|  |  |
| --- | --- |
| **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 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №3**

**«Работа с матрицами и преобразованиями в OpenGL»**

**ДИСЦИПЛИНА: «Компьютерная графика»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Глебов С.А. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

**Цель:** формирование практических навыков по работе с матричными преобразованиями для изменения сцены в целом и отдельных её 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.

**Вариант 7**

**Задание:**

1. **Листинг 1**: на каждую существующую орбиту добавить по одному атому другого цвета.
2. С помощью **Листинга 2** и **Листинга 3** изобразить фигуру с разным цветом внутренних и внешних граней (возможен разный цвет всех внешних граней). Поменять цвет фона.
3. Для **Листинга 4** добавить к Земле еще одни спутник меньшего размера, находящийся на другой траектории и с вращением в противоположную сторону.
4. Для **Листинга 5** реализовать хаотичное изменение скорости вращения тора.
5. Для **Листинга 7** добавить к вращающемуся тору еще две вращающиеся сферы в разных плоскостях.

**Листинг 1:**

#include "glew.h"

#include "glut.h"

#include <math.h>

namespace Listing\_3\_1

{

// Величина поворота

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

// 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);

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

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

// Красное ядро

glColor3ub(255, 0, 0);

glutSolidSphere(10.0f, 15, 15);

// Желтые электроны

glColor3ub(255, 255, 0);

// Орбита первого электрона

// Записываем преобразование наблюдения

glPushMatrix();

// Поворачиваем на угол поворота

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

// Трансляция элемента от начала координат на орбиту

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

// Рисуем электрон

glutSolidSphere(6.0f, 15, 15);

glTranslatef(-180.0f, 0.0f, 0.0f);

glColor3ub(255, 0, 255);

glutSolidSphere(6.0f, 15, 15);

glColor3ub(255, 255, 0);

// 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);

glTranslatef(140.0f, 0.0f, 0.0f);

glColor3ub(255, 0, 255);

glutSolidSphere(6.0f, 15, 15);

glColor3ub(255, 255, 0);

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);

glTranslatef(0.0f, 0.0f, -120.0f);

glColor3ub(255, 0, 255);

glutSolidSphere(6.0f, 15, 15);

glPopMatrix();

// Увеличиваем угол поворота

fElect1 += 5.0f;

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(1.0f, 1.0f, 1.0f, 1.0f);

}

void SpecialKeys(int key, int x, int y)

{

if (key == GLUT\_KEY\_UP)

xRot -= 5.0f;

if (key == GLUT\_KEY\_DOWN)

xRot += 5.0f;

if (key == GLUT\_KEY\_LEFT)

yRot -= 5.0f;

if (key == GLUT\_KEY\_RIGHT)

yRot += 5.0f;

if (xRot > 356.0f)

xRot = 0.0f;

if (xRot < -1.0f)

xRot = 355.0f;

if (yRot > 356.0f)

yRot = 0.0f;

if (yRot < -1.0f)

yRot = 355.0f;

// Перерисовывает сцену с новыми координатами

//glutPostRedisplay();

}

void TimerFunc(int value)

{

glutPostRedisplay();

glutTimerFunc(10, 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);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

glutTimerFunc(500, TimerFunc, 1);

SetupRC();

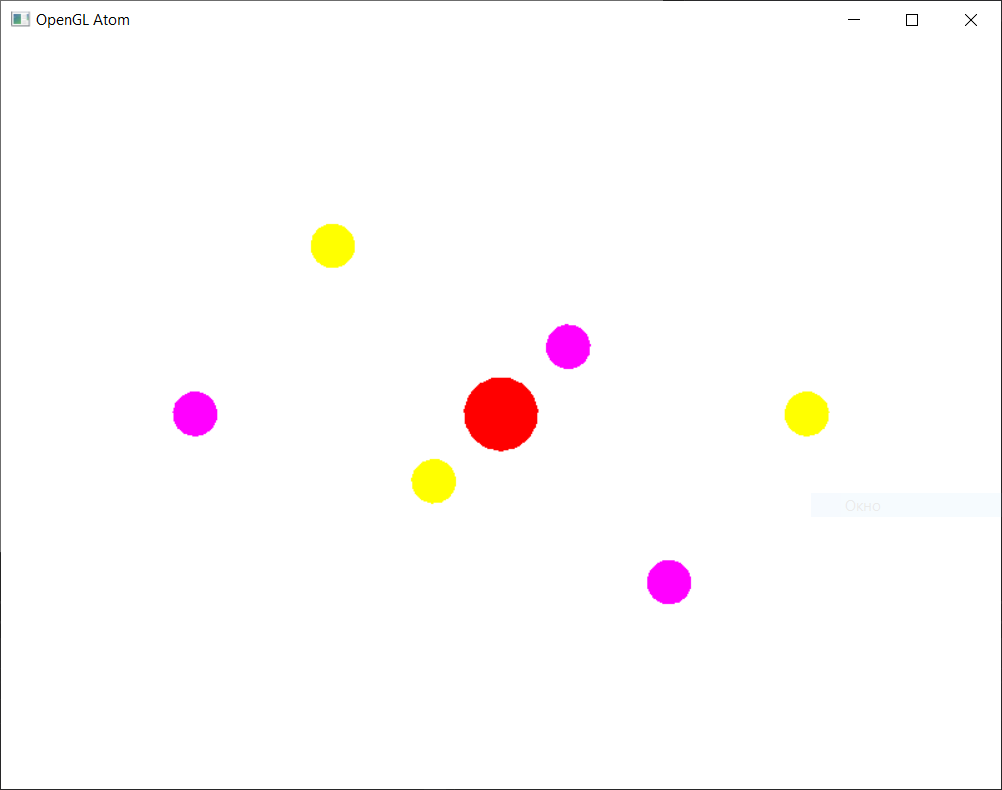
glutMainLoop();

return 0;

}

}

**Результат листинга 1:**



**Рисунок 1.** Результат листинга 1

**Листинг 2:**

#include "glew.h"

#include "glut.h"

namespace Listing\_3\_2

{

// Величина поворота

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

// Вызывается при изменении размеров окна

void ChangeSize(GLsizei w, GLsizei h)

{

GLfloat nRange = 120.0f;

// Предотвращает деление на нуль

if (h == 0)

h = 1;

// Устанавливает размеры поля просмотра равны размерам окна

glViewport(0, 0, w, h);

// Устанавливаем перспективную систему координат

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();

}

// Эта функция выполняет необходимую инициализацию в контексте

// визуализации. В данном случае устанавливается

// и инициализируется освещение сцены.

void SetupRC()

{

// Параметры света

GLfloat whiteLight[] = { 0.45f, 0.45f, 0.45f, 1.0f };

GLfloat sourceLight[] = { 0.25f, 0.25f, 0.25f, 1.0f };

GLfloat lightPos[] = { -50.f, 25.0f, 250.0f, 0.0f };

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

// Активизация освещения

glEnable(GL\_LIGHTING);

// Устанавливается и активизируется источник света 0

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

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

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(1.0f, 1.0f, 1.0f, 1.0f);

}

// Реагирует на клавиши со стрелками, двигая систему отсчета камеры

void SpecialKeys(int key, int x, int y)

{

if (key == GLUT\_KEY\_UP)

xRot -= 5.0f;

if (key == GLUT\_KEY\_DOWN)

xRot += 5.0f;

if (key == GLUT\_KEY\_LEFT)

yRot -= 5.0f;

if (key == GLUT\_KEY\_RIGHT)

yRot += 5.0f;

xRot = (GLfloat)((const int)xRot % 360);

yRot = (GLfloat)((const int)yRot % 360);

// Обновляем окно

glutPostRedisplay();

}

// Вызывается для рисования сцены

void RenderScene(void)

{

float fZ, bZ;

// Очищаем окно текущим цветом очистки

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

fZ = 100.0f;

bZ = -100.0f;

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

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

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

// В качестве текущего цвета рисования задает красный

glColor3f(1.0f, 0.0f, 1.0f);

// Лицевая часть

///////////////////////////////////

glBegin(GL\_QUADS);

// прямо вверх оси z

glNormal3f(0.0f, 0.0f, 1.0f);

// Левая сторона

glVertex3f(-50.0f, 50.0f, fZ);

glVertex3f(-50.0f, -50.0f, fZ);

glVertex3f(-35.0f, -50.0f, fZ);

glVertex3f(-35.0f, 50.0f, fZ);

// Правая сторона

glVertex3f(50.0f, 50.0f, fZ);

glVertex3f(35.0f, 50.0f, fZ);

glVertex3f(35.0f, -50.0f, fZ);

glVertex3f(50.0f, -50.0f, fZ);

// Верх

glVertex3f(-35.0f, 50.0f, fZ);

glVertex3f(-35.0f, 35.0f, fZ);

glVertex3f(0.0f, 15.0f, fZ);

glVertex3f(0.0f, 30.0f, fZ);

glVertex3f(35.0f, 35.0f, fZ);

glVertex3f(35.0f, 50.0f, fZ);

glVertex3f(0.0f, 30.0f, fZ);

glVertex3f(0.0f, 15.0f, fZ);

// Низ

glVertex3f(-35.0f, -35.0f, fZ);

glVertex3f(-35.0f, -50.0f, fZ);

glVertex3f(35.0f, -50.0f, fZ);

glVertex3f(35.0f, -35.0f, fZ);

// Верхняя длинная часть

////////////////////////////

//Все нормали для верхней части вверх оси Y

glNormal3f(0.0f, 1.0f, 0.0f);

glVertex3f(-50.0f, 50.0f, fZ);

glVertex3f(-35.0f, 50.0f, fZ);

glVertex3f(-35.0f, 50.0f, bZ);

glVertex3f(-50.0f, 50.0f, bZ);

glNormal3f(0.0f, 1.0f, 0.0f);

glVertex3f(35.0f, 50.0f, fZ);

glVertex3f(50.0f, 50.0f, fZ);

glVertex3f(50.0f, 50.0f, bZ);

glVertex3f(35.0f, 50.0f, bZ);

glNormal3f(1.0f, 1.0f, 0.0f);

glVertex3f(-35.0f, 50.0f, fZ);

glVertex3f(0.0f, 30.0f, fZ);

glVertex3f(0.0f, 30.0f, bZ);

glVertex3f(-35.0f, 50.0f, bZ);

glNormal3f(-1.0f, 1.0f, 0.0f);

glVertex3f(0.0f, 30.0f, fZ);

glVertex3f(35.0f, 50.0f, fZ);

glVertex3f(35.0f, 50.0f, bZ);

glVertex3f(0.0f, 30.0f, bZ);

// Нижняя секция

glNormal3f(0.0f, -1.0f, 0.0f);

glVertex3f(-50.0f, -50.0f, fZ);

glVertex3f(-50.0f, -50.0f, bZ);

glVertex3f(50.0f, -50.0f, bZ);

glVertex3f(50.0f, -50.0f, fZ);

// Левая секция

glNormal3f(1.0f, 0.0f, 0.0f);

glVertex3f(50.0f, 50.0f, fZ);

glVertex3f(50.0f, -50.0f, fZ);

glVertex3f(50.0f, -50.0f, bZ);

glVertex3f(50.0f, 50.0f, bZ);

// Правая секция

glNormal3f(-1.0f, 0.0f, 0.0f);

glVertex3f(-50.0f, 50.0f, fZ);

glVertex3f(-50.0f, 50.0f, bZ);

glVertex3f(-50.0f, -50.0f, bZ);

glVertex3f(-50.0f, -50.0f, fZ);

glEnd();

glFrontFace(GL\_CW); // элементы с обходом по часовой стрелке

// смотрят наружу

glBegin(GL\_QUADS);

// Задняя часть

// Параллельно оси z

glNormal3f(0.0f, 0.0f, -1.0f);

// Левая сторона

glVertex3f(-50.0f, 50.0f, bZ);

glVertex3f(-50.0f, -50.0f, bZ);

glVertex3f(-35.0f, -50.0f, bZ);

glVertex3f(-35.0f, 50.0f, bZ);

// Правая сторона

glVertex3f(50.0f, 50.0f, bZ);

glVertex3f(35.0f, 50.0f, bZ);

glVertex3f(35.0f, -50.0f, bZ);

glVertex3f(50.0f, -50.0f, bZ);

// Верх

glVertex3f(-35.0f, 50.0f, bZ);

glVertex3f(-35.0f, 35.0f, bZ);

glVertex3f(0.0f, 15.0f, bZ);

glVertex3f(0.0f, 30.0f, bZ);

glVertex3f(35.0f, 35.0f, bZ);

glVertex3f(35.0f, 50.0f, bZ);

glVertex3f(0.0f, 30.0f, bZ);

glVertex3f(0.0f, 15.0f, bZ);

// Низ

glVertex3f(-35.0f, -35.0f, bZ);

glVertex3f(-35.0f, -50.0f, bZ);

glVertex3f(35.0f, -50.0f, bZ);

glVertex3f(35.0f, -35.0f, bZ);

// Внутрення часть

/////////////////////////////

glColor3f(0.0f, 1.0f, 0.0f);

// Нормаль указывает на ось Y

glNormal3f(1.0f, 1.0f, 0.0f);

glVertex3f(-35.0f, 35.0f, fZ);

glVertex3f(0.0f, 15.0f, fZ);

glVertex3f(0.0f, 15.0f, bZ);

glVertex3f(-35.0f, 35.0f, bZ);

glNormal3f(-1.0f, 1.0f, 0.0f);

glVertex3f(0.0f, 15.0f, fZ);

glVertex3f(35.0f, 35.0f, fZ);

glVertex3f(35.0f, 35.0f, bZ);

glVertex3f(0.0f, 15.0f, bZ);

// Нижняя часть

glNormal3f(0.0f, 1.0f, 0.0f);

glVertex3f(-35.0f, -35.0f, fZ);

glVertex3f(-35.0f, -35.0f, bZ);

glVertex3f(35.0f, -35.0f, bZ);

glVertex3f(35.0f, -35.0f, fZ);

// Левая часть

glNormal3f(1.0f, 0.0f, 0.0f);

glVertex3f(-35.0f, 35.0f, fZ);

glVertex3f(-35.0f, 35.0f, bZ);

glVertex3f(-35.0f, -35.0f, bZ);

glVertex3f(-35.0f, -35.0f, fZ);

// Правая часть

glNormal3f(-1.0f, 0.0f, 0.0f);

glVertex3f(35.0f, 35.0f, fZ);

glVertex3f(35.0f, -35.0f, fZ);

glVertex3f(35.0f, -35.0f, bZ);

glVertex3f(35.0f, 35.0f, bZ);

glEnd();

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

// Восстанавливается состояние матрицы

glPopMatrix();

// Буфер-обмена

glutSwapBuffers();

}

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

{

glutInit(&argc, argv);

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

glutInitWindowSize(800, 600);

glutCreateWindow("Orthographic Projection");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

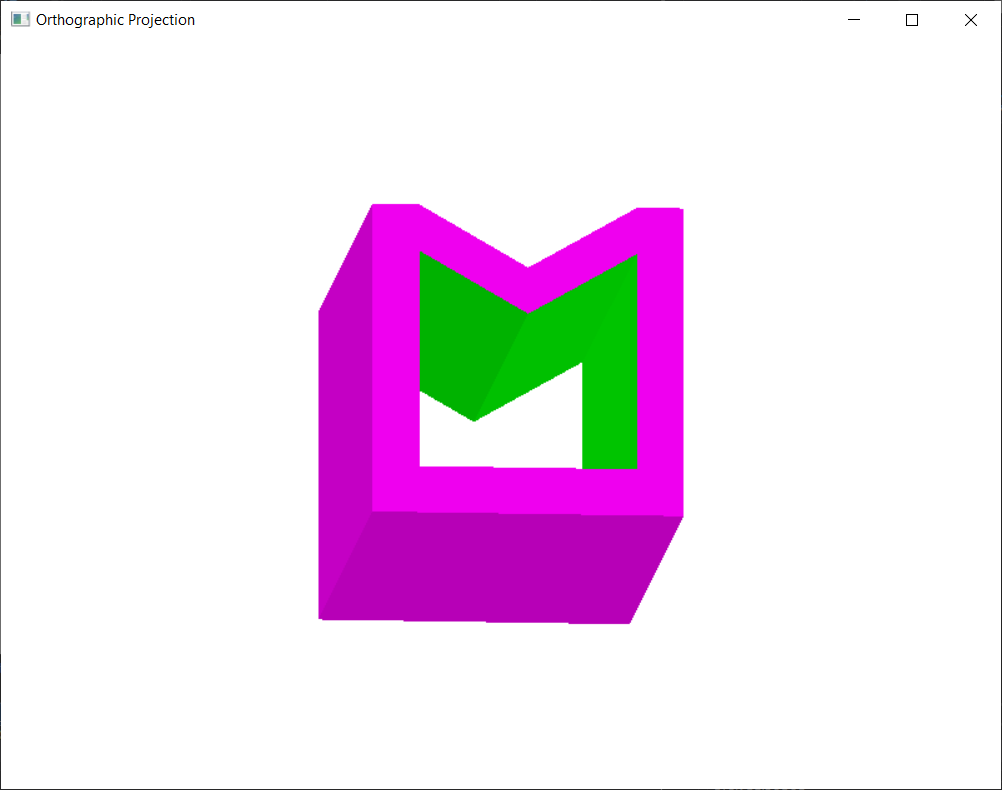
glutMainLoop();

return 0;

}

}

**Результат листинга 2:**



**Рисунок 2.** Результат листинга 2

**Листинг 3:**

#include "glew.h"

#include "glut.h"

namespace Listing\_3\_3

{

// Величина поворота

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

// Меняет наблюдаемый объем и поле просмотра

// Вызывается при изменении размеров окна

void ChangeSize(GLsizei w, GLsizei h)

{

GLfloat fAspect;

// Предотвращает деление на ноль

if (h == 0)

h = 1;

// Устанавливает поле просмотра по размерам окна

glViewport(0, 0, w, h);

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

// Обновляет стек матрицы проектирования

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Генерирует перспективную проекцию

gluPerspective(60.0f, fAspect, 1.0, 400.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

// Эта функция выполняет необходимую инициализацию в контексте

// визуализации. В данном случае устанавливается

// и инициализируется освещение сцены.

void SetupRC()

{

// Параметры освещения и координаты

GLfloat whiteLight[] = { 0.45f, 0.45f, 0.45f, 1.0f };

GLfloat sourceLight[] = { 0.25f, 0.25f, 0.25f, 1.0f };

GLfloat lightPos[] = { -50.f, 25.0f, 250.0f, 0.0f };

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

// Активизация освешения

glEnable(GL\_LIGHTING);

// Устанавливается и активизируется источник света 0

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

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

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(1.0f, 1.0f, 1.0f, 1.0f);

}

// Реагирует на клавиши со стрелками, двигая систему отсчета камеры

void SpecialKeys(int key, int x, int y)

{

if (key == GLUT\_KEY\_UP)

xRot -= 5.0f;

if (key == GLUT\_KEY\_DOWN)

xRot += 5.0f;

if (key == GLUT\_KEY\_LEFT)

yRot -= 5.0f;

if (key == GLUT\_KEY\_RIGHT)

yRot += 5.0f;

xRot = (GLfloat)((const int)xRot % 360);

yRot = (GLfloat)((const int)yRot % 360);

// Обновляем окно

glutPostRedisplay();

}

// Вызывается для рисования сцены

void RenderScene(void)

{

float fZ, bZ;

// Очищаем окно текущим цветом заливки

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

fZ = 100.0f;

bZ = -100.0f;

// Записывается состояние матрицы и выполняются повороты

glPushMatrix();

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

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

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

// В качестве текущего цвета рисования задает красный

glColor3f(1.0f, 0.0f, 1.0f);

// Лицевая часть

///////////////////////////////////

glBegin(GL\_QUADS);

// прямо вверх оси z

glNormal3f(0.0f, 0.0f, 1.0f);

// Левая сторона

glVertex3f(-50.0f, 50.0f, fZ);

glVertex3f(-50.0f, -50.0f, fZ);

glVertex3f(-35.0f, -50.0f, fZ);

glVertex3f(-35.0f, 50.0f, fZ);

// Правая сторона

glVertex3f(50.0f, 50.0f, fZ);

glVertex3f(35.0f, 50.0f, fZ);

glVertex3f(35.0f, -50.0f, fZ);

glVertex3f(50.0f, -50.0f, fZ);

// Верх

glVertex3f(-35.0f, 50.0f, fZ);

glVertex3f(-35.0f, 35.0f, fZ);

glVertex3f(0.0f, 15.0f, fZ);

glVertex3f(0.0f, 30.0f, fZ);

glVertex3f(35.0f, 35.0f, fZ);

glVertex3f(35.0f, 50.0f, fZ);

glVertex3f(0.0f, 30.0f, fZ);

glVertex3f(0.0f, 15.0f, fZ);

// Низ

glVertex3f(-35.0f, -35.0f, fZ);

glVertex3f(-35.0f, -50.0f, fZ);

glVertex3f(35.0f, -50.0f, fZ);

glVertex3f(35.0f, -35.0f, fZ);

// Верхняя длинная часть//

//////////////////////////

// Нормаль указывает на ось Y

glNormal3f(0.0f, 1.0f, 0.0f);

glVertex3f(-50.0f, 50.0f, fZ);

glVertex3f(-35.0f, 50.0f, fZ);

glVertex3f(-35.0f, 50.0f, bZ);

glVertex3f(-50.0f, 50.0f, bZ);

glNormal3f(0.0f, 1.0f, 0.0f);

glVertex3f(35.0f, 50.0f, fZ);

glVertex3f(50.0f, 50.0f, fZ);

glVertex3f(50.0f, 50.0f, bZ);

glVertex3f(35.0f, 50.0f, bZ);

glNormal3f(1.0f, 1.0f, 0.0f);

glVertex3f(-35.0f, 50.0f, fZ);

glVertex3f(0.0f, 30.0f, fZ);

glVertex3f(0.0f, 30.0f, bZ);

glVertex3f(-35.0f, 50.0f, bZ);

glNormal3f(-1.0f, 1.0f, 0.0f);

glVertex3f(0.0f, 30.0f, fZ);

glVertex3f(35.0f, 50.0f, fZ);

glVertex3f(35.0f, 50.0f, bZ);

glVertex3f(0.0f, 30.0f, bZ);

// Низ

glNormal3f(0.0f, -1.0f, 0.0f);

glVertex3f(-50.0f, -50.0f, fZ);

glVertex3f(-50.0f, -50.0f, bZ);

glVertex3f(50.0f, -50.0f, bZ);

glVertex3f(50.0f, -50.0f, fZ);

// Левая сторона

glNormal3f(1.0f, 0.0f, 0.0f);

glVertex3f(50.0f, 50.0f, fZ);

glVertex3f(50.0f, -50.0f, fZ);

glVertex3f(50.0f, -50.0f, bZ);

glVertex3f(50.0f, 50.0f, bZ);

// Правая сторона

glNormal3f(-1.0f, 0.0f, 0.0f);

glVertex3f(-50.0f, 50.0f, fZ);

glVertex3f(-50.0f, 50.0f, bZ);

glVertex3f(-50.0f, -50.0f, bZ);

glVertex3f(-50.0f, -50.0f, fZ);

glEnd();

glFrontFace(GL\_CW); // элементы с обходом по часовой стрелке

//смотрят наружу

glBegin(GL\_QUADS);

// Задняя часть

// Нормали направлены прямо вверх оси z

glNormal3f(0.0f, 0.0f, -1.0f);

// Левая сторона

glVertex3f(-50.0f, 50.0f, bZ);

glVertex3f(-50.0f, -50.0f, bZ);

glVertex3f(-35.0f, -50.0f, bZ);

glVertex3f(-35.0f, 50.0f, bZ);

// Правая сторона

glVertex3f(50.0f, 50.0f, bZ);

glVertex3f(35.0f, 50.0f, bZ);

glVertex3f(35.0f, -50.0f, bZ);

glVertex3f(50.0f, -50.0f, bZ);

// Верх

glVertex3f(-35.0f, 50.0f, bZ);

glVertex3f(-35.0f, 35.0f, bZ);

glVertex3f(0.0f, 15.0f, bZ);

glVertex3f(0.0f, 30.0f, bZ);

glVertex3f(35.0f, 35.0f, bZ);

glVertex3f(35.0f, 50.0f, bZ);

glVertex3f(0.0f, 30.0f, bZ);

glVertex3f(0.0f, 15.0f, bZ);

// Низ

glVertex3f(-35.0f, -35.0f, bZ);

glVertex3f(-35.0f, -50.0f, bZ);

glVertex3f(35.0f, -50.0f, bZ);

glVertex3f(35.0f, -35.0f, bZ);

// Внутренняя часть//

///////////////////////////

glColor3f(0.0f, 1.0f, 0.0f);

// Нормаль направлена вверх оси Y

glNormal3f(1.0f, 1.0f, 0.0f);

glVertex3f(-35.0f, 35.0f, fZ);

glVertex3f(0.0f, 15.0f, fZ);

glVertex3f(0.0f, 15.0f, bZ);

glVertex3f(-35.0f, 35.0f, bZ);

glNormal3f(-1.0f, 1.0f, 0.0f);

glVertex3f(0.0f, 15.0f, fZ);

glVertex3f(35.0f, 35.0f, fZ);

glVertex3f(35.0f, 35.0f, bZ);

glVertex3f(0.0f, 15.0f, bZ);

// Низ

glNormal3f(0.0f, 1.0f, 0.0f);

glVertex3f(-35.0f, -35.0f, fZ);

glVertex3f(-35.0f, -35.0f, bZ);

glVertex3f(35.0f, -35.0f, bZ);

glVertex3f(35.0f, -35.0f, fZ);

//Левая сторона

glNormal3f(1.0f, 0.0f, 0.0f);

glVertex3f(-35.0f, 35.0f, fZ);

glVertex3f(-35.0f, 35.0f, bZ);

glVertex3f(-35.0f, -35.0f, bZ);

glVertex3f(-35.0f, -35.0f, fZ);

// Правая сторона

glNormal3f(-1.0f, 0.0f, 0.0f);

glVertex3f(35.0f, 35.0f, fZ);

glVertex3f(35.0f, -35.0f, fZ);

glVertex3f(35.0f, -35.0f, bZ);

glVertex3f(35.0f, 35.0f, bZ);

glEnd();

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

//часовой стрелки направлены наружу

// Восстанавливается состояние матрицы

glPopMatrix();

// Буфер обмена

glutSwapBuffers();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB |

GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("Perspective Projection");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

glutMainLoop();

return 0;

}

}

**Результат листинга 3:**



**Рисунок 3.** Результат листинга 3

**Листинг 4:**

#include "glew.h"

#include "glut.h"

#include <math.h>

namespace Listing\_3\_4

{

// 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;

static float fSatelliteRot = 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);

glPushMatrix();

// 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 += 3.0f;

if (fMoonRot > 360.0f)

fMoonRot -= 360.0f;

glutSolidSphere(6.0f, 30, 17);

glPopMatrix();

glColor3ub(225, 150, 100);

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

glTranslatef(45.0f, 0.0f, 0.0f);

fSatelliteRot -= 5.0f;

if (fSatelliteRot < 0.0f)

fSatelliteRot += 360.0f;

glutSolidSphere(4.0f, 30, 17);

// Восстанавливается состояние матрицы

glPopMatrix(); // Матрица наблюдения модели

// Шаг по орбите Земли равен пяти градусам

fEarthRot += 1.0f;

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 glCo\_lor

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(10, 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();

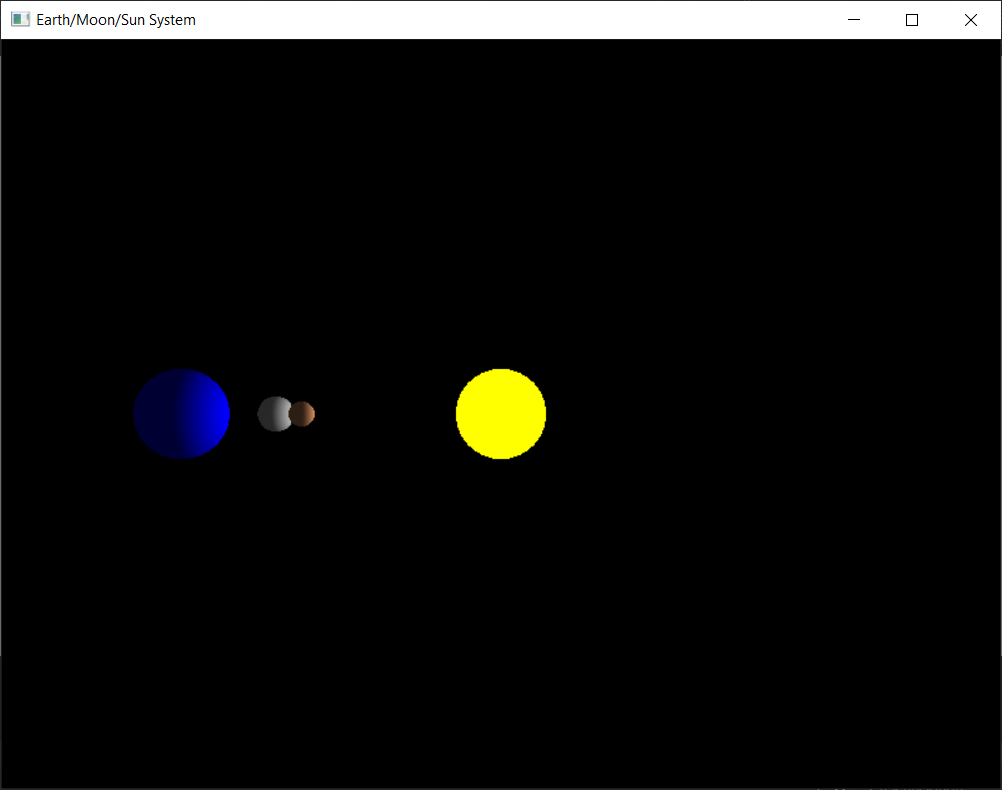
glutMainLoop();

return 0;

}

}

**Результат листинга 4:**



**Рисунок 4.** Результат листинга 4

**Листинг 5:**

#include "glew.h"

#include "glut.h"

#include <windows.h>

#include <math.h>

#include <random>

#include <time.h>

namespace Listing\_3\_5

{

// Используемые константы

#define GLT\_PI 3.14159265358979323846

#define GLT\_PI\_DIV\_180 0.017453292519943296

#define GLT\_INV\_PI\_DIV\_180 57.2957795130823229

#define gltDegToRad(x) ((x)\*GLT\_PI\_DIV\_180)

// Некоторые типы данных

typedef GLfloat GLTVector2[2];//Двукомпонентный вектор с плавающей запятой

typedef GLfloat GLTVector3[3];//Трехомпонентный вектор с плавающей запятой

typedef GLfloat GLTVector4[4];//Четырехкомпонентный вектор с плавающей //запятой

typedef GLfloat GLTMatrix[16];// Основноц столбец матрицы 4х4 с плавающей //запятой

void gltLoadIdentityMatrix(GLTMatrix m)

{

static GLTMatrix identity = {

1.0f, 0.0f, 0.0f, 0.0f,

0.0f, 1.0f, 0.0f, 0.0f,

0.0f, 0.0f, 1.0f, 0.0f,

0.0f, 0.0f, 0.0f, 1.0f };

memcpy(m, identity, sizeof(GLTMatrix));

};

void gltTransformPoint(const GLTVector3 vSrcVector,

const GLTMatrix mMatrix, GLTVector3 vOut)

{

vOut[0] = mMatrix[0] \* vSrcVector[0] + mMatrix[4] \*

vSrcVector[1] + mMatrix[8] \* vSrcVector[2] + mMatrix[12];

vOut[1] = mMatrix[1] \* vSrcVector[0] + mMatrix[5] \*

vSrcVector[1] + mMatrix[9] \* vSrcVector[2] + mMatrix[13];

vOut[2] = mMatrix[2] \* vSrcVector[0] + mMatrix[6] \*

vSrcVector[1] + mMatrix[10] \* vSrcVector[2] + mMatrix[14];

};

void gltRotationMatrix(float angle, float x, float y,

float z, GLTMatrix mMatrix)

{

float vecLength, sinSave, cosSave, oneMinusCos;

float xx, yy, zz, xy, yz, zx, xs, ys, zs;

// Если нулевой вектор проходит, то …

if (x == 0.0f && y == 0.0f && z == 0.0f)

{

gltLoadIdentityMatrix(mMatrix);

return;

}

// Вектор масштабирования

vecLength = (float)sqrt(x \* x + y \* y + z \* z);

// Нормализованная матрица вращения

x /= vecLength;

y /= vecLength;

z /= vecLength;

sinSave = (float)sin(angle);

cosSave = (float)cos(angle);

oneMinusCos = 1.0f - cosSave;

xx = x \* x;

yy = y \* y;

zz = z \* z;

xy = x \* y;

yz = y \* z;

zx = z \* x;

xs = x \* sinSave;

ys = y \* sinSave;

zs = z \* sinSave;

mMatrix[0] = (oneMinusCos \* xx) + cosSave;

mMatrix[4] = (oneMinusCos \* xy) - zs;

mMatrix[8] = (oneMinusCos \* zx) + ys;

mMatrix[12] = 0.0f;

mMatrix[1] = (oneMinusCos \* xy) + zs;

mMatrix[5] = (oneMinusCos \* yy) + cosSave;

mMatrix[9] = (oneMinusCos \* yz) - xs;

mMatrix[13] = 0.0f;

mMatrix[2] = (oneMinusCos \* zx) - ys;

mMatrix[6] = (oneMinusCos \* yz) + xs;

mMatrix[10] = (oneMinusCos \* zz) + cosSave;

mMatrix[14] = 0.0f;

mMatrix[3] = 0.0f;

mMatrix[7] = 0.0f;

mMatrix[11] = 0.0f;

mMatrix[15] = 1.0f;

};

typedef struct { // Контейнер системы отсчета

GLTVector3 vLocation;

GLTVector3 vUp;

GLTVector3 vForward;

} GLTFrame;

// Прорисовка тора(бублика), с использованием текущей структуры 1D для легкого затемнения света

void DrawTorus(GLTMatrix mTransform)

{

GLfloat majorRadius = 0.35f;

GLfloat minorRadius = 0.15f;

GLint numMajor = 40;

GLint numMinor = 20;

GLTVector3 objectVertex; // Вершина, находящаяся в поле зрения

GLTVector3 transformedVertex; // Новая измененная вершина

double majorStep = 2.0f \* GLT\_PI / numMajor;

double minorStep = 2.0f \* GLT\_PI / numMinor;

int i, j;

for (i = 0; i < numMajor; ++i)

{

double a0 = i \* majorStep;

double a1 = a0 + majorStep;

GLfloat x0 = (GLfloat)cos(a0);

GLfloat y0 = (GLfloat)sin(a0);

GLfloat x1 = (GLfloat)cos(a1);

GLfloat y1 = (GLfloat)sin(a1);

glBegin(GL\_TRIANGLE\_STRIP);

for (j = 0; j <= numMinor; ++j)

{

double b = j \* minorStep;

GLfloat c = (GLfloat)cos(b);

GLfloat r = minorRadius \* c + majorRadius;

GLfloat z = minorRadius \* (GLfloat)sin(b);

// Первая точка

objectVertex[0] = x0 \* r;

objectVertex[1] = y0 \* r;

objectVertex[2] = z;

gltTransformPoint(objectVertex, mTransform,

transformedVertex);

glVertex3fv(transformedVertex);

// Вторая точка

objectVertex[0] = x1 \* r;

objectVertex[1] = y1 \* r;

objectVertex[2] = z;

gltTransformPoint(objectVertex, mTransform,

transformedVertex);

glVertex3fv(transformedVertex);

}

glEnd();

}

}

// Вызывается для рисования сцены

void RenderScene(void)

{

GLTMatrix transformationMatrix; // Матрица поворота

static GLfloat yRot = 0.0f; // Угол поворота, задействованный

//в

//анимации

static GLfloat delta = 5.0f;

yRot += delta;

delta += 1 + rand() % (2 - 1 + 1);

delta \*= pow(-1, rand());

if (delta < 0)

delta = 1.0f;

// Очистка окна ткущим цветом очистки

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

// Строим матрицу поворота

gltRotationMatrix(gltDegToRad(yRot), 0.0f, 1.0f, 0.0f, transformationMatrix);

transformationMatrix[12] = 0.0f;

transformationMatrix[13] = 0.0f;

transformationMatrix[14] = -2.5f;

DrawTorus(transformationMatrix);

// Буфер обмена

glutSwapBuffers();

}

// Функция выполняет всю необходимую инициализацию в контексте

// визуализации

void SetupRC()

{

// Голубой фон

glClearColor(0.0f, 0.0f, .50f, 1.0f);

// Все рисуется в каркасном виде

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

}

///////////////////////////////////////////////////////

////

// Вызывается библиотекой GLUT в холостом состоянии

//(окно не меняет размера и не перемещается)

void TimerFunction(int value)

{

// Перерисовывает сцену с новыми координатами

glutPostRedisplay();

glutTimerFunc(33, TimerFunction, 1);

}

void ChangeSize(int w, int h)

{

GLfloat fAspect;

// Предотвращает деление на нуль, когда окно слишком маленькое

// (нельзя сделать окно нулевой ширины).

if (h == 0)

h = 1;

glViewport(0, 0, w, h);

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

// Система координат обновляется перед модификацией

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Генерируется перспективная проекция

gluPerspective(35.0f, fAspect, 1.0f, 50.0f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

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

{

srand(time(NULL));

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB |

GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("Manual Transformations Demo");

glutReshapeFunc(ChangeSize);

glutDisplayFunc(RenderScene);

SetupRC();

glutTimerFunc(33, TimerFunction, 1);

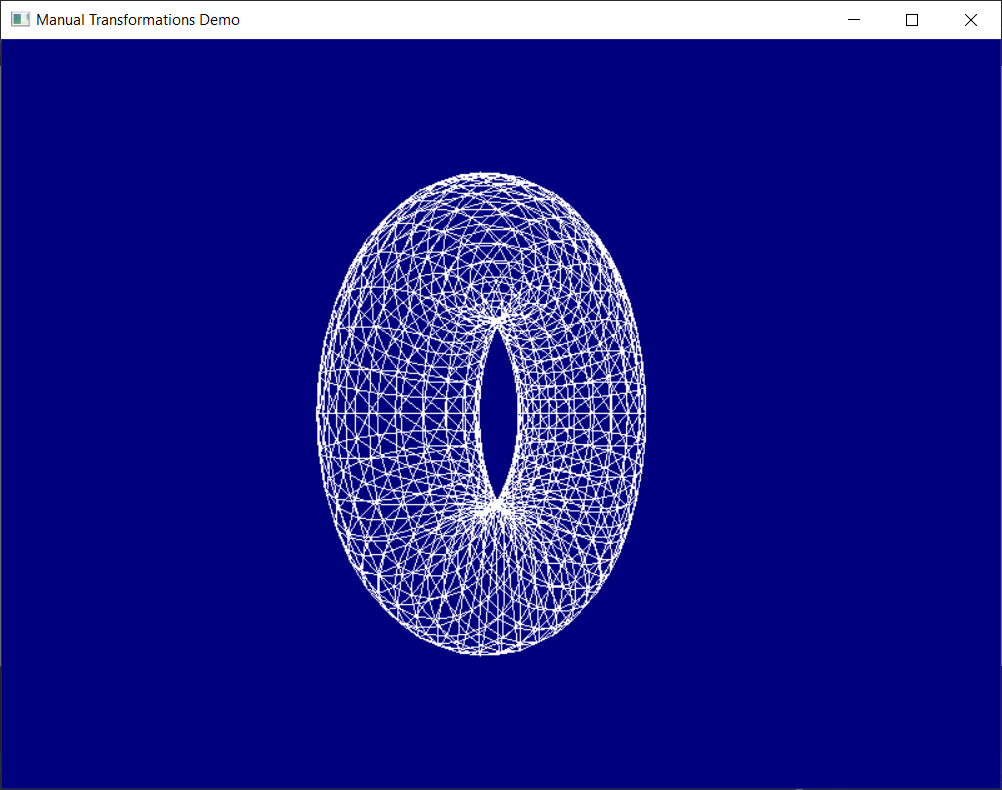
glutMainLoop();

return 0;

}

}

**Результат листинга 5:**



**Рисунок 5.** Результат листинга 5

**Листинг 7:**

#include "glew.h"

#include "glut.h"

#include <windows.h>

#include <math.h>

namespace Listing\_3\_7

{

///////////////////////////////////////////////////////

// Используемые константы

#define GLT\_PI 3.14159265358979323846

#define GLT\_PI\_DIV\_180 0.017453292519943296

#define GLT\_INV\_PI\_DIV\_180 57.2957795130823229

///////////////////////////////////////////////////////

////////////////////////

// Полезные клавиши быстрого вызова и макросы

// Измерения в радианах, но нам нужен способ, чтобы поменять обратно

#define gltDegToRad(x) ((x)\*GLT\_PI\_DIV\_180)

#define gltRadToDeg(x) ((x)\*GLT\_INV\_PI\_DIV\_180)

// Некоторые типы данных

typedef GLfloat GLTVector2[2];//Двукомпонентный вектор с плавающей запятой

typedef GLfloat GLTVector3[3];//Трехомпонентный вектор с плавающей запятой

typedef GLfloat GLTVector4[4];//Четырехкомпонентный вектор с плавающей //запятой

typedef GLfloat GLTMatrix[16];// Основноц столбец матрицы 4х4 с плавающей //запятой

typedef struct { // Рамка для контейнера ссылок

GLTVector3 vLocation;

GLTVector3 vUp;

GLTVector3 vForward;

} GLTFrame;

#define NUM\_SPHERES 50

GLTFrame spheres[NUM\_SPHERES];

GLTFrame frameCamera;

// Вычислить векторное произведение двух векторов

void gltVectorCrossProduct(const GLTVector3 vU, const

GLTVector3 vV, GLTVector3 vResult)

{

vResult[0] = vU[1] \* vV[2] - vV[1] \* vU[2];

vResult[1] = -vU[0] \* vV[2] + vV[0] \* vU[2];

vResult[2] = vU[0] \* vV[1] - vV[0] \* vU[1];

}

void gltLoadIdentityMatrix(GLTMatrix m)

{

static GLTMatrix identity = { 1.0f, 0.0f, 0.0f,

0.0f,

0.0f, 1.0f, 0.0f,

0.0f,

0.0f, 0.0f, 1.0f,

0.0f,

0.0f, 0.0f, 0.0f,

1.0f };

memcpy(m, identity, sizeof(GLTMatrix));

}

// Создает матрицу вращения 4x4, используются радианы, а не в градусы

void gltRotationMatrix(float angle, float x, float y,

float z, GLTMatrix mMatrix)

{

float vecLength, sinSave, cosSave, oneMinusCos;

float xx, yy, zz, xy, yz, zx, xs, ys, zs;

// Если нулевой вектор проходит, то...

if (x == 0.0f && y == 0.0f && z == 0.0f)

{

gltLoadIdentityMatrix(mMatrix);

return;

}

// Масштабирование вектора

vecLength = (float)sqrt(x \* x + y \* y + z \* z);

// Нормарованная матрица вращения

x /= vecLength;

y /= vecLength;

z /= vecLength;

sinSave = (float)sin(angle);

cosSave = (float)cos(angle);

oneMinusCos = 1.0f - cosSave;

xx = x \* x;

yy = y \* y;

zz = z \* z;

xy = x \* y;

yz = y \* z;

zx = z \* x;

xs = x \* sinSave;

ys = y \* sinSave;

zs = z \* sinSave;

mMatrix[0] = (oneMinusCos \* xx) + cosSave;

mMatrix[4] = (oneMinusCos \* xy) - zs;

mMatrix[8] = (oneMinusCos \* zx) + ys;

mMatrix[12] = 0.0f;

mMatrix[1] = (oneMinusCos \* xy) + zs;

mMatrix[5] = (oneMinusCos \* yy) + cosSave;

mMatrix[9] = (oneMinusCos \* yz) - xs;

mMatrix[13] = 0.0f;

mMatrix[2] = (oneMinusCos \* zx) - ys;

mMatrix[6] = (oneMinusCos \* yz) + xs;

mMatrix[10] = (oneMinusCos \* zz) + cosSave;

mMatrix[14] = 0.0f;

mMatrix[3] = 0.0f;

mMatrix[7] = 0.0f;

mMatrix[11] = 0.0f;

mMatrix[15] = 1.0f;

}

// Поворот вектора с использованием матрицы 4х4. Перевод столбца игнорируется

void gltRotateVector(const GLTVector3 vSrcVector, const

GLTMatrix mMatrix, GLTVector3 vOut)

{

vOut[0] = mMatrix[0] \* vSrcVector[0] + mMatrix[4] \*

vSrcVector[1] + mMatrix[8] \* vSrcVector[2];

vOut[1] = mMatrix[1] \* vSrcVector[0] + mMatrix[5] \*

vSrcVector[1] + mMatrix[9] \* vSrcVector[2];

vOut[2] = mMatrix[2] \* vSrcVector[0] + mMatrix[6] \*

vSrcVector[1] + mMatrix[10] \* vSrcVector[2];

}

///////////////////////////////////////////////////////

//

//Это простое перемещение вперед вдоль переднего вектора.

void gltInitFrame(GLTFrame\* pFrame)

{

pFrame->vLocation[0] = 0.0f;

pFrame->vLocation[1] = 0.0f;

pFrame->vLocation[2] = 0.0f;

pFrame->vUp[0] = 0.0f;

pFrame->vUp[1] = 1.0f;

pFrame->vUp[2] = 0.0f;

pFrame->vForward[0] = 0.0f;

pFrame->vForward[1] = 0.0f;

pFrame->vForward[2] = -1.0f;

}

///////////////////////////////////////////////////////

////////////

// Выводит матрицу преобразования 4x4 от системы отсчета

void gltGetMatrixFromFrame(GLTFrame\* pFrame, GLTMatrix

mMatrix)

{

GLTVector3 vXAxis; // Производные оси X

// Рассчитать оси X

gltVectorCrossProduct(pFrame->vUp, pFrame->vForward, vXAxis);

// Просто заполнения матрицы столбцом вектора X

memcpy(mMatrix, vXAxis, sizeof(GLTVector3));

mMatrix[3] = 0.0f;

// столбец вектора Y

memcpy(mMatrix + 4, pFrame->vUp, sizeof(GLTVector3));

mMatrix[7] = 0.0f;

// Столбец вектора Z

memcpy(mMatrix + 8, pFrame->vForward, sizeof(GLTVector3));

mMatrix[11] = 0.0f;

// Перевод/Расположение вектора

memcpy(mMatrix + 12, pFrame->vLocation, sizeof(GLTVector3));

mMatrix[15] = 1.0f;

}

///////////////////////////////////////////////////////

/////////////

// Использовать действующее лицо, учитывая его преобразования системы отсчета

void gltApplyActorTransform(GLTFrame\* pFrame)

{

GLTMatrix mTransform;

gltGetMatrixFromFrame(pFrame, mTransform);

glMultMatrixf(mTransform);

}

///////////////////////////////////////////////////////

///////////

// Применение камеры преобразования данной системы отсчета.Это в

//значительной степени альтернативная реализация gluLookAt использующую тип

//float вместо double и имеющую прямой вектор вместо указателя.

void gltApplyCameraTransform(GLTFrame\* pCamera)

{

GLTMatrix mMatrix;

GLTVector3 vAxisX;

GLTVector3 zFlipped;

zFlipped[0] = -pCamera->vForward[0];

zFlipped[1] = -pCamera->vForward[1];

zFlipped[2] = -pCamera->vForward[2];

// Вывести X вектор

gltVectorCrossProduct(pCamera->vUp, zFlipped, vAxisX);

// Заполнение матрицы, обратите внимание, это поворот и перенос

mMatrix[0] = vAxisX[0];

mMatrix[4] = vAxisX[1];

mMatrix[8] = vAxisX[2];

mMatrix[12] = 0.0f;

mMatrix[1] = pCamera->vUp[0];

mMatrix[5] = pCamera->vUp[1];

mMatrix[9] = pCamera->vUp[2];

mMatrix[13] = 0.0f;

mMatrix[2] = zFlipped[0];

mMatrix[6] = zFlipped[1];

mMatrix[10] = zFlipped[2];

mMatrix[14] = 0.0f;

mMatrix[3] = 0.0f;

mMatrix[7] = 0.0f;

mMatrix[11] = 0.0f;

mMatrix[15] = 1.0f;

// Делаем первый поворот

glMultMatrixf(mMatrix);

// Обратные преобразования

glTranslatef(-pCamera->vLocation[0], -pCamera->vLocation[1], -pCamera->vLocation[2]);

}

void gltMoveFrameForward(GLTFrame\* pFrame, GLfloat

fStep)

{

pFrame->vLocation[0] += pFrame->vForward[0] \*

fStep;

pFrame->vLocation[1] += pFrame->vForward[1] \*

fStep;

pFrame->vLocation[2] += pFrame->vForward[2] \*

fStep;

}

///////////////////////////////////////////////////////

//

// Перемещение системы отсчета вверх оси Y

void gltMoveFrameUp(GLTFrame\* pFrame, GLfloat fStep)

{

pFrame->vLocation[0] += pFrame->vUp[0] \* fStep;

pFrame->vLocation[1] += pFrame->vUp[1] \* fStep;

pFrame->vLocation[2] += pFrame->vUp[2] \* fStep;

}

///////////////////////////////////////////////////////

// Перемещение системы отсчета вдоль оси X

void gltMoveFrameRight(GLTFrame\* pFrame, GLfloat fStep)

{

GLTVector3 vCross;

gltVectorCrossProduct(pFrame->vUp, pFrame->vForward, vCross);

pFrame->vLocation[0] += vCross[0] \* fStep;

pFrame->vLocation[1] += vCross[1] \* fStep;

pFrame->vLocation[2] += vCross[2] \* fStep;

}

///////////////////////////////////////////////////////

//

// Перевод рамки в мировых координатах

void gltTranslateFrameWorld(GLTFrame\* pFrame, GLfloat

x, GLfloat y, GLfloat z)

{

pFrame->vLocation[0] += x; pFrame->vLocation[1]

+= y; pFrame->vLocation[2] += z;

}

///////////////////////////////////////////////////////

//

// Перевод рамки в локальных координатах

void gltTranslateFrameLocal(GLTFrame\* pFrame, GLfloat

x, GLfloat y, GLfloat z)

{

gltMoveFrameRight(pFrame, x);

gltMoveFrameUp(pFrame, y);

gltMoveFrameForward(pFrame, z);

}

///////////////////////////////////////////////////////

//

// Поворот рамки вокруг локальной оси Y

void gltRotateFrameLocalY(GLTFrame\* pFrame, GLfloat

fAngle)

{

GLTMatrix mRotation;

GLTVector3 vNewForward;

gltRotationMatrix((float)gltDegToRad(fAngle), 0.0f,

1.0f, 0.0f, mRotation);

gltRotationMatrix(fAngle, pFrame->vUp[0], pFrame->vUp[1], pFrame->vUp[2], mRotation);

gltRotateVector(pFrame->vForward, mRotation, vNewForward);

memcpy(pFrame->vForward, vNewForward, sizeof(GLTVector3));

}

///////////////////////////////////////////////////////

///

// Поворот рамки вокруг локальной оси X

void gltRotateFrameLocalX(GLTFrame\* pFrame, GLfloat

fAngle)

{

GLTMatrix mRotation;

GLTVector3 vCross;

gltVectorCrossProduct(vCross, pFrame->vUp, pFrame->vForward);

gltRotationMatrix(fAngle, vCross[0], vCross[1],

vCross[2], mRotation);

GLTVector3 vNewVect;

// Встроеную матрицу 3x3 умножаем для вращения

vNewVect[0] = mRotation[0] \* pFrame->vForward[0] +

mRotation[4] \* pFrame->vForward[1] + mRotation[8] \*

pFrame->vForward[2];

vNewVect[1] = mRotation[1] \* pFrame->vForward[0] +

mRotation[5] \* pFrame->vForward[1] + mRotation[9] \*

pFrame->vForward[2];

vNewVect[2] = mRotation[2] \* pFrame->vForward[0] +

mRotation[6] \* pFrame->vForward[1] + mRotation[10] \*

pFrame->vForward[2];

memcpy(pFrame->vForward, vNewVect, sizeof(GLfloat) \* 3);

// Обновление вектора, направленного вверх

vNewVect[0] = mRotation[0] \* pFrame->vUp[0] + mRotation[4] \* pFrame->vUp[1] + mRotation[8] \* pFrame->vUp[2];

vNewVect[1] = mRotation[1] \* pFrame->vUp[0] + mRotation[5] \* pFrame->vUp[1] + mRotation[9] \* pFrame->vUp[2];

vNewVect[2] = mRotation[2] \* pFrame->vUp[0] + mRotation[6] \* pFrame->vUp[1] + mRotation[10] \* pFrame->vUp[2];

memcpy(pFrame->vUp, vNewVect, sizeof(GLfloat) \* 3);

}

///////////////////////////////////////////////////////

//////

// Поворот рамки вокруг локальной оси Z

void gltRotateFrameLocalZ(GLTFrame\* pFrame, GLfloat

fAngle)

{

GLTMatrix mRotation;

// Повернуть требуется только вектор, направленный вверх

gltRotationMatrix(fAngle, pFrame->vForward[0],

pFrame->vForward[1], pFrame->vForward[2], mRotation);

GLTVector3 vNewVect;

vNewVect[0] = mRotation[0] \* pFrame->vUp[0] + mRotation[4] \* pFrame->vUp[1] + mRotation[8] \* pFrame->vUp[2];

vNewVect[1] = mRotation[1] \* pFrame->vUp[0] + mRotation[5] \* pFrame->vUp[1] + mRotation[9] \* pFrame->vUp[2];

vNewVect[2] = mRotation[2] \* pFrame->vUp[0] + mRotation[6] \* pFrame->vUp[1] + mRotation[10] \* pFrame->vUp[2];

memcpy(pFrame->vUp, vNewVect, sizeof(GLfloat) \* 3);

}

// Скалярное масштабирование вектора

void gltScaleVector(GLTVector3 vVector, const GLfloat

fScale)

{

vVector[0] \*= fScale; vVector[1] \*= fScale; vVector[2] \*= fScale;

}

// Возвращает длину вектора в квадрате

GLfloat gltGetVectorLengthSqrd(const GLTVector3 vVector)

{

return (vVector[0] \* vVector[0]) + (vVector[1] \* vVector[1]) + (vVector[2] \* vVector[2]);

}

// Возвращает длину вектора

GLfloat gltGetVectorLength(const GLTVector3 vVector)

{

return

(GLfloat)sqrt(gltGetVectorLengthSqrd(vVector));

}

// Масштабирование вектора по длине - создание единичсного вектора

void gltNormalizeVector(GLTVector3 vNormal)

{

GLfloat fLength = 1.0f / gltGetVectorLength(vNormal);

gltScaleVector(vNormal, fLength);

}

void gltDrawTorus(GLfloat majorRadius, GLfloat minorRadius, GLint numMajor, GLint numMinor)

{

GLTVector3 vNormal;

double majorStep = 2.0f \* GLT\_PI / numMajor;

double minorStep = 2.0f \* GLT\_PI / numMinor;

int i, j;

for (i = 0; i < numMajor; ++i)

{

double a0 = i \* majorStep;

double a1 = a0 + majorStep;

GLfloat x0 = (GLfloat)cos(a0);

GLfloat y0 = (GLfloat)sin(a0);

GLfloat x1 = (GLfloat)cos(a1);

GLfloat y1 = (GLfloat)sin(a1);

glBegin(GL\_TRIANGLE\_STRIP);

for (j = 0; j <= numMinor; ++j)

{

double b = j \* minorStep;

GLfloat c = (GLfloat)cos(b);

GLfloat r = minorRadius \* c + majorRadius;

GLfloat z = minorRadius \* (GLfloat)

sin(b);

// First point

glTexCoord2f((float)(i) / (float)(numMajor),

(float)(j) / (float)(numMinor));

vNormal[0] = x0 \* c;

vNormal[1] = y0 \* c;

vNormal[2] = z / minorRadius;

gltNormalizeVector(vNormal);

glNormal3fv(vNormal);

glVertex3f(x0 \* r, y0 \* r, z);

glTexCoord2f((float)(i + 1) / (float)(numMajor),

(float)(j) / (float)(numMinor));

vNormal[0] = x1 \* c;

vNormal[1] = y1 \* c;

vNormal[2] = z / minorRadius;

glNormal3fv(vNormal);

glVertex3f(x1 \* r, y1 \* r, z);

}

glEnd();

}

}

///////////////////////////////////////////////////////

///////////

// Эта функция выполняет необходимую инициализацию в контексте

// визуализации

void SetupRC()

{

int iSphere;

// Голубоватый фон

glClearColor(0.0f, 0.0f, .50f, 1.0f);

// Все рисуется в каркасном виде

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

gltInitFrame(&frameCamera); // Инициализируется камера

// Случайным образом размещаются жители-сферы

for (iSphere = 0; iSphere < NUM\_SPHERES; iSphere++)

{

gltInitFrame(&spheres[iSphere]); // Инициализируется камера

// Выбираются случайные положения между -20 и 20 с шагом 0.1

spheres[iSphere].vLocation[0] = (float)((rand()

% 400) - 200) \* 0.1f;

spheres[iSphere].vLocation[1] = 0.0f;

spheres[iSphere].vLocation[2] = (float)((rand()

% 400) - 200) \* 0.1f;

}

}

///////////////////////////////////////////////////////

////

// Рисуется земля с координатной сеткой

void DrawGround(void)

{

GLfloat fExtent = 20.0f;

GLfloat fStep = 1.0f;

GLfloat y = -0.4f;

GLint iLine;

glBegin(GL\_LINES);

for (iLine = -fExtent; iLine <= fExtent; iLine +=

fStep)

{

glVertex3f(iLine, y, fExtent); // Рисуются z - дорожки

glVertex3f(iLine, y, -fExtent);

glVertex3f(fExtent, y, iLine);

glVertex3f(-fExtent, y, iLine);

}

glEnd();

}

// Вызывается для рисования сцены

void RenderScene(void)

{

int i;

static GLfloat yRot = 0.0f; // Используемый в анимации угол поворота

yRot += 0.5f;

// Очищаем окно текущим цветом очистки

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

glPushMatrix();

gltApplyCameraTransform(&frameCamera);

// Рисуем землю

DrawGround();

// Рисуем случайным образом расположенные сферы

for (i = 0; i < NUM\_SPHERES; i++)

{

glPushMatrix();

gltApplyActorTransform(&spheres[i]);

glutSolidSphere(0.1f, 13, 26);

glPopMatrix();

}

glPushMatrix();

glTranslatef(0.0f, 0.0f, -2.5f);

glPushMatrix();

glRotatef(-yRot \* 2.0f, 0.0f, 1.0f, 0.0f);

glTranslatef(1.0f, 0.0f, 0.0f);

glutSolidSphere(0.1f, 13, 26);

glPopMatrix();

glPushMatrix();

glRotatef(-yRot \* 2.0f, 1.0f, 0.0f, 0.0f);

glTranslatef(0.0f, 1.0f, 0.0f);

glutSolidSphere(0.1f, 13, 26);

glPopMatrix();

glPushMatrix();

glRotatef(-yRot \* 2.0f, 0.0f, 0.0f, 1.0f);

glTranslatef(1.0f, 0.0f, 0.0f);

glutSolidSphere(0.1f, 13, 26);

glPopMatrix();

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

gltDrawTorus(0.35, 0.15, 40, 20);

glPopMatrix();

glPopMatrix();

// Переключение буферов

glutSwapBuffers();

}

// Реагирует на клавиши со стрелками, двигая систему отсчета камеры

void SpecialKeys(int key, int x, int y)

{

if (key == GLUT\_KEY\_UP)

gltMoveFrameForward(&frameCamera, 0.1f);

if (key == GLUT\_KEY\_DOWN)

gltMoveFrameForward(&frameCamera, -0.1f);

if (key == GLUT\_KEY\_LEFT)

gltRotateFrameLocalY(&frameCamera, 0.1);

if (key == GLUT\_KEY\_RIGHT)

gltRotateFrameLocalY(&frameCamera, -0.1);

// Обновление окна

glutPostRedisplay();

}

///////////////////////////////////////////////////////

////

//Вызывается библиотекой GLUT в холостом состоянии (окно не меняет

// размера и не перемещается)

void TimerFunction(int value)

{

// Перерисовывает сцену с новыми координатами

glutPostRedisplay();

glutTimerFunc(3, TimerFunction, 1);

}

void ChangeSize(int w, int h)

{

GLfloat fAspect;

// Предотвращает деление на нуль, когда окно слишком маленькое

// (нельзя сделать окно нулевой ширины).

if (h == 0)

h = 1;

glViewport(0, 0, w, h);

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

// Система координат обновляется перед модификацией

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Устанавливается объѐм сечения

gluPerspective(35.0f, fAspect, 1.0f, 50.0f);

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 SphereWorld Demo");

glutReshapeFunc(ChangeSize);

glutDisplayFunc(RenderScene);

glutSpecialFunc(SpecialKeys);

SetupRC();

glutTimerFunc(33, TimerFunction, 1);

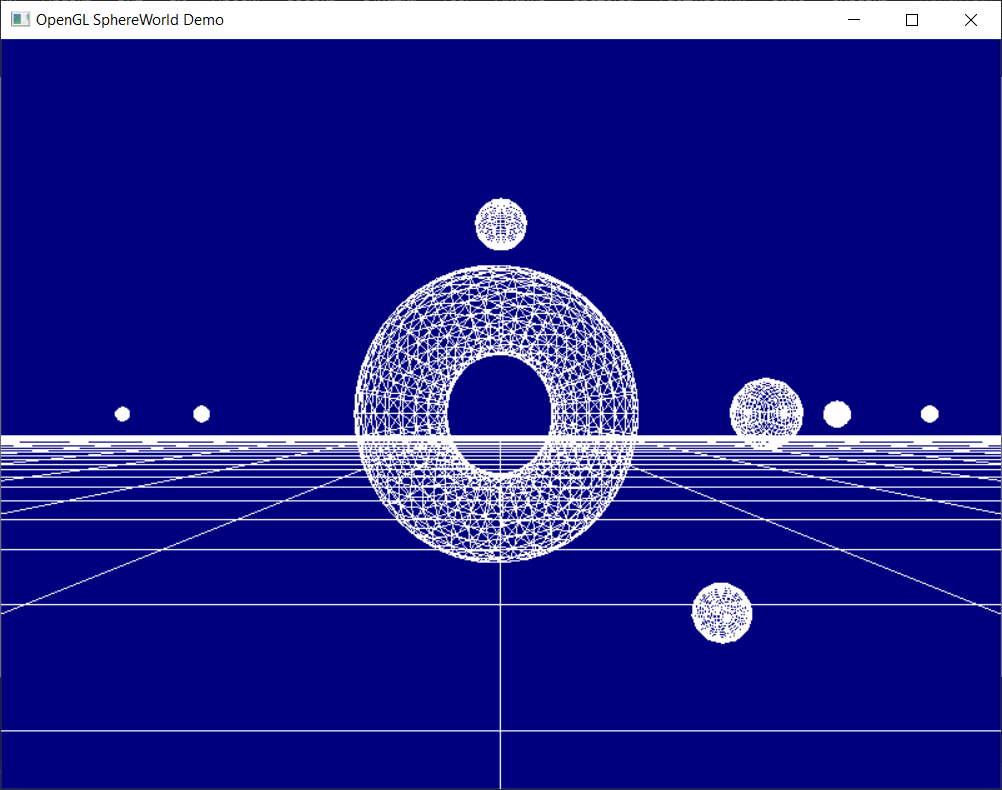
glutMainLoop();

return 0;

}

}

**Результат листинга 7:**



**Рисунок 6.** Результат листинга 7

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки реализации преобразований наблюдения модели и проектирования, ортографической и перспективной проекций, работы с матрицами преобразований, актерами.