|  |  |
| --- | --- |
| **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 №4**

**«Освещение 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 освещением средствами OpenGL, а также созданию настройки свойств материалов объектов и применения эффектов тени и тумана для большей реалистичности изображений.

**Задачи:**

* знать, что такое затенение и особенности его применения к объектам различной геометрии;
* уметь работать с различными типами освещения и согласовывать свет с характеристиками материала;
* уметь устанавливать характеристики источников освещения, а также создавать блики и прожектора;
* понимать принципы формирования тени объекта. Уметь создавать тень на различных поверхностях;
* уметь реализовывать эффект отражения;
* понимать принципы формирования тумана и знать основные характеристики;
* понимать принципы сглаживания и уметь задавать необходимые в контексте каждой задачи параметры для выполнения эффекта сглаживания.

**Вариант 7**

**Задание:**

1. Выполнить затенение **параллелепипеда**.
2. Продемонстрировать на 6 примерах различные варианты работы функций glLightModelfv и glColorMaterial. Модель – **велосипед детский трехколесный**.
3. На основе объекта из задания 2 продемонстрировать установку **трех источников света разного цвета** (Примечание: для тех, кто претендует на оценку «отлично», предусмотреть включение-выключение источника света через меню).
4. На основе объекта из задания 2 продемонстрировать создание бликов на объекте.
5. На основе объекта из задания 2 создать **3 прожектора с углом между ними 60 градусов: по 30 градусов**.
6. На основе объекта из задания 2 создать **два источника света. Отобразить тень объекта на исходную плоскость: из меню выбирать включение и выключение каждого источника света**.
7. На основе листинга 6 (7) согласно варианту из лабораторной работы №3 продемонстрировать эффект отражения.
8. На основе листинга 6 и 7 согласно варианту из лабораторной работы №3 создать туман с разными параметрами. Должен посредством меню меняться цвет, режим и уравнение тумана.
9. На основе листингов 2, 3, 4 и 5 лабораторной работы №2 продемонстрировать работу механизма сглаживания.

**Листинг 1:**

#include "glew.h"

#include "glut.h"

#include <math.h>

#include "ObjLoader.h"

namespace Task\_4\_1

{

ObjLoader\* loader;

// Величина поворота

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

typedef GLfloat GLTVector3[3];

// Значение скалярного вектора

void gltScaleVector(GLTVector3 vVector, const GLfloat

fScale)

{

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

vVector[2] \*= fScale;

}

// Получает квадрат длины вектора

GLfloat gltGetVectorLengthSqrd(const GLTVector3

vVector)

{

return (vVector[0] \* vVector[0]) +

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

}

// Получает длину вектора

GLfloat gltGetVectorLength(const GLTVector3 vVector)

{

return

(GLfloat)sqrt(gltGetVectorLengthSqrd(vVector));

}

// Нормируется вектор источника света

void gltNormalizeVector(GLTVector3 vNormal)

{

GLfloat fLength = 1.0f /

gltGetVectorLength(vNormal);

gltScaleVector(vNormal, fLength);

}

// 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];

}

// Вычесть один вектор из другого

void gltSubtractVectors(const GLTVector3 vFirst,

const GLTVector3 vSecond, GLTVector3 vResult)

{

vResult[0] = vFirst[0] - vSecond[0];

vResult[1] = vFirst[1] - vSecond[1];

vResult[2] = vFirst[2] - vSecond[2];

}

// Даны три точки на плоскости направлены против часовой стрелки, вычислить // нормаль

void gltGetNormalVector(const GLTVector3 vP1, const

GLTVector3 vP2, const GLTVector3 vP3, GLTVector3

vNormal)

{

GLTVector3 vV1, vV2;

gltSubtractVectors(vP2, vP1, vV1);

gltSubtractVectors(vP3, vP1, vV2);

gltVectorCrossProduct(vV1, vV2, vNormal);

gltNormalizeVector(vNormal);

}

// Вызывается для рисования сцены

void RenderScene(void)

{

GLTVector3 vNormal; // Память для вычисления нормали к поверхности

// Очистка окна текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT |

GL\_DEPTH\_BUFFER\_BIT);

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

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

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

// Носовой конус - прямо вниз

// Устанавливается цвет материала

glColor3ub(128, 0, 128);

loader->Draw();

// Восстановление состояния матрицы

glPopMatrix();

// Отобразить результаты

glutSwapBuffers();

}

// Эта функция выполняет необходимую инициализацию в контексте

// визуализации.

void SetupRC()

{

// Координаты

GLfloat ambientLight[] = { 0.3f, 0.3f, 0.3f,

1.0f };

GLfloat diffuseLight[] = { 0.7f, 0.7f, 0.7f,

1.0f };

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

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

// направлены вперед

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

// Активизируется освещение

glEnable(GL\_LIGHTING);

// Устанавливается и активизируется источник света 0

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

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

glEnable(GL\_LIGHT0);

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

// Свойства материалов соответствуют кодам glColor

glColorMaterial(GL\_FRONT,

GL\_AMBIENT\_AND\_DIFFUSE);

glClearColor(0.5f, 1.0f, 0.5f, 1.0f);

loader = new ObjLoader("parallelepiped.obj");

}

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

// Упарвление стрелками

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;

// Обновление окна

glutPostRedisplay();

}

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

/////

// Обновляется проекция и положение источника света

void ChangeSize(int w, int h)

{

GLfloat fAspect;

GLfloat lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f

};

// Предотвращает деление на нуль

if (h == 0)

h = 1;

// Размер поля просмотра устанавливается равным размеру окна glViewport(0, 0, w, h);

// Обновляется система координат

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

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

gluPerspective(45.0f, fAspect, 1.0f, 225.0f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

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

glTranslatef(0.0f, 0.0f, -150.0f);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB |

GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("Parallelepiped");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

glutMainLoop();

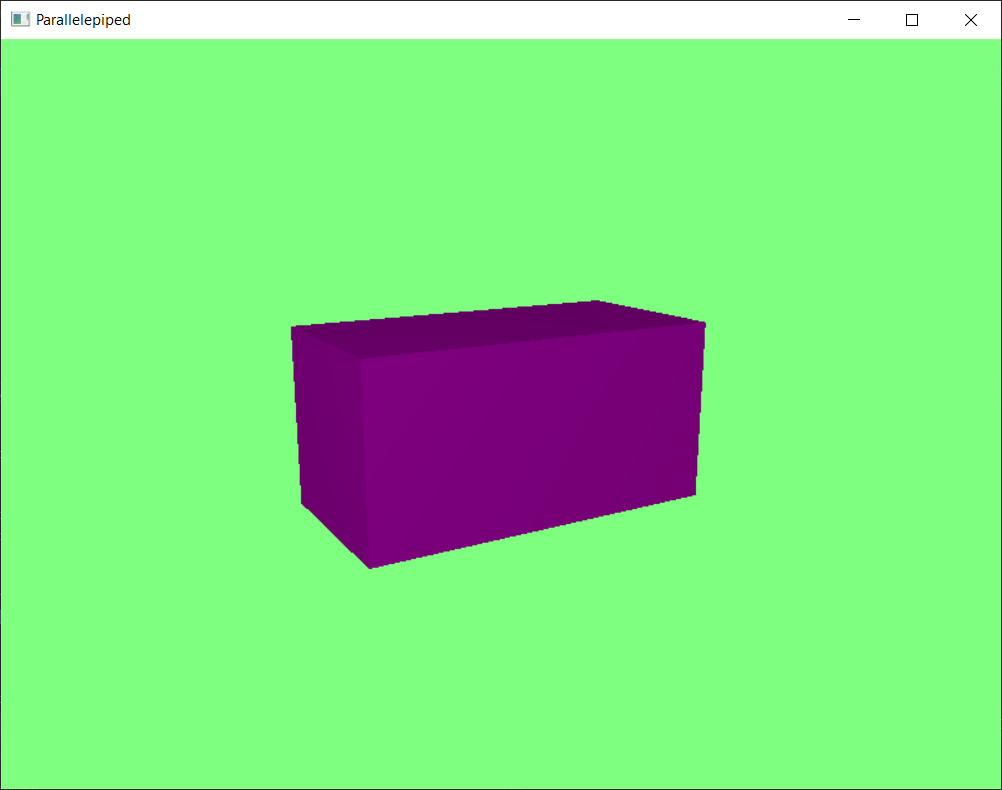
return 0;

}

}

int main(int argc, char\* argv[]) { return Task\_4\_1::main(argc, argv); }

**Результат листинга 1:**



**Рисунок 1.** Результат листинга 1

**Листинг 2:**

#include "glew.h"

#include "glut.h"

#include <math.h>

#include "ObjLoader.h"

namespace Task\_4\_2

{

ObjLoader\* loader;

bool diffuseIsPurple = true;

// Величина поворота

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

typedef GLfloat GLTVector3[3];

// Значение скалярного вектора

void gltScaleVector(GLTVector3 vVector, const GLfloat

fScale)

{

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

vVector[2] \*= fScale;

}

// Получает квадрат длины вектора

GLfloat gltGetVectorLengthSqrd(const GLTVector3

vVector)

{

return (vVector[0] \* vVector[0]) +

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

}

// Получает длину вектора

GLfloat gltGetVectorLength(const GLTVector3 vVector)

{

return

(GLfloat)sqrt(gltGetVectorLengthSqrd(vVector));

}

// Нормируется вектор источника света

void gltNormalizeVector(GLTVector3 vNormal)

{

GLfloat fLength = 1.0f /

gltGetVectorLength(vNormal);

gltScaleVector(vNormal, fLength);

}

// 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];

}

// Вычесть один вектор из другого

void gltSubtractVectors(const GLTVector3 vFirst,

const GLTVector3 vSecond, GLTVector3 vResult)

{

vResult[0] = vFirst[0] - vSecond[0];

vResult[1] = vFirst[1] - vSecond[1];

vResult[2] = vFirst[2] - vSecond[2];

}

// Даны три точки на плоскости направлены против часовой стрелки, вычислить // нормаль

void gltGetNormalVector(const GLTVector3 vP1, const

GLTVector3 vP2, const GLTVector3 vP3, GLTVector3

vNormal)

{

GLTVector3 vV1, vV2;

gltSubtractVectors(vP2, vP1, vV1);

gltSubtractVectors(vP3, vP1, vV2);

gltVectorCrossProduct(vV1, vV2, vNormal);

gltNormalizeVector(vNormal);

}

// Вызывается для рисования сцены

void RenderScene(void)

{

GLTVector3 vNormal; // Память для вычисления нормали к поверхности

// Очистка окна текущим цветом очистки

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

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

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

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

GLfloat lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };

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

loader->Draw();

// Восстановление состояния матрицы

glPopMatrix();

// Отобразить результаты

glutSwapBuffers();

}

// Эта функция выполняет необходимую инициализацию в контексте

// визуализации.

void SetupRC()

{

GLfloat diffuseLight[] = { 0.5f, 0.5f, 0.5f, 1.0f };

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

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

// направлены вперед

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

// Активизируется освещение

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

glEnable(GL\_LIGHT0);

glClearColor(0.5f, 1.0f, 0.5f, 1.0f);

glEnable(GL\_LIGHTING);

GLfloat white[] = { 1.0f, 1.0f, 1.0f, 1.0f };

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

glColor3ub(128, 0, 128);

glColorMaterial(GL\_FRONT, GL\_AMBIENT);

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

glDisable(GL\_COLOR\_MATERIAL);

glColor3ub(128, 128, 128);

glColorMaterial(GL\_FRONT, GL\_DIFFUSE);

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

loader = new ObjLoader("tricycle.obj");

}

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

// Упарвление стрелками

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;

// Обновление окна

glutPostRedisplay();

}

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

/////

// Обновляется проекция и положение источника света

void ChangeSize(int w, int h)

{

GLfloat fAspect;

// Предотвращает деление на нуль

if (h == 0)

h = 1;

// Размер поля просмотра устанавливается равным размеру окна glViewport(0, 0, w, h);

// Обновляется система координат

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

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

gluPerspective(45.0f, fAspect, 1.0f, 225.0f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0f, -30.0f, -150.0f);

}

void processMenuEvents(int option) {

GLfloat white[] = { 1.0f, 1.0f, 1.0f, 1.0f };

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

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

switch (option) {

case 1:

if (diffuseIsPurple)

glColor3ub(128, 0, 0);

else

glColor3ub(128, 128, 128);

diffuseIsPurple = !diffuseIsPurple;

break;

case 2:

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

break;

case 3:

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

break;

case 4:

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

break;

}

RenderScene();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB |

GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("Tricycle");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

glutCreateMenu(processMenuEvents);

//добавить записи в нашем меню

glutAddMenuEntry("Switch diffuse color", 1);

glutAddMenuEntry("White ambient light", 2);

glutAddMenuEntry("Red ambient light", 3);

glutAddMenuEntry("Green ambient light", 4);

// прикрепить меню правой кнопки

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

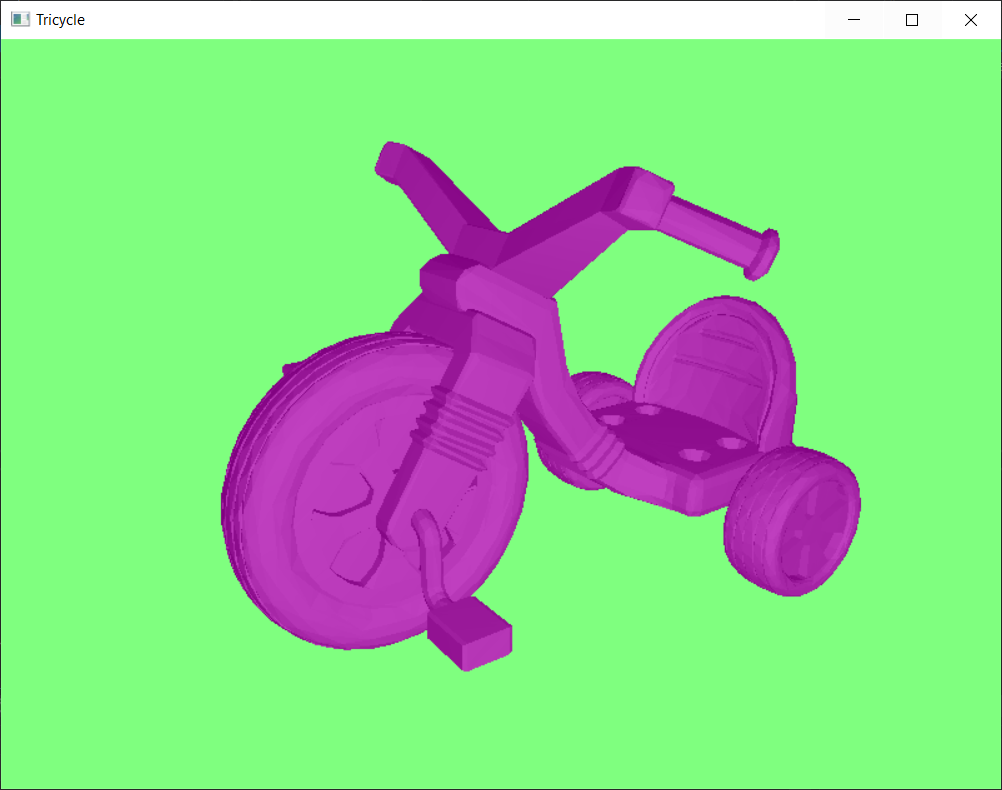
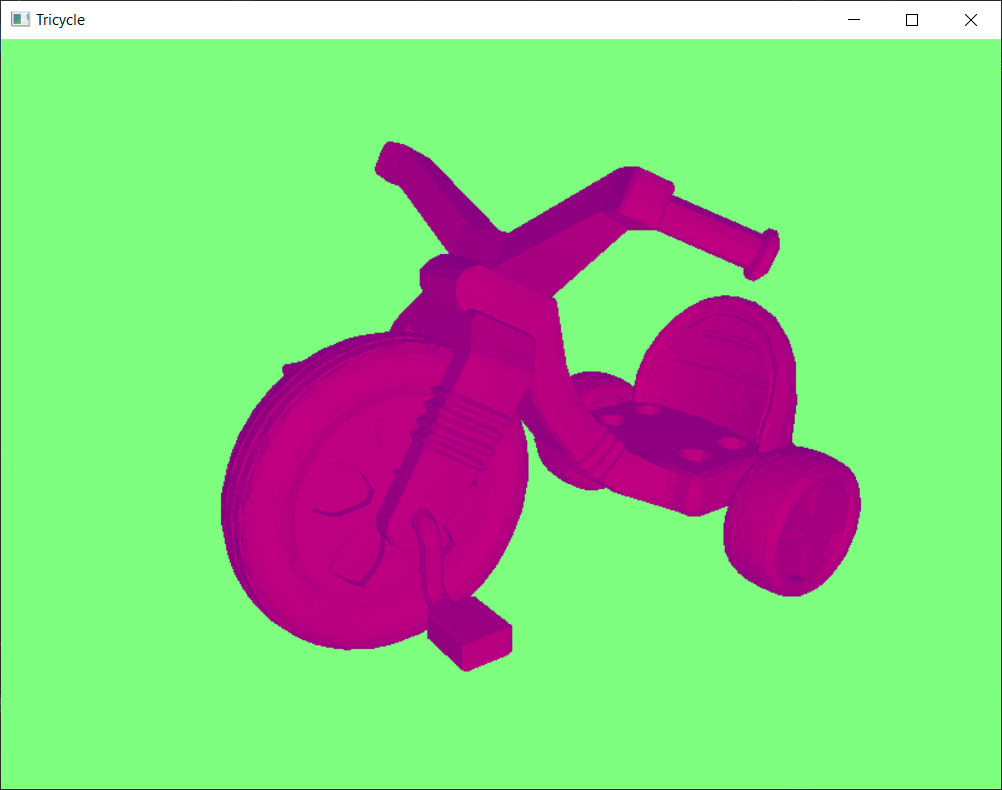
return 0;

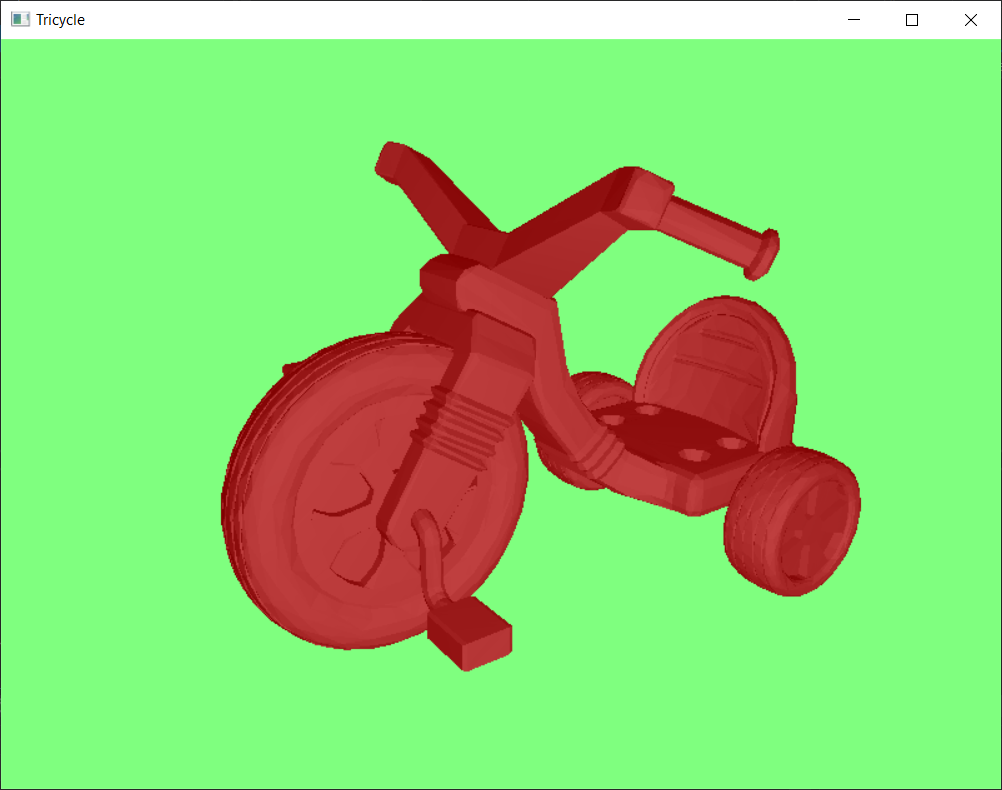
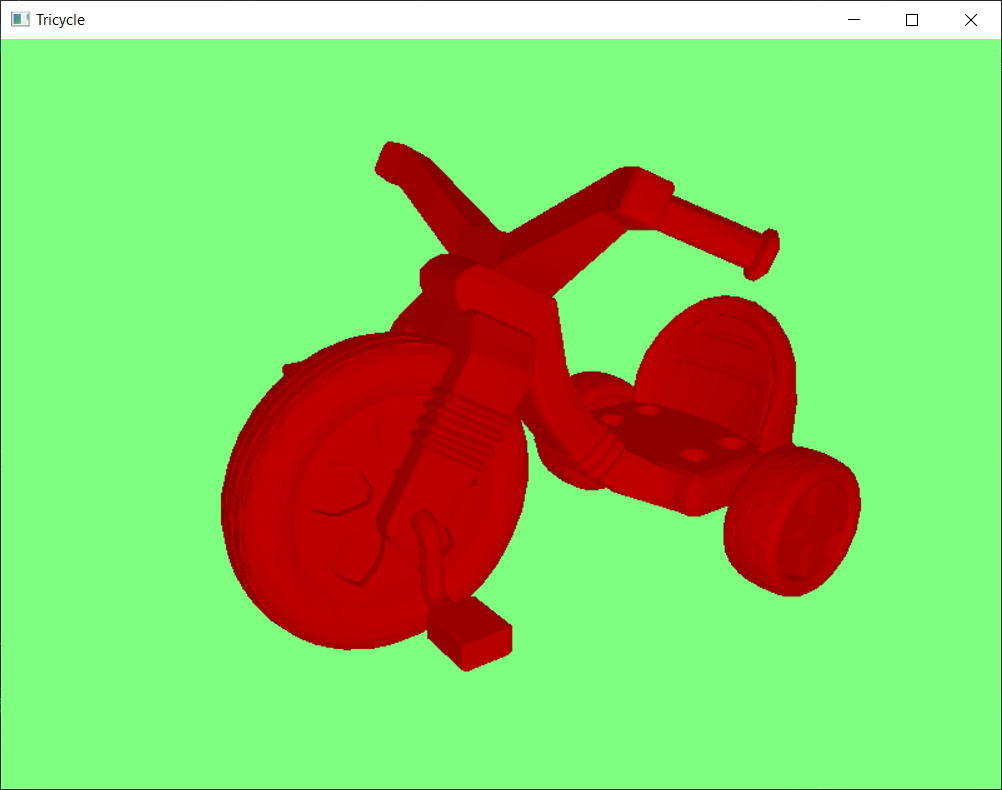
}

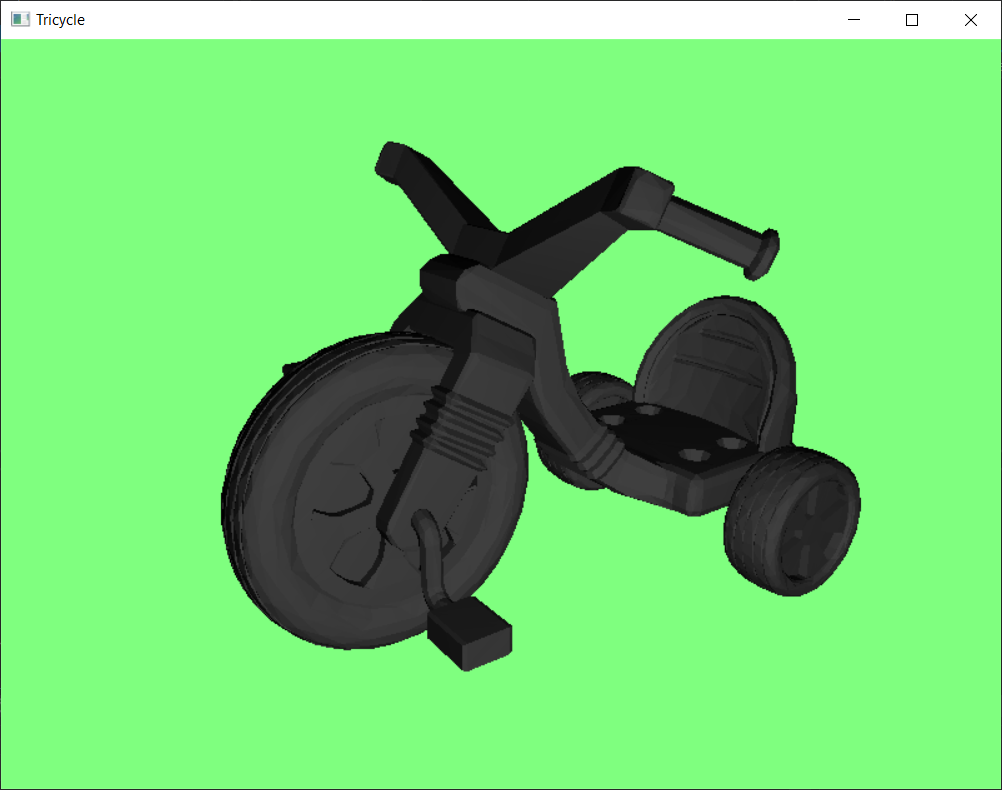
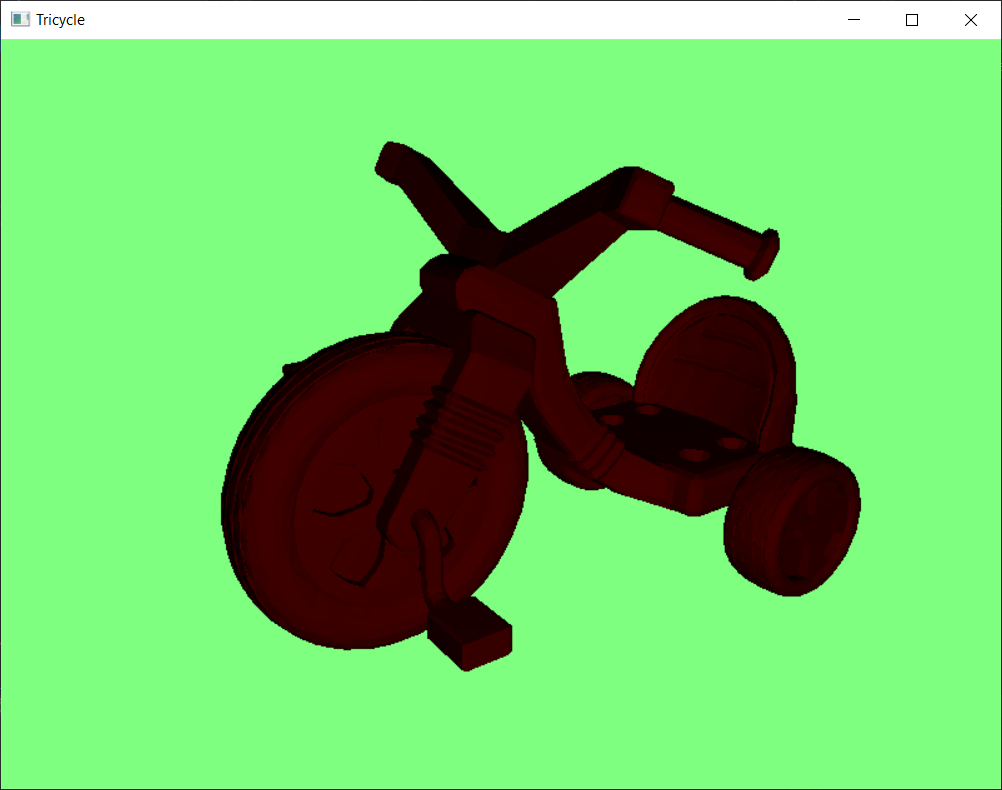
}

int main(int argc, char\* argv[]) { return Task\_4\_2::main(argc, argv); }

**Результат листинга 2:**

**Рисунок 2.** Результат листинга 2

**Листинг 3:**

#include "glew.h"

#include "glut.h"

#include <math.h>

#include "ObjLoader.h"

namespace Task\_4\_3

{

ObjLoader\* loader;

bool redIsEnabled = true;

bool greenIsEnabled = true;

bool blueIsEnabled = true;

// Величина поворота

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

typedef GLfloat GLTVector3[3];

// Значение скалярного вектора

void gltScaleVector(GLTVector3 vVector, const GLfloat

fScale)

{

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

vVector[2] \*= fScale;

}

// Получает квадрат длины вектора

GLfloat gltGetVectorLengthSqrd(const GLTVector3

vVector)

{

return (vVector[0] \* vVector[0]) +

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

}

// Получает длину вектора

GLfloat gltGetVectorLength(const GLTVector3 vVector)

{

return

(GLfloat)sqrt(gltGetVectorLengthSqrd(vVector));

}

// Нормируется вектор источника света

void gltNormalizeVector(GLTVector3 vNormal)

{

GLfloat fLength = 1.0f /

gltGetVectorLength(vNormal);

gltScaleVector(vNormal, fLength);

}

// 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];

}

// Вычесть один вектор из другого

void gltSubtractVectors(const GLTVector3 vFirst,

const GLTVector3 vSecond, GLTVector3 vResult)

{

vResult[0] = vFirst[0] - vSecond[0];

vResult[1] = vFirst[1] - vSecond[1];

vResult[2] = vFirst[2] - vSecond[2];

}

// Даны три точки на плоскости направлены против часовой стрелки, вычислить // нормаль

void gltGetNormalVector(const GLTVector3 vP1, const

GLTVector3 vP2, const GLTVector3 vP3, GLTVector3

vNormal)

{

GLTVector3 vV1, vV2;

gltSubtractVectors(vP2, vP1, vV1);

gltSubtractVectors(vP3, vP1, vV2);

gltVectorCrossProduct(vV1, vV2, vNormal);

gltNormalizeVector(vNormal);

}

// Вызывается для рисования сцены

void RenderScene(void)

{

GLTVector3 vNormal; // Память для вычисления нормали к поверхности

// Очистка окна текущим цветом очистки

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

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

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

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

GLfloat lightPos\_1[] = { 0.0f, 0.0f, 100.0f, 1.0f };

GLfloat lightPos\_2[] = { 0.0f, 0.0f, -100.0f, 1.0f };

GLfloat lightPos\_3[] = { 100.0f, 0.0f, 0.0f, 1.0f };

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPos\_1);

glLightfv(GL\_LIGHT1, GL\_POSITION, lightPos\_2);

glLightfv(GL\_LIGHT2, GL\_POSITION, lightPos\_3);

loader->Draw();

// Восстановление состояния матрицы

glPopMatrix();

// Отобразить результаты

glutSwapBuffers();

}

// Эта функция выполняет необходимую инициализацию в контексте

// визуализации.

void SetupRC()

{

GLfloat diffuseLight\_1[] = { 1.0f, 0.0f, 0.0f, 1.0f };

GLfloat diffuseLight\_2[] = { 0.0f, 1.0f, 0.0f, 1.0f };

GLfloat diffuseLight\_3[] = { 0.0f, 0.0f, 1.0f, 1.0f };

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

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

// направлены вперед

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

glEnable(GL\_LIGHTING);

// Активизируется освещение

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuseLight\_1);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, diffuseLight\_2);

glEnable(GL\_LIGHT1);

glLightfv(GL\_LIGHT2, GL\_DIFFUSE, diffuseLight\_3);

glEnable(GL\_LIGHT2);

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

GLfloat white[] = { 1.0f, 1.0f, 1.0f, 1.0f };

//glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, white);

glColor3ub(128, 128, 128);

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

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

loader = new ObjLoader("tricycle.obj");

}

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

// Упарвление стрелками

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;

// Обновление окна

RenderScene();

}

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

/////

// Обновляется проекция и положение источника света

void ChangeSize(int w, int h)

{

GLfloat fAspect;

// Предотвращает деление на нуль

if (h == 0)

h = 1;

// Размер поля просмотра устанавливается равным размеру окна glViewport(0, 0, w, h);

// Обновляется система координат

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

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

gluPerspective(45.0f, fAspect, 1.0f, 225.0f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0f, -30.0f, -150.0f);

}

void processMenuEvents(int option) {

switch (option) {

case 1:

if (redIsEnabled)

glDisable(GL\_LIGHT0);

else

glEnable(GL\_LIGHT0);

redIsEnabled = !redIsEnabled;

break;

case 2:

if (greenIsEnabled)

glDisable(GL\_LIGHT1);

else

glEnable(GL\_LIGHT1);

greenIsEnabled = !greenIsEnabled;

break;

case 3:

if (blueIsEnabled)

glDisable(GL\_LIGHT2);

else

glEnable(GL\_LIGHT2);

blueIsEnabled = !blueIsEnabled;

break;

}

RenderScene();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB |

GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("Tricycle");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

glutCreateMenu(processMenuEvents);

//добавить записи в нашем меню

glutAddMenuEntry("Switch red light", 1);

glutAddMenuEntry("Switch green light", 2);

glutAddMenuEntry("Switch blue light", 3);

// прикрепить меню правой кнопки

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

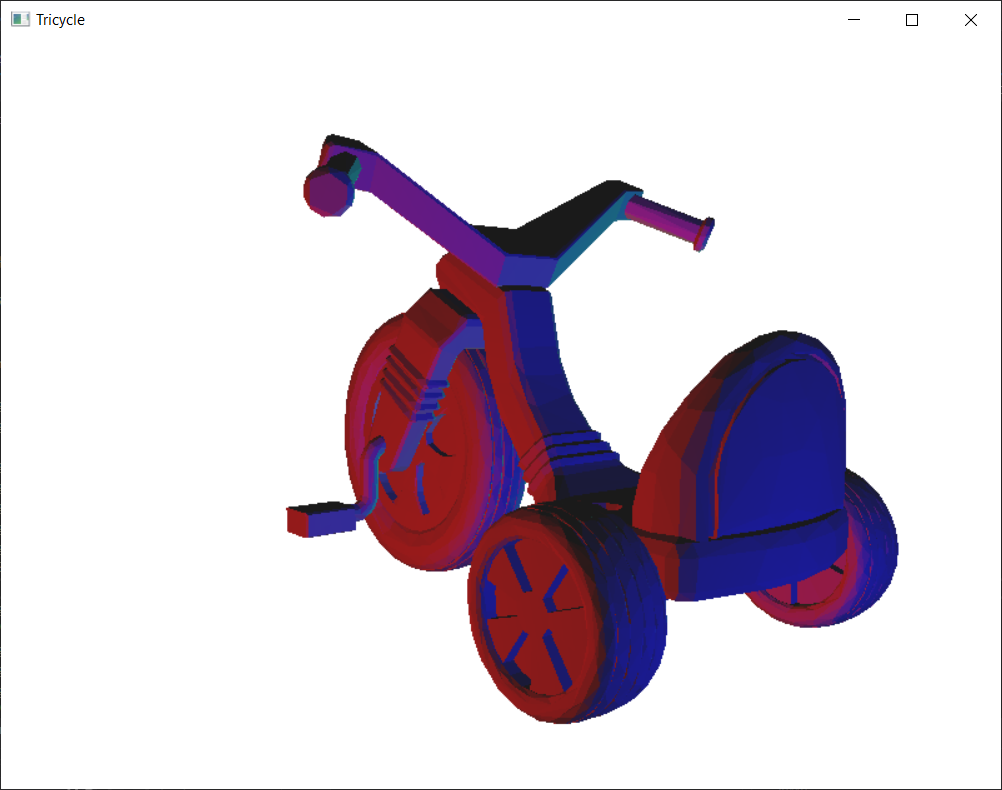
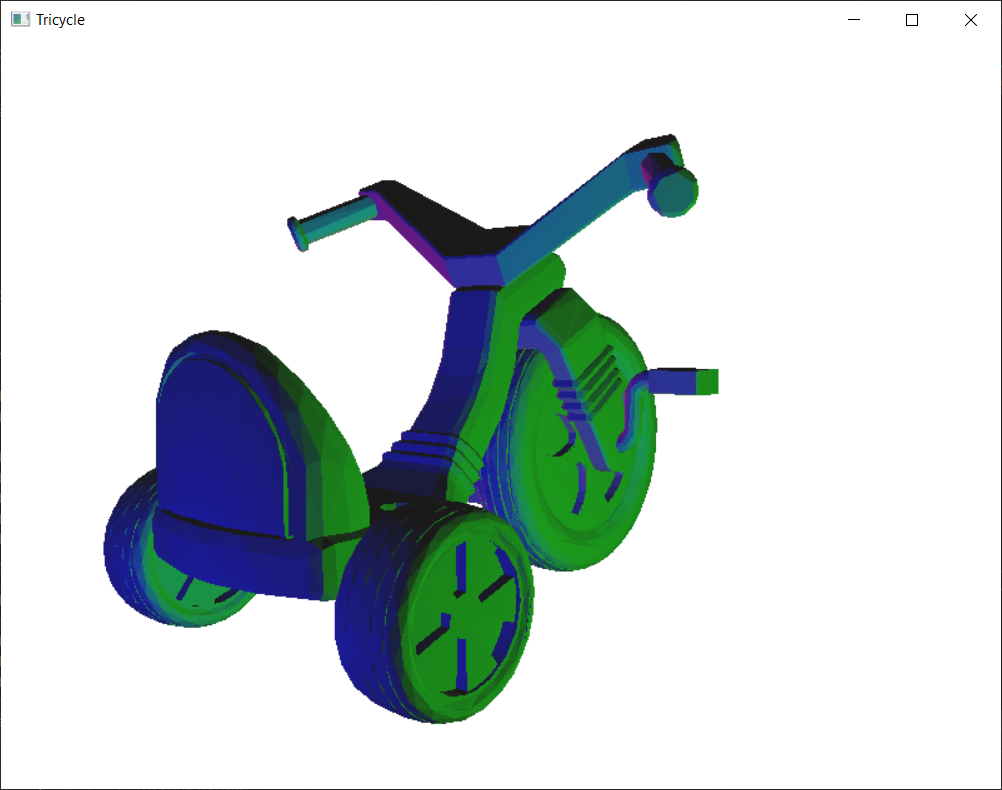
return 0;

}

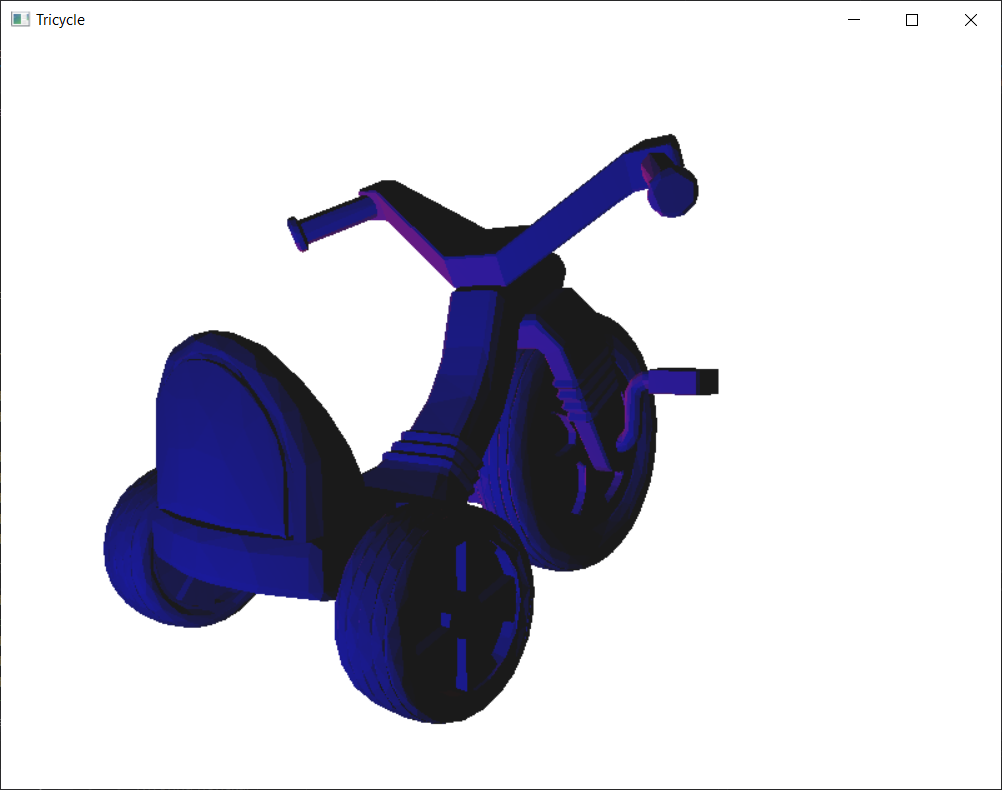
}

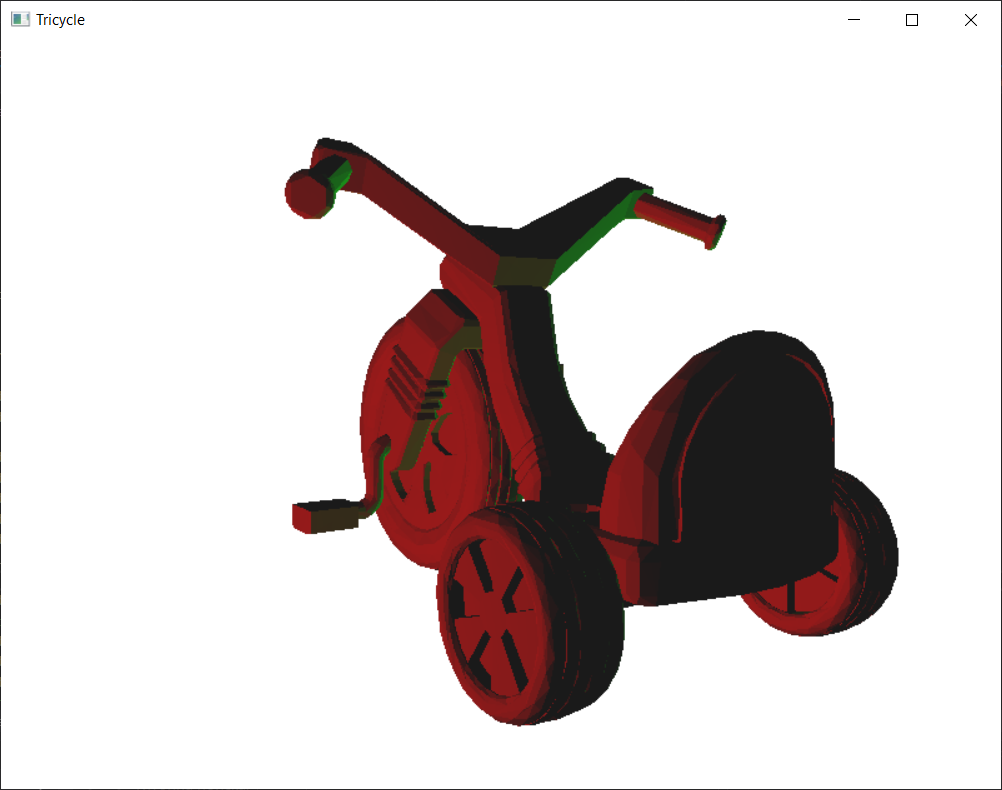
int main(int argc, char\* argv[]) { return Task\_4\_3::main(argc, argv); }

**Результат листинга 3:**

**Рисунок 3.1.** Результат листинга 3

**Рисунок 3.1.** Результат листинга 3

**Листинг 4:**

#include "glew.h"

#include "glut.h"

#include <math.h>

#include "ObjLoader.h"

namespace Task\_4\_4

{

ObjLoader\* loader;

bool diffuseIsPurple = true;

// Величина поворота

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

typedef GLfloat GLTVector3[3];

// Значение скалярного вектора

void gltScaleVector(GLTVector3 vVector, const GLfloat

fScale)

{

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

vVector[2] \*= fScale;

}

// Получает квадрат длины вектора

GLfloat gltGetVectorLengthSqrd(const GLTVector3

vVector)

{

return (vVector[0] \* vVector[0]) +

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

}

// Получает длину вектора

GLfloat gltGetVectorLength(const GLTVector3 vVector)

{

return

(GLfloat)sqrt(gltGetVectorLengthSqrd(vVector));

}

// Нормируется вектор источника света

void gltNormalizeVector(GLTVector3 vNormal)

{

GLfloat fLength = 1.0f /

gltGetVectorLength(vNormal);

gltScaleVector(vNormal, fLength);

}

// 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];

}

// Вычесть один вектор из другого

void gltSubtractVectors(const GLTVector3 vFirst,

const GLTVector3 vSecond, GLTVector3 vResult)

{

vResult[0] = vFirst[0] - vSecond[0];

vResult[1] = vFirst[1] - vSecond[1];

vResult[2] = vFirst[2] - vSecond[2];

}

// Даны три точки на плоскости направлены против часовой стрелки, вычислить // нормаль

void gltGetNormalVector(const GLTVector3 vP1, const

GLTVector3 vP2, const GLTVector3 vP3, GLTVector3

vNormal)

{

GLTVector3 vV1, vV2;

gltSubtractVectors(vP2, vP1, vV1);

gltSubtractVectors(vP3, vP1, vV2);

gltVectorCrossProduct(vV1, vV2, vNormal);

gltNormalizeVector(vNormal);

}

// Вызывается для рисования сцены

void RenderScene(void)

{

GLTVector3 vNormal; // Память для вычисления нормали к поверхности

// Очистка окна текущим цветом очистки

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

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

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

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

GLfloat lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };

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

loader->Draw();

// Восстановление состояния матрицы

glPopMatrix();

// Отобразить результаты

glutSwapBuffers();

}

// Эта функция выполняет необходимую инициализацию в контексте

// визуализации.

void SetupRC()

{

GLfloat diffuseLight[] = { 0.5f, 0.5f, 0.5f, 1.0f };

GLfloat specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };

GLfloat specref[] = { 1.0f, 1.0f, 1.0f, 1.0f };

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

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

// направлены вперед

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

// Активизируется освещение

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

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

glEnable(GL\_LIGHT0);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, specref);

glMateriali(GL\_FRONT, GL\_SHININESS, 128);

glClearColor(0.5f, 1.0f, 0.5f, 1.0f);

glEnable(GL\_LIGHTING);

GLfloat white[] = { 1.0f, 1.0f, 1.0f, 1.0f };

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

glColor3ub(128, 0, 128);

glColorMaterial(GL\_FRONT, GL\_AMBIENT);

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

glDisable(GL\_COLOR\_MATERIAL);

glColor3ub(128, 128, 128);

glColorMaterial(GL\_FRONT, GL\_DIFFUSE);

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

loader = new ObjLoader("tricycle.obj");

}

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

// Упарвление стрелками

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;

// Обновление окна

glutPostRedisplay();

}

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

/////

// Обновляется проекция и положение источника света

void ChangeSize(int w, int h)

{

GLfloat fAspect;

// Предотвращает деление на нуль

if (h == 0)

h = 1;

// Размер поля просмотра устанавливается равным размеру окна glViewport(0, 0, w, h);

// Обновляется система координат

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

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

gluPerspective(45.0f, fAspect, 1.0f, 225.0f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0f, -30.0f, -150.0f);

}

void processMenuEvents(int option) {

GLfloat white[] = { 1.0f, 1.0f, 1.0f, 1.0f };

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

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

switch (option) {

case 1:

if (diffuseIsPurple)

glColor3ub(128, 0, 0);

else

glColor3ub(128, 128, 128);

diffuseIsPurple = !diffuseIsPurple;

break;

case 2:

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

break;

case 3:

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

break;

case 4:

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

break;

}

RenderScene();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB |

GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("Tricycle");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

glutCreateMenu(processMenuEvents);

//добавить записи в нашем меню

glutAddMenuEntry("Switch diffuse color", 1);

glutAddMenuEntry("White ambient light", 2);

glutAddMenuEntry("Red ambient light", 3);

glutAddMenuEntry("Green ambient light", 4);

// прикрепить меню правой кнопки

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

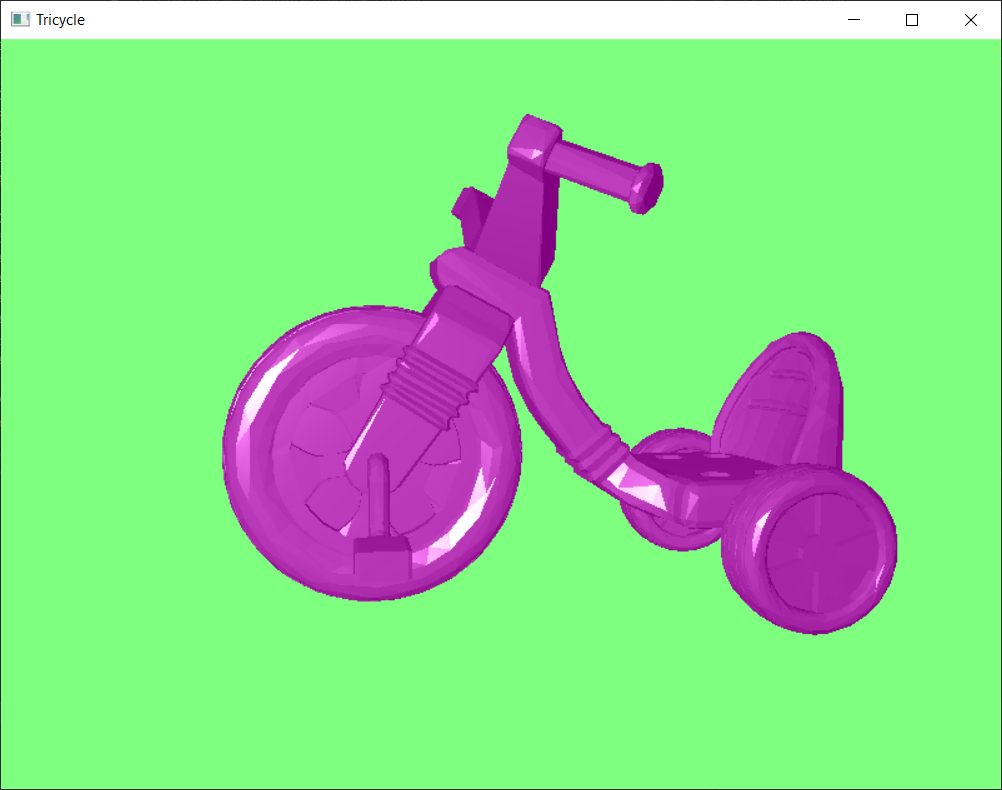
return 0;

}

}

int main(int argc, char\* argv[]) { return Task\_4\_4::main(argc, argv); }

**Результат листинга 4:**



**Рисунок 4.** Результат листинга 4

**Листинг 5:**

#include "glew.h"

#include "glut.h"

#include <math.h>

#include "ObjLoader.h"

namespace Task\_4\_5

{

ObjLoader\* loader;

bool diffuseIsPurple = true;

// Величина поворота

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

typedef GLfloat GLTVector3[3];

// Значение скалярного вектора

void gltScaleVector(GLTVector3 vVector, const GLfloat

fScale)

{

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

vVector[2] \*= fScale;

}

// Получает квадрат длины вектора

GLfloat gltGetVectorLengthSqrd(const GLTVector3

vVector)

{

return (vVector[0] \* vVector[0]) +

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

}

// Получает длину вектора

GLfloat gltGetVectorLength(const GLTVector3 vVector)

{

return

(GLfloat)sqrt(gltGetVectorLengthSqrd(vVector));

}

// Нормируется вектор источника света

void gltNormalizeVector(GLTVector3 vNormal)

{

GLfloat fLength = 1.0f /

gltGetVectorLength(vNormal);

gltScaleVector(vNormal, fLength);

}

// 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];

}

// Вычесть один вектор из другого

void gltSubtractVectors(const GLTVector3 vFirst,

const GLTVector3 vSecond, GLTVector3 vResult)

{

vResult[0] = vFirst[0] - vSecond[0];

vResult[1] = vFirst[1] - vSecond[1];

vResult[2] = vFirst[2] - vSecond[2];

}

// Даны три точки на плоскости направлены против часовой стрелки, вычислить // нормаль

void gltGetNormalVector(const GLTVector3 vP1, const

GLTVector3 vP2, const GLTVector3 vP3, GLTVector3

vNormal)

{

GLTVector3 vV1, vV2;

gltSubtractVectors(vP2, vP1, vV1);

gltSubtractVectors(vP3, vP1, vV2);

gltVectorCrossProduct(vV1, vV2, vNormal);

gltNormalizeVector(vNormal);

}

// Вызывается для рисования сцены

void RenderScene(void)

{

GLTVector3 vNormal; // Память для вычисления нормали к поверхности

// Очистка окна текущим цветом очистки

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

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

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

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

GLfloat lightPos[] = { -10.0f, 0.0f, 50.0f, 1.0f };

GLfloat spotDir\_0[] = { 0.0f, 0.0f, -1.0f };

GLfloat spotDir\_1[] = { sqrt(3)/2, 0.0f, -0.5f };

GLfloat spotDir\_2[] = { sqrt(3)/6, sqrt(6)/3, -0.5f };

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

glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, spotDir\_0);

glLightfv(GL\_LIGHT1, GL\_POSITION, lightPos);

glLightfv(GL\_LIGHT1, GL\_SPOT\_DIRECTION, spotDir\_1);

glLightfv(GL\_LIGHT2, GL\_POSITION, lightPos);

glLightfv(GL\_LIGHT2, GL\_SPOT\_DIRECTION, spotDir\_2);

loader->Draw();

// Восстановление состояния матрицы

glPopMatrix();

// Отобразить результаты

glutSwapBuffers();

}

// Эта функция выполняет необходимую инициализацию в контексте

// визуализации.

void SetupRC()

{

GLfloat diffuseLight\_0[] = { 0.5f, 0.0f, 0.0f, 1.0f };

GLfloat diffuseLight\_1[] = { 0.0f, 0.5f, 0.0f, 1.0f };

GLfloat diffuseLight\_2[] = { 0.0f, 0.0f, 0.5f, 1.0f };

GLfloat specular\_0[] = { 1.0f, 0.0f, 0.0f, 1.0f };

GLfloat specular\_1[] = { 0.0f, 1.0f, 0.0f, 1.0f };

GLfloat specular\_2[] = { 0.0f, 0.0f, 1.0f, 1.0f };

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

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

// направлены вперед

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

// Активизируется освещение

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuseLight\_0);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, specular\_0);

glLightf(GL\_LIGHT0, GL\_SPOT\_CUTOFF, 30.0f);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, diffuseLight\_1);

glLightfv(GL\_LIGHT1, GL\_SPECULAR, specular\_1);

glLightf(GL\_LIGHT1, GL\_SPOT\_CUTOFF, 30.0f);

glEnable(GL\_LIGHT1);

glLightfv(GL\_LIGHT2, GL\_DIFFUSE, diffuseLight\_2);

glLightfv(GL\_LIGHT2, GL\_SPECULAR, specular\_2);

glLightf(GL\_LIGHT2, GL\_SPOT\_CUTOFF, 30.0f);

glEnable(GL\_LIGHT2);

glClearColor(0.5f, 1.0f, 0.5f, 1.0f);

glEnable(GL\_LIGHTING);

GLfloat white[] = { 1.0f, 1.0f, 1.0f, 1.0f };

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

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

glMaterialfv(GL\_FRONT, GL\_SPECULAR, specref);

glMateriali(GL\_FRONT, GL\_SHININESS, 128);

glColor3ub(128, 0, 128);

glColorMaterial(GL\_FRONT, GL\_AMBIENT);

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

glDisable(GL\_COLOR\_MATERIAL);

glColor3ub(128, 128, 128);

glColorMaterial(GL\_FRONT, GL\_DIFFUSE);

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

loader = new ObjLoader("tricycle.obj");

}

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

// Упарвление стрелками

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;

// Обновление окна

glutPostRedisplay();

}

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

/////

// Обновляется проекция и положение источника света

void ChangeSize(int w, int h)

{

GLfloat fAspect;

// Предотвращает деление на нуль

if (h == 0)

h = 1;

// Размер поля просмотра устанавливается равным размеру окна glViewport(0, 0, w, h);

// Обновляется система координат

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

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

gluPerspective(45.0f, fAspect, 1.0f, 225.0f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0f, -30.0f, -150.0f);

}

void processMenuEvents(int option) {

GLfloat white[] = { 1.0f, 1.0f, 1.0f, 1.0f };

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

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

switch (option) {

case 1:

if (diffuseIsPurple)

glColor3ub(128, 0, 0);

else

glColor3ub(128, 128, 128);

diffuseIsPurple = !diffuseIsPurple;

break;

case 2:

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

break;

case 3:

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

break;

case 4:

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

break;

}

RenderScene();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB |

GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("Tricycle");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

glutCreateMenu(processMenuEvents);

//добавить записи в нашем меню

glutAddMenuEntry("Switch diffuse color", 1);

glutAddMenuEntry("White ambient light", 2);

glutAddMenuEntry("Red ambient light", 3);

glutAddMenuEntry("Green ambient light", 4);

// прикрепить меню правой кнопки

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

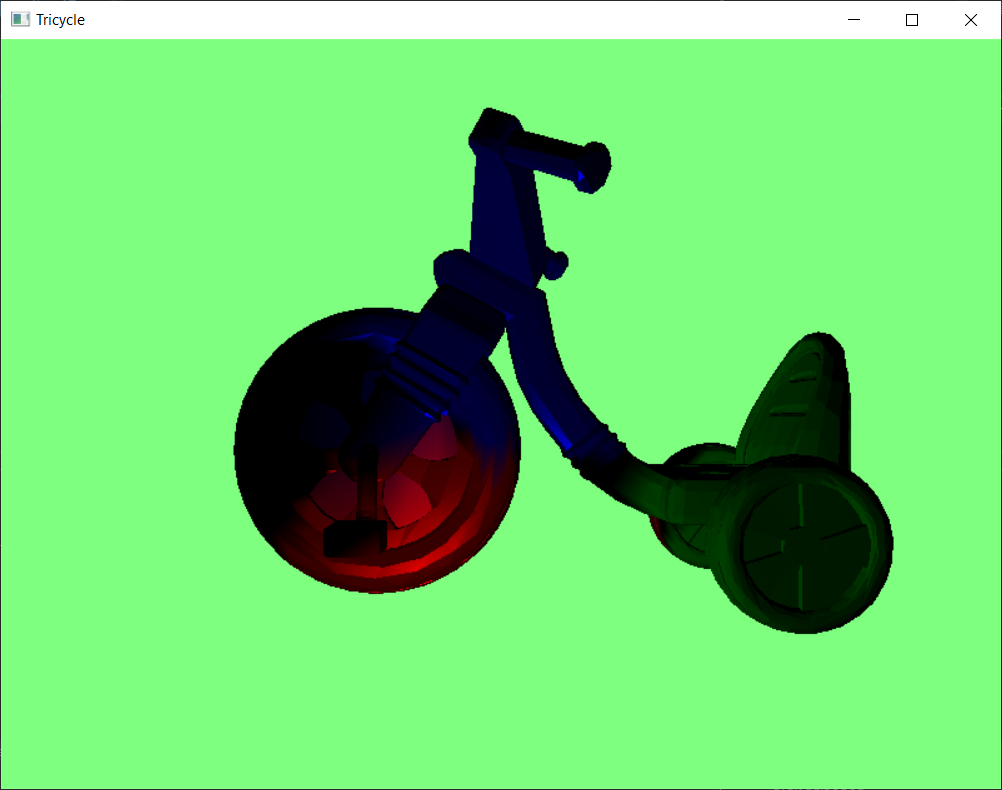
return 0;

}

}

int main(int argc, char\* argv[]) { return Task\_4\_5::main(argc, argv); }

**Результат листинга 5:**



**Рисунок 5.** Результат листинга 5

**Листинг 6:**

#include "glew.h"

#include "glut.h"

#include <math.h>

#include "ObjLoader.h"

namespace Task\_4\_6

{

typedef GLfloat GLTVector3[3];

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

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

ObjLoader\* loader;

bool light\_0IsEnabled = true;

bool light\_1IsEnabled = true;

// Величина поворота

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

GLTMatrix shadowMat\_0;

GLTMatrix shadowMat\_1;

GLfloat lightPos\_0[] = { -50.f, 150.0f, 100.0f, 1.0f };

GLfloat lightPos\_1[] = { 50.f, 150.0f, 100.0f, 1.0f };

// Значение скалярного вектора

void gltScaleVector(GLTVector3 vVector, const GLfloat

fScale)

{

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

vVector[2] \*= fScale;

}

// Получает квадрат длины вектора

GLfloat gltGetVectorLengthSqrd(const GLTVector3

vVector)

{

return (vVector[0] \* vVector[0]) +

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

}

// Получает длину вектора

GLfloat gltGetVectorLength(const GLTVector3 vVector)

{

return

(GLfloat)sqrt(gltGetVectorLengthSqrd(vVector));

}

// Нормируется вектор источника света

void gltNormalizeVector(GLTVector3 vNormal)

{

GLfloat fLength = 1.0f /

gltGetVectorLength(vNormal);

gltScaleVector(vNormal, fLength);

}

// 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];

}

// Вычесть один вектор из другого

void gltSubtractVectors(const GLTVector3 vFirst,

const GLTVector3 vSecond, GLTVector3 vResult)

{

vResult[0] = vFirst[0] - vSecond[0];

vResult[1] = vFirst[1] - vSecond[1];

vResult[2] = vFirst[2] - vSecond[2];

}

// Даны три точки на плоскости направлены против часовой стрелки, вычислить // нормаль

void gltGetNormalVector(const GLTVector3 vP1, const

GLTVector3 vP2, const GLTVector3 vP3, GLTVector3

vNormal)

{

GLTVector3 vV1, vV2;

gltSubtractVectors(vP2, vP1, vV1);

gltSubtractVectors(vP3, vP1, vV2);

gltVectorCrossProduct(vV1, vV2, vNormal);

gltNormalizeVector(vNormal);

}

// Вызывается для рисования сцены

void RenderScene(void)

{

GLTVector3 vNormal; // Память для вычисления нормали к поверхности

// Очистка окна текущим цветом очистки

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

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

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

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

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPos\_0);

glLightfv(GL\_LIGHT1, GL\_POSITION, lightPos\_1);

glBegin(GL\_QUADS);

glColor3ub(128, 128, 0);

glVertex3f(200.0f, 0.0f, -200.0f);

glVertex3f(-200.0f, 0.0f, -200.0f);

glColor3ub(0, 255, 0);

glVertex3f(-200.0f, 0.0f, 200.0f);

glVertex3f(200.0f, 0.0f, 200.0f);

glEnd();

// Восстановление состояния матрицы

glPopMatrix();

if (light\_0IsEnabled) {

glDisable(GL\_DEPTH\_TEST);

glDisable(GL\_LIGHTING);

glPushMatrix();

// Текущая матрица множится на матрицу проекции тени

// Самолетик поворачивается в новом плоском пространстве

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

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

glMultMatrixf((GLfloat\*)shadowMat\_0);

glColor3ub(16, 16, 16);

loader->Draw();

// Восстанавливается нормальная проекция

glPopMatrix();

// Восстанавливаются переменные состояния освещения

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHTING);

}

if (light\_1IsEnabled) {

glDisable(GL\_DEPTH\_TEST);

glDisable(GL\_LIGHTING);

glPushMatrix();

// Текущая матрица множится на матрицу проекции тени

// Самолетик поворачивается в новом плоском пространстве

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

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

glMultMatrixf((GLfloat\*)shadowMat\_1);

glColor3ub(16, 16, 16);

loader->Draw();

// Восстанавливается нормальная проекция

glPopMatrix();

// Восстанавливаются переменные состояния освещения

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHTING);

}

glPushMatrix();

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

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

glColor3ub(128, 0, 128);

loader->Draw();

glPopMatrix();

// Отобразить результаты

glutSwapBuffers();

}

// Полученные три коэффициента уравнения плоскости дают три точки на поверхности

void gltGetPlaneEquation(GLTVector3 vPoint1, GLTVector3 vPoint2, GLTVector3 vPoint3, GLTVector3 vPlane)

{

// Получение нормали из трех точек. Нормаль – первые три коэффициента

// уравнения плоскости

gltGetNormalVector(vPoint1, vPoint2, vPoint3, vPlane);

// Итоговый коэффициент находится обратной под\_становкой

vPlane[3] = -(vPlane[0] \* vPoint3[0] + vPlane[1] \* vPoint3[1] + vPlane[2] \* vPoint3[2]);

}

//По коэффициентам уравнения плоскости и поло\_жению источника света

// создается матрица отбрасывания тени. Полу\_ченное

// значение сохраняется в destMat

void gltMakeShadowMatrix(GLTVector3 vPoints[3], GLTVector4 vLightPos, GLTMatrix destMat)

{

GLTVector4 vPlaneEquation;

GLfloat dot;

gltGetPlaneEquation(vPoints[0], vPoints[1], vPoints[2], vPlaneEquation);

// Скалярное произведение направляющего вектора

// плоскости и положения источника света

dot = vPlaneEquation[0] \* vLightPos[0] +

vPlaneEquation[1] \* vLightPos[1] +

vPlaneEquation[2] \* vLightPos[2] +

vPlaneEquation[3] \* vLightPos[3];

// Выполняется проектирование

// Первый столбец

destMat[0] = dot - vLightPos[0] \* vPlaneEquation[0];

destMat[4] = 0.0f - vLightPos[0] \* vPlaneEquation[1];

destMat[8] = 0.0f - vLightPos[0] \* vPlaneEquation[2];

destMat[12] = 0.0f - vLightPos[0] \* vPlaneEquation[3];

// Второй столбец

destMat[1] = 0.0f - vLightPos[1] \* vPlaneEquation[0];

destMat[5] = dot - vLightPos[1] \* vPlaneEquation[1];

destMat[9] = 0.0f - vLightPos[1] \* vPlaneEquation[2];

destMat[13] = 0.0f - vLightPos[1] \* vPlaneEquation[3];

// Третий столбец

destMat[2] = 0.0f - vLightPos[2] \* vPlaneEquation[0];

destMat[6] = 0.0f - vLightPos[2] \* vPlaneEquation[1];

destMat[10] = dot - vLightPos[2] \* vPlaneEquation[2];

destMat[14] = 0.0f - vLightPos[2] \* vPlaneEquation[3];

// Четвертый столбец

destMat[3] = 0.0f - vLightPos[3] \* vPlaneEquation[0];

destMat[7] = 0.0f - vLightPos[3] \* vPlaneEquation[1];

destMat[11] = 0.0f - vLightPos[3] \* vPlaneEquation[2];

destMat[15] = dot - vLightPos[3] \* vPlaneEquation[3];

}

// Эта функция выполняет необходимую инициализацию в контексте

// визуализации.

void SetupRC()

{

GLTVector3 points[3] = {

{ 1.0f, 0.0f, -1.0f },

{ -1.0f, 0.0f, -1.0f },

{ -1.0f, 0.0f, 1.0f } };

GLfloat diffuseLight[] = { 0.5f, 0.5f, 0.5f, 1.0f };

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

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

// направлены вперед

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

// Активизируется освещение

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

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, diffuseLight);

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHT1);

glClearColor(0.5f, 1.0f, 0.5f, 1.0f);

glEnable(GL\_LIGHTING);

GLfloat white[] = { 1.0f, 1.0f, 1.0f, 1.0f };

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

glColor3ub(128, 0, 128);

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

// Активизирует согласование цветов

glEnable(GL\_COLOR\_MATERIAL);

gltMakeShadowMatrix(points, lightPos\_0, shadowMat\_0);

gltMakeShadowMatrix(points, lightPos\_1, shadowMat\_1);

loader = new ObjLoader("tricycle.obj");

}

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

// Упарвление стрелками

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;

// Обновление окна

RenderScene();

}

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

/////

// Обновляется проекция и положение источника света

void ChangeSize(int w, int h)

{

GLfloat fAspect;

// Предотвращает деление на нуль

if (h == 0)

h = 1;

// Размер поля просмотра устанавливается равным размеру окна glViewport(0, 0, w, h);

// Обновляется система координат

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

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

gluPerspective(45.0f, fAspect, 1.0f, 500.0f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0.0f, -30.0f, -150.0f);

}

void processMenuEvents(int option) {

switch (option) {

case 1:

if (light\_0IsEnabled)

glDisable(GL\_LIGHT0);

else

glEnable(GL\_LIGHT0);

light\_0IsEnabled = !light\_0IsEnabled;

break;

case 2:

if (light\_1IsEnabled)

glDisable(GL\_LIGHT1);

else

glEnable(GL\_LIGHT1);

light\_1IsEnabled = !light\_1IsEnabled;

break;

}

RenderScene();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB |

GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("Tricycle");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

glutCreateMenu(processMenuEvents);

//добавить записи в нашем меню

glutAddMenuEntry("Switch light 0", 1);

glutAddMenuEntry("Switch light 1", 2);

// прикрепить меню правой кнопки

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

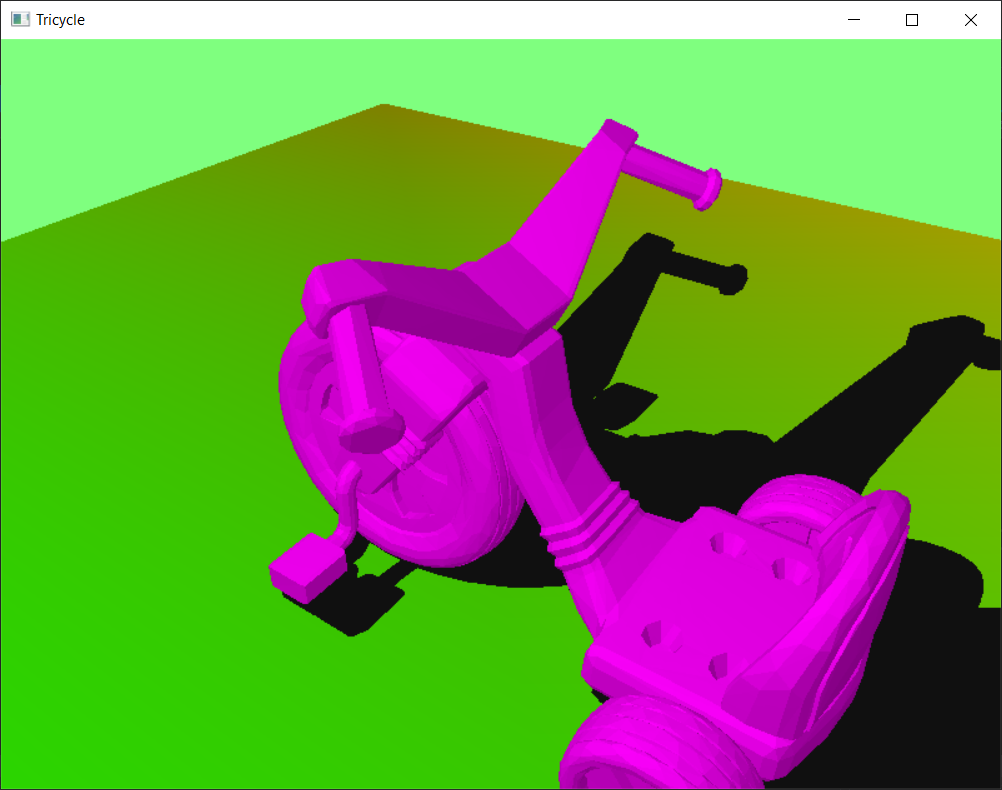
return 0;

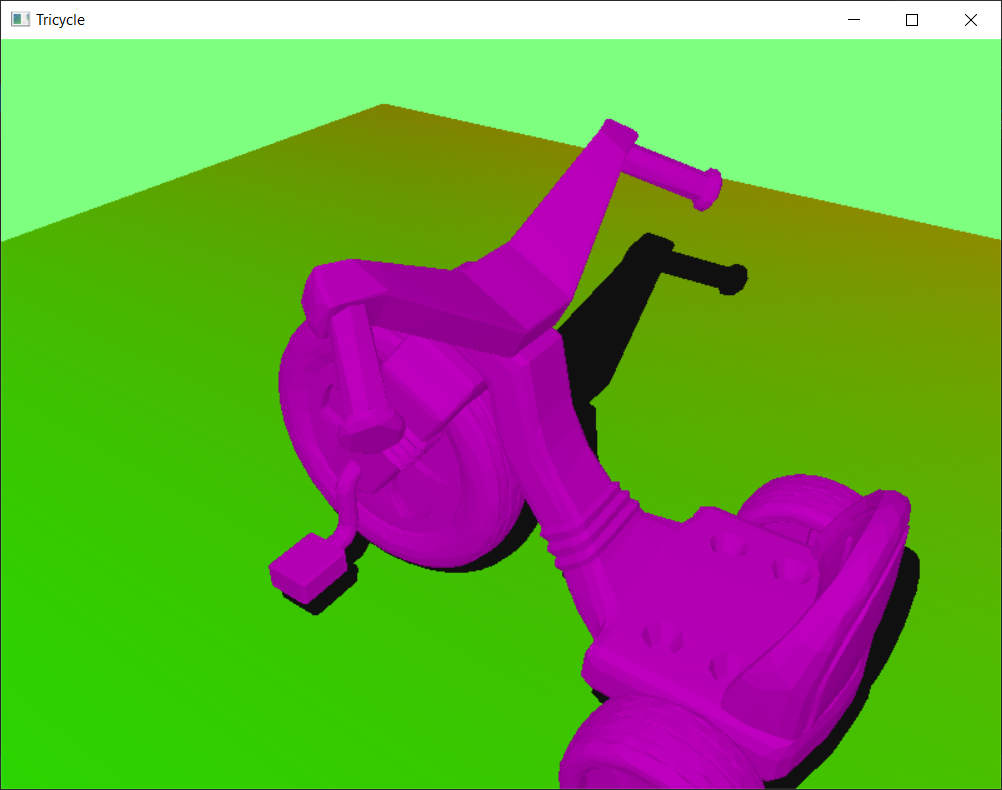
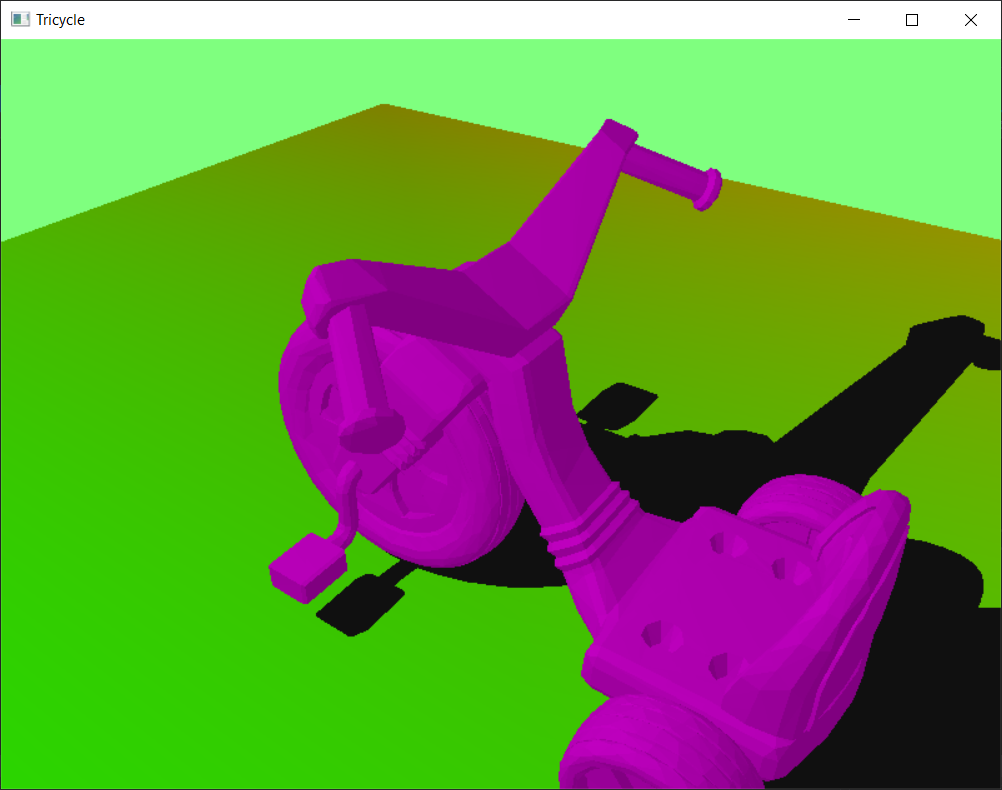
}

}

int main(int argc, char\* argv[]) { return Task\_4\_6::main(argc, argv); }

**Результат листинга 6:**



**Рисунок 6.** Результат листинга 6

**Листинг 7:**

#include "glew.h"

#include "glut.h"

#include <windows.h>

#include <math.h>

#include <stdio.h>

namespace Task\_4\_7

{

#define GLT\_PI 3.14159265358979323846

#define GLT\_PI\_DIV\_180 0.017453292519943296

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

// Данные о свете и материале

GLfloat fLightPos[4] = { -100.0f, 100.0f, 50.0f,

1.0f }; // Point source

GLfloat fLightPosMirror[4] = { -100.0f, -100.0f,

50.0f, 1.0f };

GLfloat fNoLight[] = { 0.0f, 0.0f, 0.0f, 0.0f };

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

GLfloat fBrightLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };

static GLfloat yRot = 0.0f;//Угол поворота, задейст\_вованный в анимации

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

// Для достижения наилучших результатов, поместим его в списке отображения

// Рисуем тор (бублик) при z = fZVal... в плоскости XY

// Скалярное масштабирование вектора

typedef struct { // Рамка для контейнера ссылок

float x;

float y;

float z;

} Position;

Position spheres[50];

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 еди\_ничсного 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);

// Первая точка

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()

{

// Серый фон

glClearColor(fLowLight[0], fLowLight[1],

fLowLight[2], fLowLight[3]);

// Задние части многоугольников отбрасываются

glCullFace(GL\_BACK);

glFrontFace(GL\_CCW);

glEnable(GL\_CULL\_FACE);

glEnable(GL\_DEPTH\_TEST);

// Устанавливаются параметры источника света:

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

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

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

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

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

// Преимущественно используется согласование свойств материала glEnable(GL\_COLOR\_MATERIAL);

glColorMaterial(GL\_FRONT,

GL\_AMBIENT\_AND\_DIFFUSE);

glMateriali(GL\_FRONT, GL\_SHININESS, 128);

for (int i = 0; i < 50; i++)

{

spheres[i].x = (float)((rand() % 400) - 200) \* 0.1f;

spheres[i].y = 1.0f;

spheres[i].z = (float)((rand() % 400) - 200) \* 0.1f;

}

}

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

//////

// Рисуем землю как стороны треугольников. Модели за\_тенения и цветные модели // установлены так, что мы в конечном итоге получим черно - белую шахматную

// доску.

void DrawGround(void)

{

GLfloat fExtent = 20.0f;

GLfloat fStep = 0.5f;

GLfloat y = 0.0f;

GLfloat fColor;

GLfloat iStrip, iRun;

GLint iBounce = 0;

glShadeModel(GL\_FLAT);

for (iStrip = -fExtent; iStrip <= fExtent; iStrip

+= fStep)

{

glBegin(GL\_TRIANGLE\_STRIP);

for (iRun = fExtent; iRun >= -fExtent;

iRun -= fStep)

{

if ((iBounce % 2) == 0)

fColor = 1.0f;

else

fColor = 0.0f;

glColor4f(fColor, fColor, fColor,

0.5f);

glVertex3f(iStrip, y, iRun);

glVertex3f(iStrip + fStep, y, iRun);

iBounce++;

}

glEnd();

}

glShadeModel(GL\_SMOOTH);

}

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

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

// Рисуем случайных обитателей и вращающихся тор / сферу

void DrawWorld(void)

{

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

for (int i = 0; i < 50; i++)

{

glPushMatrix();

glTranslatef(spheres[i].x, spheres[i].y, spheres[i].z);

glutSolidSphere(0.1f, 13, 26);

glPopMatrix();

}

glPushMatrix();

glTranslatef(0.0f, 1.3f, -3.5f);

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, 61, 37);

glPopMatrix();

}

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

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

// Вызывается для рисования сцены

void RenderScene(void)

{

// Очищаем окно текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT |

GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix();

// Источник света помещается под полом и освещает

// "отраженный" мир

glLightfv(GL\_LIGHT0, GL\_POSITION,

fLightPosMirror);

glPushMatrix();

glFrontFace(GL\_CW); // Зеркально отображается геометрия,

// инвертируется ориентация

glScalef(1.0f, -1.0f, 1.0f);

DrawWorld();

glFrontFace(GL\_CCW);

glPopMatrix();

// Над "отраженными" объектами рисуется

// прозрачная земля

glDisable(GL\_LIGHTING);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA,

GL\_ONE\_MINUS\_SRC\_ALPHA);

DrawGround();

glDisable(GL\_BLEND);

glEnable(GL\_LIGHTING);

// Восстанавливается правильное освещение и

// правильно рисуется мир

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

DrawWorld();

glPopMatrix();

// Переключение буферов

glutSwapBuffers();

}

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

//////

//Вызывается библиотекой GLUT в холостом состоянии (окно не меняет //размера и не перемещается)

void TimerFunction(int value)

{

yRot += 1.0f; // Обновление коэффициента пово\_рота

// Перерисовка сцены

glutPostRedisplay();

// Сброс таймера

glutTimerFunc(10, 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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Генерируется перспективная проекция

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

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glRotatef(20.0f, 1.0f, 0.0f, 0.0f);

glTranslatef(0.0f, -3.0f, -4.0f);

}

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

////////

// Точка входа программы

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB |

GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("OpenGL Blending and Transparency");

glutReshapeFunc(ChangeSize);

glutDisplayFunc(RenderScene);

glutTimerFunc(50, TimerFunction, 1);

SetupRC();

glutMainLoop();

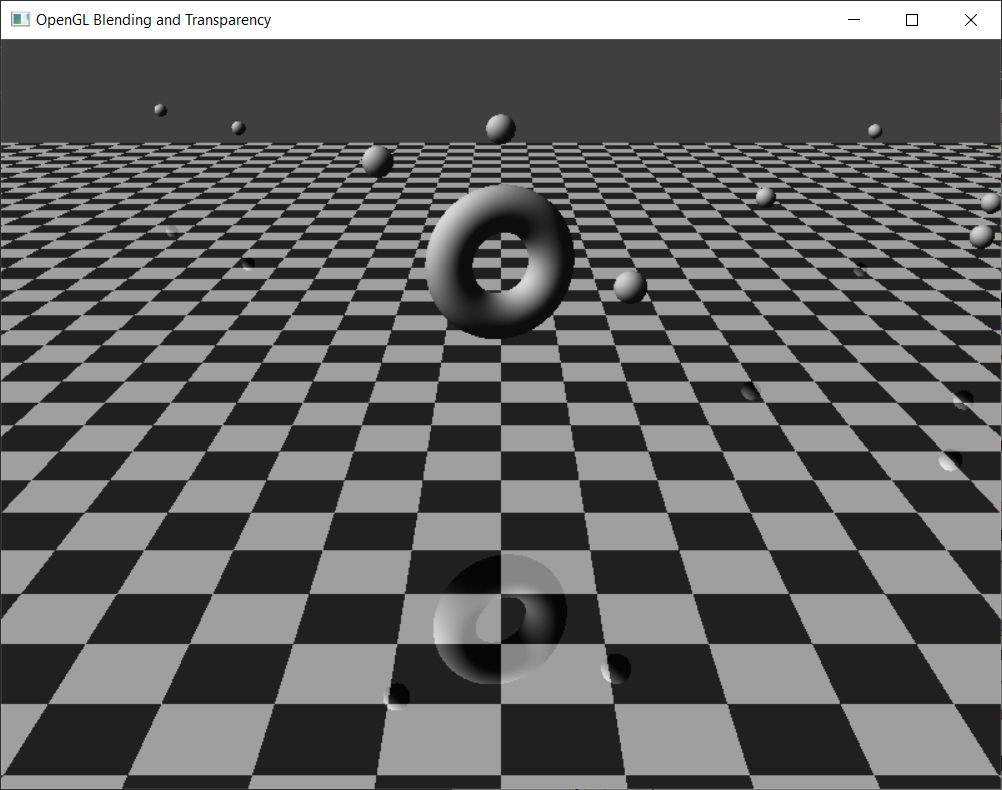
return 0;

}

}

int main(int argc, char\* argv[]) { return Task\_4\_7::main(argc, argv); }

**Результат листинга 7:**



**Рисунок 7.** Результат листинга 7

**Листинг 8:**

#include "glew.h"

#include "glut.h"

#include <windows.h>

#include <math.h>

#include <stdio.h>

namespace Task\_4\_8

{

#define GLT\_PI 3.14159265358979323846

#define GLT\_PI\_DIV\_180 0.017453292519943296

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

// Данные о свете и материале

GLfloat fLightPos[4] = { -100.0f, 100.0f, 50.0f,

1.0f }; // Point source

GLfloat fLightPosMirror[4] = { -100.0f, -100.0f,

50.0f, 1.0f };

GLfloat fNoLight[] = { 0.0f, 0.0f, 0.0f, 0.0f };

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

GLfloat fBrightLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };

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

static GLfloat yRot = 0.0f;//Угол поворота, задейст\_вованный в анимации

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

// Для достижения наилучших результатов, поместим его в списке отображения

// Рисуем тор (бублик) при z = fZVal... в плоскости XY

// Скалярное масштабирование вектора

typedef struct { // Рамка для контейнера ссылок

float x;

float y;

float z;

} Position;

Position spheres[50];

bool greyColorEnabled = true;

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 еди\_ничсного 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);

// Первая точка

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()

{

// Серый фон

glClearColor(fLowLight[0], fLowLight[1],

fLowLight[2], fLowLight[3]);

// Задние части многоугольников отбрасываются

glCullFace(GL\_BACK);

glFrontFace(GL\_CCW);

glEnable(GL\_CULL\_FACE);

glEnable(GL\_DEPTH\_TEST);

// Устанавливаются параметры источника света:

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

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

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

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

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

// Преимущественно используется согласование свойств материала glEnable(GL\_COLOR\_MATERIAL);

glColorMaterial(GL\_FRONT,

GL\_AMBIENT\_AND\_DIFFUSE);

glMateriali(GL\_FRONT, GL\_SHININESS, 128);

for (int i = 0; i < 50; i++)

{

spheres[i].x = (float)((rand() % 400) - 200) \* 0.1f;

spheres[i].y = 1.0f;

spheres[i].z = (float)((rand() % 400) - 200) \* 0.1f;

}

// Установка параметров тумана

glEnable(GL\_FOG); // Включается туман

glFogfv(GL\_FOG\_COLOR, fLowLight);

// Цвет тумана соответствует фону

glFogf(GL\_FOG\_START, 1.0f); // Насколько далеко

// начинается туман

glFogf(GL\_FOG\_END, 25.0f); // Насколько далеко

// заканчивается туман

glFogf(GL\_FOG\_DENSITY, 0.1f);

glFogi(GL\_FOG\_MODE, GL\_LINEAR); // Какое уравнение

// тумана используется

}

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

//////

// Рисуем землю как стороны треугольников. Модели затенения и цветные модели // установлены так, что мы в конечном итоге получим черно - белую шахматную

// доску.

void DrawGround(void)

{

GLfloat fExtent = 20.0f;

GLfloat fStep = 0.5f;

GLfloat y = 0.0f;

GLfloat fColor;

GLfloat iStrip, iRun;

GLint iBounce = 0;

glShadeModel(GL\_FLAT);

for (iStrip = -fExtent; iStrip <= fExtent; iStrip

+= fStep)

{

glBegin(GL\_TRIANGLE\_STRIP);

for (iRun = fExtent; iRun >= -fExtent;

iRun -= fStep)

{

if ((iBounce % 2) == 0)

fColor = 1.0f;

else

fColor = 0.0f;

glColor4f(fColor, fColor, fColor,

0.5f);

glVertex3f(iStrip, y, iRun);

glVertex3f(iStrip + fStep, y, iRun);

iBounce++;

}

glEnd();

}

glShadeModel(GL\_SMOOTH);

}

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

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

// Рисуем случайных обитателей и вращающихся тор / сферу

void DrawWorld(void)

{

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

for (int i = 0; i < 50; i++)

{

glPushMatrix();

glTranslatef(spheres[i].x, spheres[i].y, spheres[i].z);

glutSolidSphere(0.1f, 13, 26);

glPopMatrix();

}

glPushMatrix();

glTranslatef(0.0f, 1.3f, -3.5f);

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, 61, 37);

glPopMatrix();

}

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

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

// Вызывается для рисования сцены

void RenderScene(void)

{

// Очищаем окно текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT |

GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix();

glDisable(GL\_LIGHTING);

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA,

GL\_ONE\_MINUS\_SRC\_ALPHA);

DrawGround();

glDisable(GL\_BLEND);

glEnable(GL\_LIGHTING);

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

DrawWorld();

glPopMatrix();

// Переключение буферов

glutSwapBuffers();

}

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

//////

//Вызывается библиотекой GLUT в холостом состоянии (окно не меняет //размера и не перемещается)

void TimerFunction(int value)

{

yRot += 1.0f; // Обновление коэффициента пово\_рота

// Перерисовка сцены

glutPostRedisplay();

// Сброс таймера

glutTimerFunc(10, 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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Генерируется перспективная проекция

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

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glRotatef(20.0f, 1.0f, 0.0f, 0.0f);

glTranslatef(0.0f, -3.0f, -4.0f);

}

void processMenuEvents(int option) {

switch (option) {

case 1:

if (greyColorEnabled)

glFogfv(GL\_FOG\_COLOR, redLight);

else

glFogfv(GL\_FOG\_COLOR, fLowLight);

greyColorEnabled = !greyColorEnabled;

break;

case 2:

glFogi(GL\_FOG\_MODE, GL\_LINEAR);

break;

case 3:

glFogi(GL\_FOG\_MODE, GL\_EXP);

break;

case 4:

glFogi(GL\_FOG\_MODE, GL\_EXP2);

break;

}

RenderScene();

}

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

////////

// Точка входа программы

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB |

GLUT\_DEPTH);

glutInitWindowSize(800, 600);

glutCreateWindow("OpenGL Blending and Transparency");

glutReshapeFunc(ChangeSize);

glutDisplayFunc(RenderScene);

glutTimerFunc(50, TimerFunction, 1);

glutCreateMenu(processMenuEvents);

//добавить записи в нашем меню

glutAddMenuEntry("Switch color", 1);

glutAddMenuEntry("Linear mode", 2);

glutAddMenuEntry("Exp mode", 3);

glutAddMenuEntry("Exp2 mode", 4);

// прикрепить меню правой кнопки

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

SetupRC();

glutMainLoop();

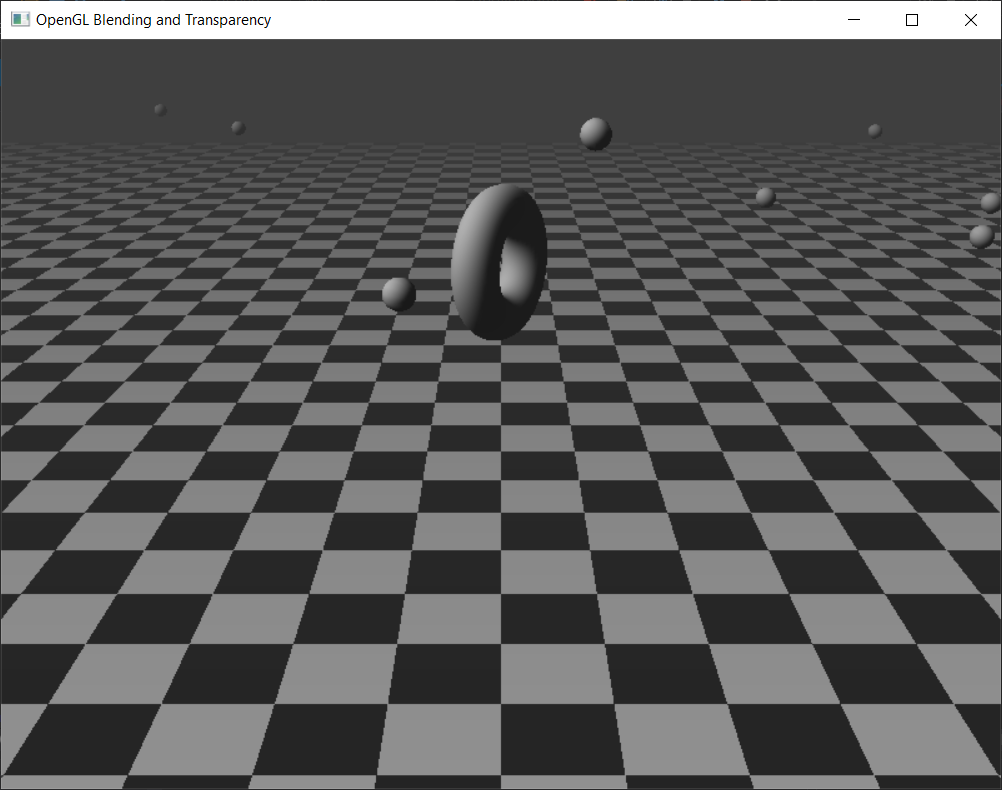
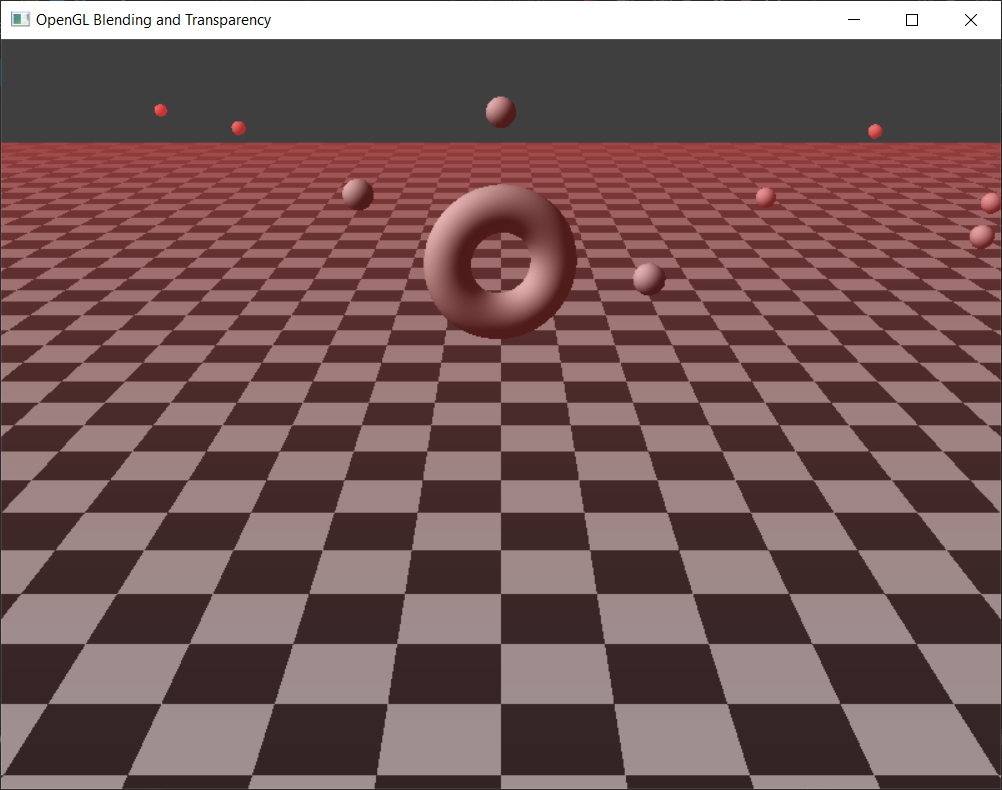
return 0;

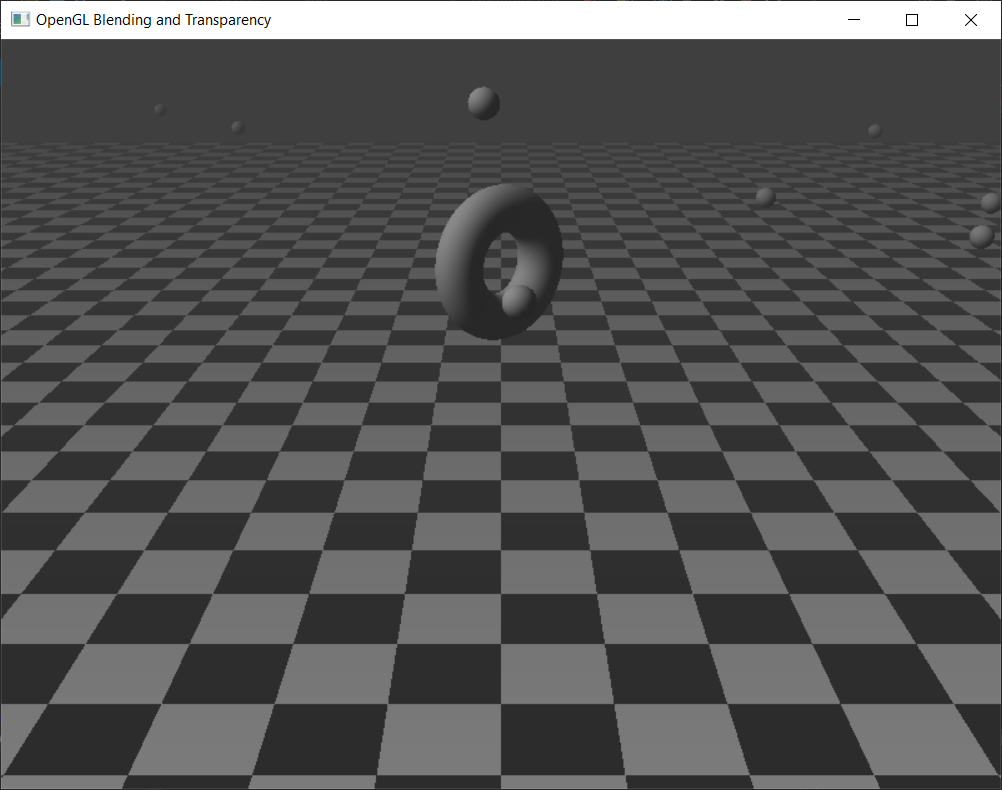
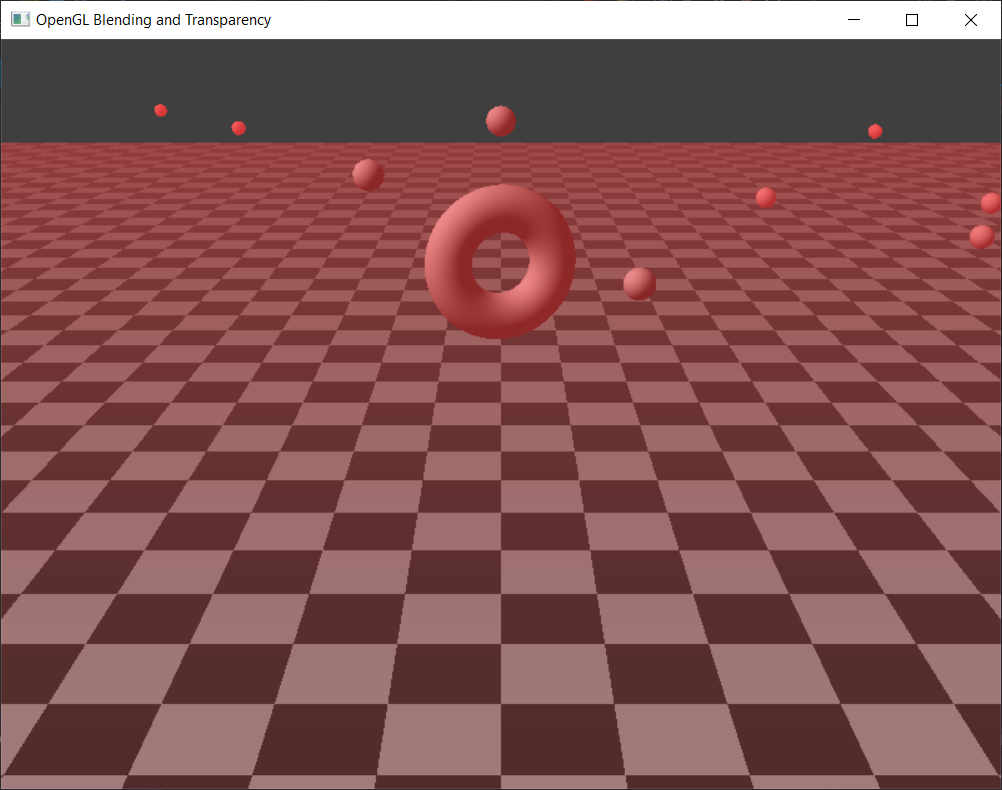
}

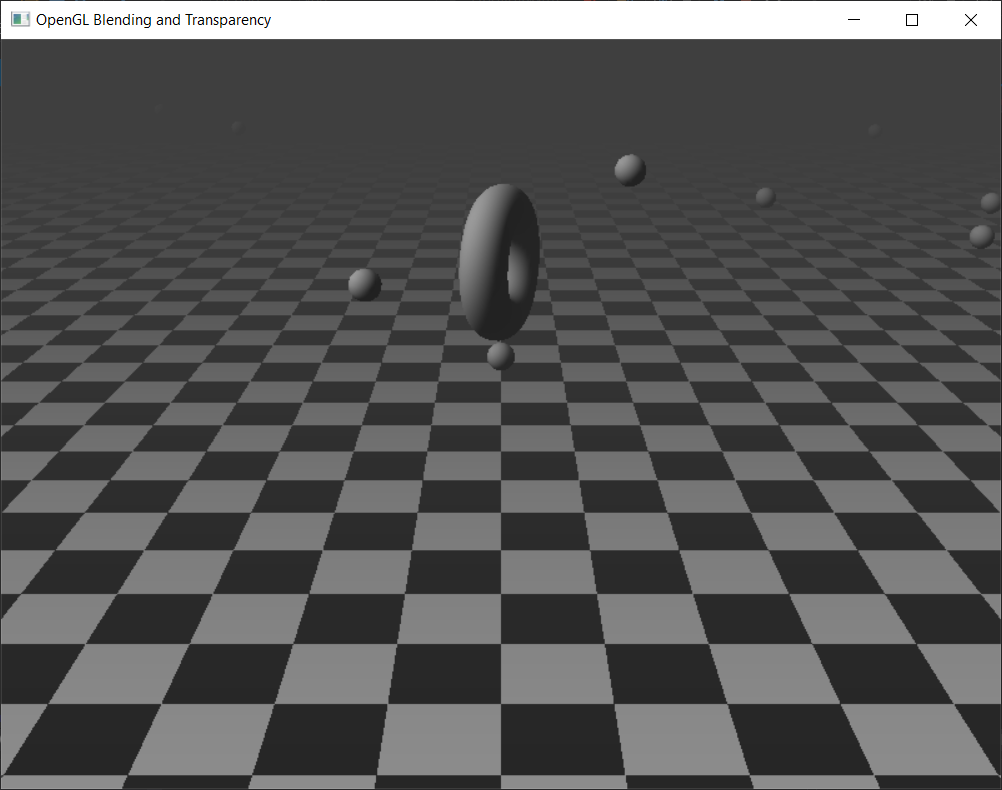
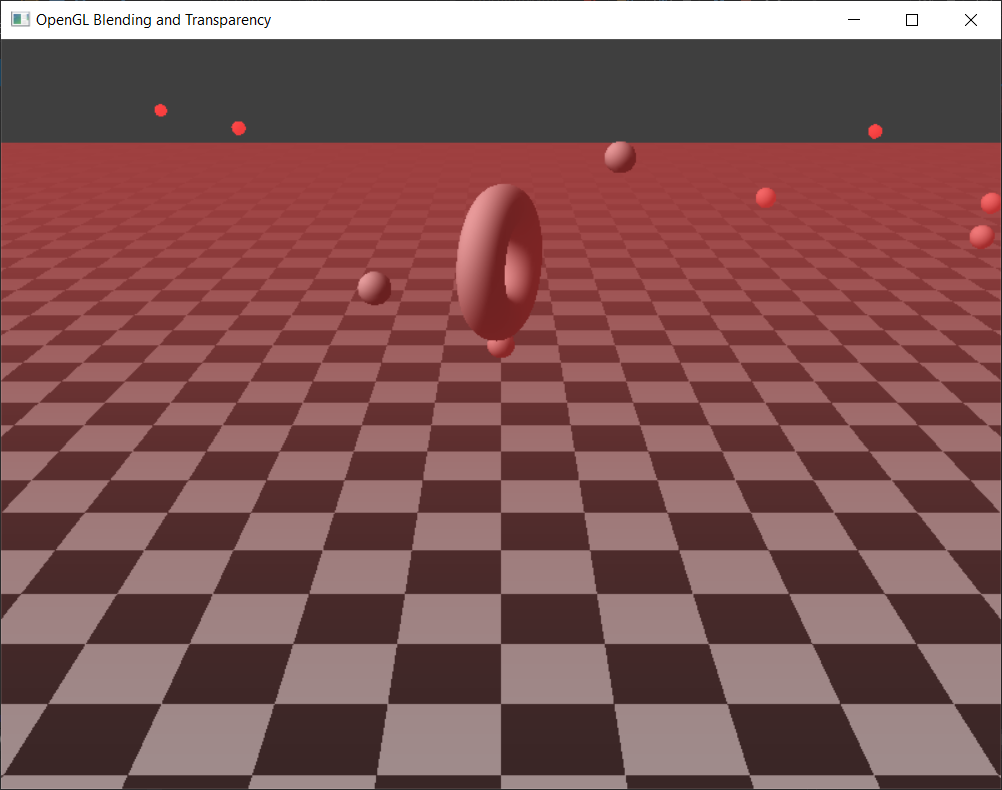
}

int main(int argc, char\* argv[]) { return Task\_4\_8::main(argc, argv); }

**Результат листинга 8:**

**Рисунок 8.** Результат листинга 8

**Изменение в листингах 2, 3, 4, 5 ЛР 2:**

void SetupRC()

{

...

glEnable(GL\_BLEND);

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

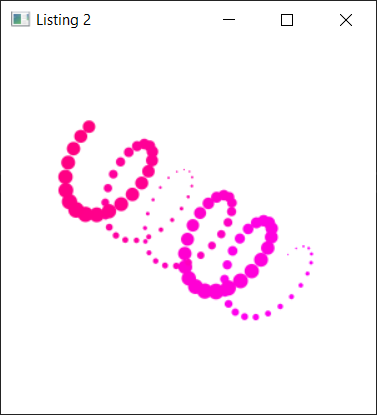
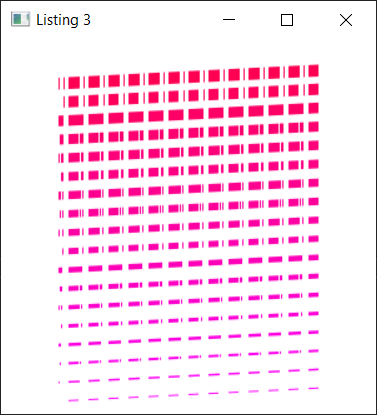
glEnable(GL\_POINT\_SMOOTH); // Сглаживание точек

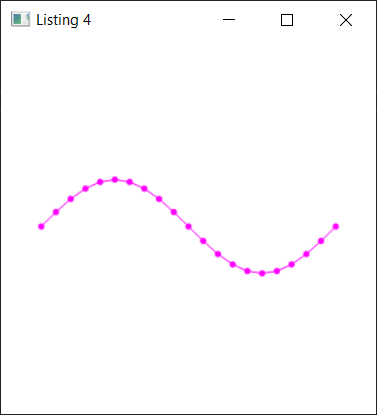
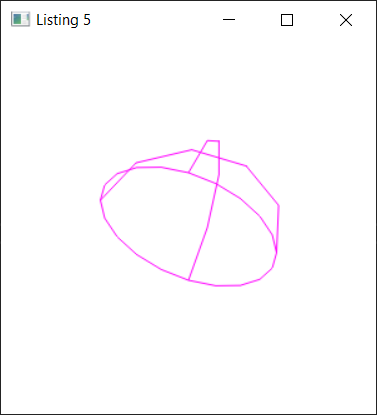
glEnable(GL\_LINE\_SMOOTH); // Сглаживание линий

glEnable(GL\_POLYGON\_SMOOTH); // Сглаживание краев многоугольников

}

**Результат изменений:**

**Рисунок 9.** Результат изменений

**Вспомогательный класс objLoader:**

***ObjLoader.h***

#pragma once

#include <vector>

#include <iostream>

#include "glut.h"

using namespace std;

class ObjLoader

{

public:

struct vertex

{

float x;

float y;

float z;

};

ObjLoader(string filename);// Чтение функции

void Draw();// Функция рисования

private:

vector<vector<GLfloat>> v;// Сохраняем координаты вершины (x, y, z)

vector<vector<GLint>> f;// Сохраняем три вершинных индекса лица

};

***ObjLoader.cpp***

#include "ObjLoader.h"

#include <fstream>

#include <string>

#include <sstream>

ObjLoader::ObjLoader(string filename)

{

ifstream file(filename);

string line;

while (getline(file, line))

{

if (line.substr(0, 1) == "v")

{

vector<GLfloat> Point;

GLfloat x, y, z;

istringstream s(line.substr(2));

s >> x; s >> y; s >> z;

Point.push\_back(x);

Point.push\_back(y);

Point.push\_back(z);

v.push\_back(Point);

}

else if (line.substr(0, 1) == "f")

{

vector<GLint> vIndexSets;

GLint u, v, w;

istringstream vtns(line.substr(2));

vtns >> u; vtns >> v; vtns >> w;

vIndexSets.push\_back(u - 1);

vIndexSets.push\_back(v - 1);

vIndexSets.push\_back(w - 1);

f.push\_back(vIndexSets);

}

}

file.close();

}

void ObjLoader::Draw()

{

glBegin(GL\_TRIANGLES);//Начать рисовать

for (int i = 0; i < f.size(); i++) {

GLfloat VN[3];//Нормальный

// Три вершины

vertex a, b, c, normal;

if ((f[i]).size() != 3) {

cout << "ERROR::THE SIZE OF f IS NOT 3!" << endl;

}

else {

GLint firstVertexIndex = (f[i])[0];// Удалить индекс вершины

GLint secondVertexIndex = (f[i])[1];

GLint thirdVertexIndex = (f[i])[2];

a.x = (v[firstVertexIndex])[0];// Первая вершина

a.y = (v[firstVertexIndex])[1];

a.z = (v[firstVertexIndex])[2];

b.x = (v[secondVertexIndex])[0]; // Вторая вершина

b.y = (v[secondVertexIndex])[1];

b.z = (v[secondVertexIndex])[2];

c.x = (v[thirdVertexIndex])[0]; // Третья вершина

c.y = (v[thirdVertexIndex])[1];

c.z = (v[thirdVertexIndex])[2];

GLfloat vec1[3], vec2[3], vec3[3];// Рассчитать нормальный вектор

//(x2-x1,y2-y1,z2-z1)

vec1[0] = a.x - b.x;

vec1[1] = a.y - b.y;

vec1[2] = a.z - b.z;

//(x3-x2,y3-y2,z3-z2)

vec2[0] = a.x - c.x;

vec2[1] = a.y - c.y;

vec2[2] = a.z - c.z;

//(x3-x1,y3-y1,z3-z1)

vec3[0] = vec1[1] \* vec2[2] - vec1[2] \* vec2[1];

vec3[1] = vec2[0] \* vec1[2] - vec2[2] \* vec1[0];

vec3[2] = vec2[1] \* vec1[0] - vec2[0] \* vec1[1];

GLfloat D = sqrt(pow(vec3[0], 2) + pow(vec3[1], 2) + pow(vec3[2], 2));

VN[0] = vec3[0] / D;

VN[1] = vec3[1] / D;

VN[2] = vec3[2] / D;

glNormal3f(VN[0], VN[1], VN[2]);// Рисуем нормальный вектор

glVertex3f(a.x, a.y, a.z);// Рисуем треугольник

glVertex3f(b.x, b.y, b.z);

glVertex3f(c.x, c.y, c.z);

}

}

glEnd();

}

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки реализации рассеянного, диффузного и отраженного света, задания материала объектов, импорта obj-файла, создания падающей тени, эффекта зеркального отражения и тумана, прожектора, сглаживания.