 255, 0);

glutSolidSphere(15.0f, 30, 17);

glEnable(GL\_LIGHTING);

// Движение источника света, после прорисовки солнца!

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

// Поворот системы координат

glRotatef(fEarthRot, 0.0f, 0.0f, 1.0f);

// \* изменить плоскость вращения Земли и Луны, скорость и направление вращения Земли.

// Прорисовка Земли

glColor3ub(0, 0, 255);

glTranslatef(80.0f, 25.0f, 0.0f);

glutSolidSphere(15.0f, 30, 17);

// Поворот в системе координат, связанной с Землей

// и изображение Луны

glColor3ub(200, 200, 200);

glRotatef(fMoonRot, 0.0f, 1.0f, 0.0f);

glTranslatef(30.0f, 0.0f, 0.0f);

fMoonRot += 15.0f;

if (fMoonRot > 360.0f)

fMoonRot = 0.0f;

glutSolidSphere(6.0f, 30, 17);

// Восстанавливается состояние матрицы

// Шаг по орбите Земли равен пяти градусам

fEarthRot += 5.0f;

if (fEarthRot > 360.0f)

fEarthRot = 0.0f;

glPopMatrix(); // Матрица наблюдения модели

// Показывается построенное изображение

glutSwapBuffers();

}

void SetupRC()

{

// Параметры света и координаты

glEnable(GL\_DEPTH\_TEST); // Удаление скрытых поверхностей

glFrontFace(GL\_CCW); // Многоугольники с обходом против часовой стрелки направлены наружу

glEnable(GL\_CULL\_FACE); // Расчеты внутри самолета не выполняются

// Активация освещения

glEnable(GL\_LIGHTING);

// Устанавливается и активизируется источник света 0

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

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

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

glEnable(GL\_LIGHT0);

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

// Свойства материалов соответствуют кодам glColor

glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

}

void TimerFunc(int value)

{

glutPostRedisplay();

glutTimerFunc(50, TimerFunc, 1);

}

void ChangeSize(int w, int h)

{

GLfloat fAspect;

// Предотвращает деление на ноль

if (h == 0)

h = 1;

// Размер поля просмотра устанавливается равным размеру окна

glViewport(0, 0, w, h);

// Расчет соотношения сторон окна

fAspect = (GLfloat)w / (GLfloat)h;

// Устанавливаем перспективную систему координат

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Поле обзора равно 45 градусов, ближняя и дальняя плоскости

// проходят через 1 и 425

gluPerspective(45.0f, fAspect, 1.0, 425.0);

// Обновляем матрицу наблюдения модели

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

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

glutInitWindowPosition(500, 100);

glutInitWindowSize(800, 800);

glutCreateWindow("Earth/Moon/Sun System");

glutReshapeFunc(ChangeSize);

glutDisplayFunc(RenderScene);

glutTimerFunc(100, TimerFunc, 1);

SetupRC();

glutMainLoop();

return 0;

}

Результат:

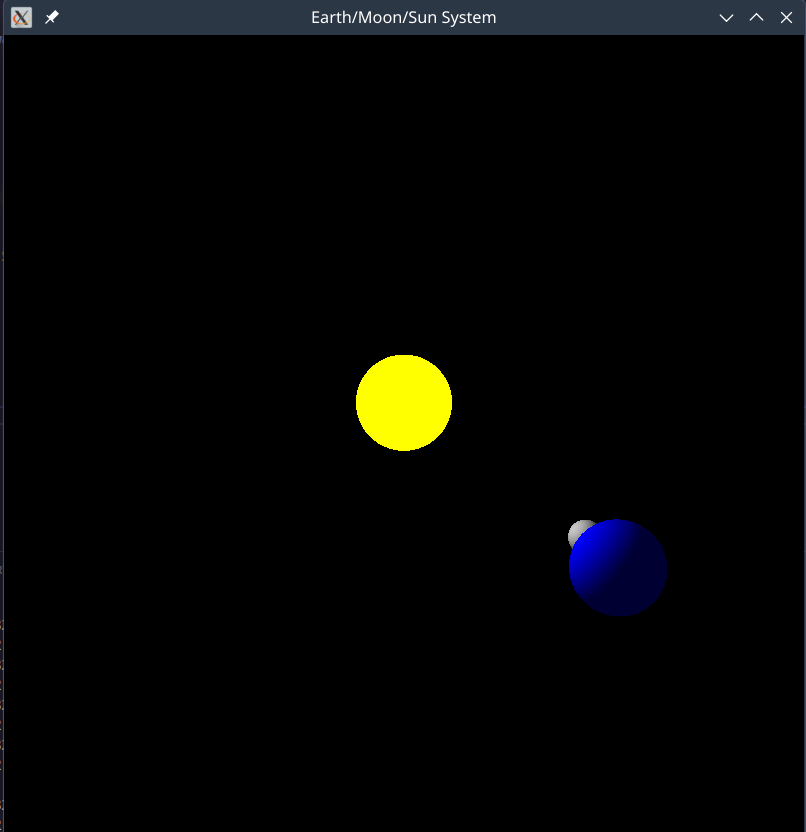


Рисунок 5 - изменены оси вращения луны и земли.

**Вывод:** в ходе выполнения лабораторной работы были сформированы практические навыки по работе с матричными преобразованиями в OpenGL для изменения сцены в целом и отдельных объектов, также были изучили способы проецирования, освоили методы работы со стеком матриц средствами OpenGL.

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Терминология преобразований OpenGL включает в себя следующие понятия:

- Преобразование модели (Model Transformation): изменение положения, ориентации и размера объектов в пространстве модели.

- Преобразование вида (View Transformation): определение точки обзора и направления обзора для определения того, как сцена отображается на экране.

- Преобразование проекции (Projection Transformation): преобразование трехмерных координат в двумерные координаты экрана с учетом перспективы.

- Преобразование вьюпорта (Viewport Transformation): определение области экрана, на которой будет отображаться изображение.

2. Координаты наблюдения - это координаты точки в пространстве, откуда происходит наблюдение за сценой. Они связаны с трехмерной декартовой системой координат тем, что определяют положение камеры или точки обзора в этом пространстве.

3. Дуализм преобразования проекции модели заключается в том, что они выполняются последовательно и в обратном порядке. Преобразование модели изменяет положение объектов в пространстве модели, а преобразование проекции проецирует эти объекты на экран. Оба преобразования влияют на отображение сцены, но в разных аспектах.

4. Конвейер преобразований OpenGL - это последовательность этапов, через которые проходят вершины сцены перед отображением на экране. Этапы конвейера включают:

- Преобразование модели

- Преобразование вида

- Преобразование проекции

- Отсечение

- Растеризация

- Фрагментация

- Окончательное смешивание

5. Способы непропорционального масштабирования включают использование различных коэффициентов масштабирования для различных осей или применение асимметричных преобразований масштабирования.

6. Преобразование поворота в OpenGL описывается с помощью матрицы поворота, которая применяется к вершинам объекта. Матрица поворота содержит угол поворота и ось вращения.

7. Трансляция на уровне матриц представляет собой изменение координат объекта путем добавления смещения к каждой из его вершин. Это осуществляется путем добавления в матрицу преобразования модели строки, представляющей собой вектор смещения.

8. Единичная матрица - это квадратная матрица, у которой на главной диагонали стоят единицы, а все остальные элементы равны нулю. Единичная матрица не изменяет координаты точек при преобразованиях.

9. Стек матриц в OpenGL используется для сохранения и восстановления матриц преобразований. Он позволяет сохранить текущее состояние матрицы и временно переключиться на другое состояние, например, для рисования вложенных объектов или применения различных преобразований.

10. Нетривиальное умножение матриц - это процесс, при котором матрицы умножаются друг на друга в определенном порядке, что позволяет комбинировать различные преобраз

ования и эффекты. Преимущества использования нетривиального умножения матриц включают более гибкое и эффективное управление преобразованиями.

11. Функция таймера в OpenGL используется для управления временем и частотой обновления сцены. Она позволяет создавать анимацию, задавая частоту обновления экрана и изменяя параметры сцены в соответствии с прошедшим временем.

12. В контексте построения сцен, актеры - это объекты или элементы сцены, которые действуют и взаимодействуют друг с другом. Например, в игровой сцене актерами могут быть персонажи, объекты окружения или другие элементы, которые являются частью игрового процесса.

**ЛИТЕРАТУРА:**

1. Боресков А.В. Основы работы с технологией CUDA / А.В. Боресков, А.А. Харламов - Издательство "ДМК Пресс", 2010. - 232 с. -

ISBN 978-5-94074-578-5; ЭБС «Лань». - URL:

https://e.lanbook.com/book/1260#book\_name (23.12.2017).

2. Васильев С.А. OpenGL. Компьютерная графика : учебное пособие /

С.А. Васильев. — Электрон. текстовые данные. — Тамбов: Тамбовский государственный технический университет, ЭБС АСВ,

2012. — 81 c. — 2227-8397. — Режим доступа:

http://www.iprbookshop.ru/63931.html — ЭБС «IPRbooks», по паролю

3. Вольф Д. OpenGL 4. Язык шейдеров. Книга рецептов/ Вольф Д. -

Издательство "ДМК Пресс", 2015. - 368 с. - 978-5-97060-255-3;

ЭБС «Лань». - URL: https://e.lanbook.com/book/ 73071#book\_name

(23.12.2017).

4. Гинсбург Д. OpenGL ES 3.0. Руководство разработчика/Д. Гинсбург, Б. Пурномо. - Издательство "ДМК Пресс", 2015. - 448 с. -

ISBN

978-5-97060-256-0; ЭБС «Лань». - URL: https://e.lanbook.com/book/

82816#book\_name (29.12.2017).

5. Лихачев В.Н. Создание графическиx моделей с помощью Open

Graphics Library / В.Н. Лихачев. — Электрон. текстовые данные. —

М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 201 c. — 2227-8397. — Режим доступа:

<http://www.iprbookshop.ru/39567.html>

6. Забелин Л.Ю. Основы компьютерной графики и технологии трехмерного моделирования : учебное пособие/ Забелин Л.Ю., Конюкова О.Л., Диль О.В.— Новосибирск: Сибирский государственный

университет телекоммуникаций и информатики, 2015.— 259 c.—

Режим доступа: http://www.iprbookshop.ru/54792.— ЭБС

«IPRbooks», по паролю

7. Папуловская Н.В. Математические основы программирования

трехмерной графики : учебно-методическое пособие / Н.В. Папу-

ловская. — Электрон. текстовые данные. — Екатеринбург: Уральский федеральный университет, 2016. — 112 c. — 978-5-7996-1942-

8. — Режим доступа: <http://www.iprbookshop.ru/68345.html>

8. Перемитина, Т.О. Компьютерная графика : учебное пособие / Т.О.

Перемитина ; Министерство образования и науки Российской Федерации, Томский Государственный Университет Систем Управления и Радиоэлектроники (ТУСУР). - Томск : Эль Контент, 2012. -

144 с. : ил.,табл., схем. - ISBN 978-5-4332-0077-7 ; - URL:

http//biblioclub.ru/index.php?page=book&id=208688 (30.11.2017).