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»***

Практическая работа

«Освещение в OpenGL»

ДИСЦИПЛИНА: «Компьютерная графика»

Выполнил: студент гр. ИУК5-32Б _____ (Подпись) __Ли Р. В.__ (Ф.И.О.)

Проверил: _____ (Подпись) __Широкова Е. В,__ (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга , 2023

Цели: 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 8

Задание 1.

Продемонстрировать на 6 примерах различные варианты работы функций glLightModelfv и glColorMaterial. Модели выбрать согласно варианту (Примечание: для создания эффекта реалистичности изображения рекомендуется получить координаты полигонов объектов из сторонних приложений.)

8) Поезд

Задание 2.

На основе объекта из Задания 1 продемонстрировать установку источников света согласно варианту (Примечание: для тех кто претендует на оценку «отлично» предусмотреть включение выключение источника света через меню.)

8) Установить пять источников света (находятся в трех октантах) одного цвета разной интенсивности.

Задание 3.

На основе объекта из задания 1 продемонстрировать создание бликов на объекте.

Листинг 1

```
// TODO: variant 8
#include <GL/freeglut.h>
#include <GL/gl.h>
#include <cmath>
const float PI = 3.14159265358979323846;
double angle = 0;

class Train {
private:
public:
void drawCube(double sizes[], double position[]) {
    double baseX = sizes[0];
    double baseY = sizes[1];
    double baseZ = sizes[2];

    GLfloat mat_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess);

    glBegin(GL_QUADS);
    // Bottom face
    glNormal3f(0.0f, -1.0f, 0.0f);
    glVertex3f(position[0], position[1], position[2]);
    glVertex3f(position[0], position[1], position[2] + baseZ);
    glVertex3f(position[0] + baseX, position[1], position[2] + baseZ);
    glVertex3f(position[0] + baseX, position[1], position[2]);

    // Top face
    glNormal3f(0.0f, 1.0f, 0.0f);
    glVertex3f(position[0], position[1] + baseY, position[2]);
    glVertex3f(position[0] + baseX, position[1] + baseY, position[2]);
    glVertex3f(position[0] + baseX, position[1] + baseY, position[2] +
baseZ);
    glVertex3f(position[0], position[1] + baseY, position[2] + baseZ);

    // Front face
    glNormal3f(0.0f, 0.0f, 1.0f);
    glVertex3f(position[0], position[1], position[2] + baseZ);
    glVertex3f(position[0], position[1] + baseY, position[2] + baseZ);
    glVertex3f(position[0] + baseX, position[1] + baseY, position[2] +
baseZ);
    glVertex3f(position[0] + baseX, position[1], position[2] + baseZ);
```

```
    // Back face
    glNormal3f(0.0f, 0.0f, -1.0f);
    glVertex3f(position[0], position[1], position[2]);
    glVertex3f(position[0] + baseX, position[1], position[2]);
    glVertex3f(position[0] + baseX, position[1] + baseY, position[2]);
    glVertex3f(position[0], position[1] + baseY, position[2]);

    // Left face
    glNormal3f(-1.0f, 0.0f, 0.0f);
    glVertex3f(position[0], position[1], position[2]);
    glVertex3f(position[0], position[1], position[2] + baseZ);
    glVertex3f(position[0], position[1] + baseY, position[2] + baseZ);
    glVertex3f(position[0], position[1] + baseY, position[2]);

    // Right face
    glNormal3f(1.0f, 0.0f, 0.0f);
    glVertex3f(position[0] + baseX, position[1], position[2]);
    glVertex3f(position[0] + baseX, position[1], position[2] + baseZ);
    glVertex3f(position[0] + baseX, position[1] + baseY, position[2] +
baseZ);
    glVertex3f(position[0] + baseX, position[1] + baseY, position[2]);

    glEnd();
}

  void drawVerticalCylinder(float position[], float radius, float height,
                            int segments) {
    glPushMatrix();
    glTranslatef(position[0], position[1], position[2]);

    GLfloat mat_diffuse[] = {0.8, 0.8, 0.8, 1.0};
    GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat mat_shininess[] = {50.0};
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

    glBegin(GL_QUAD_STRIP);
    for (int i = 0; i <= segments; ++i) {
      float theta =
          2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
      float x = radius * cos(theta);
      float z = radius * sin(theta);

      glNormal3f(x / radius, 0.0f, z / radius);
      glVertex3f(x, 0.0f, z);
      glVertex3f(x, height, z);
    }
    glEnd();

    glBegin(GL_TRIANGLE_FAN);
    glNormal3f(0.0f, -1.0f, 0.0f);
    glVertex3f(0.0f, 0.0f, 0.0f);
    for (int i = 0; i <= segments; ++i) {
      float theta =
          2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
      float x = radius * cos(theta);
      float z = radius * sin(theta);
```

```
      glVertex3f(x, 0.0f, z);
    }
    glEnd();

    glBegin(GL_TRIANGLE_FAN);
    glNormal3f(0.0f, 1.0f, 0.0f);
    glVertex3f(0.0f, height, 0.0f);
    for (int i = 0; i <= segments; ++i) {
      float theta =
          2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
      float x = radius * cos(theta);
      float z = radius * sin(theta);

      glVertex3f(x, height, z);
    }
    glEnd();
    glPopMatrix();
  }

  void drawWheel(float position[], float radius, float height, int segments)
{
    glPushMatrix();
    glTranslatef(position[0], position[1], position[2]);

    GLfloat mat_diffuse[] = {0.8, 0.8, 0.8, 1.0};
    GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat mat_shininess[] = {50.0};
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

    glBegin(GL_QUAD_STRIP);
    for (int i = 0; i <= segments; ++i) {
      float theta =
          2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
      float x = radius * cos(theta);
      float y = radius * sin(theta);

      glNormal3f(0.0f, y, x);
      glVertex3f(0, y, x);
      glVertex3f(height, y, x);
    }
    glEnd();

    glBegin(GL_TRIANGLE_FAN);
    glNormal3f(0.0f, 0.0f, -1.0f);
    glVertex3f(0.0f, 0.0f, 0.0f);
    for (int i = 0; i <= segments; ++i) {
      float theta =
          2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
      float x = radius * cos(theta);
      float y = radius * sin(theta);

      glVertex3f(0, y, x);
    }
    glEnd();

    glBegin(GL_TRIANGLE_FAN);
```

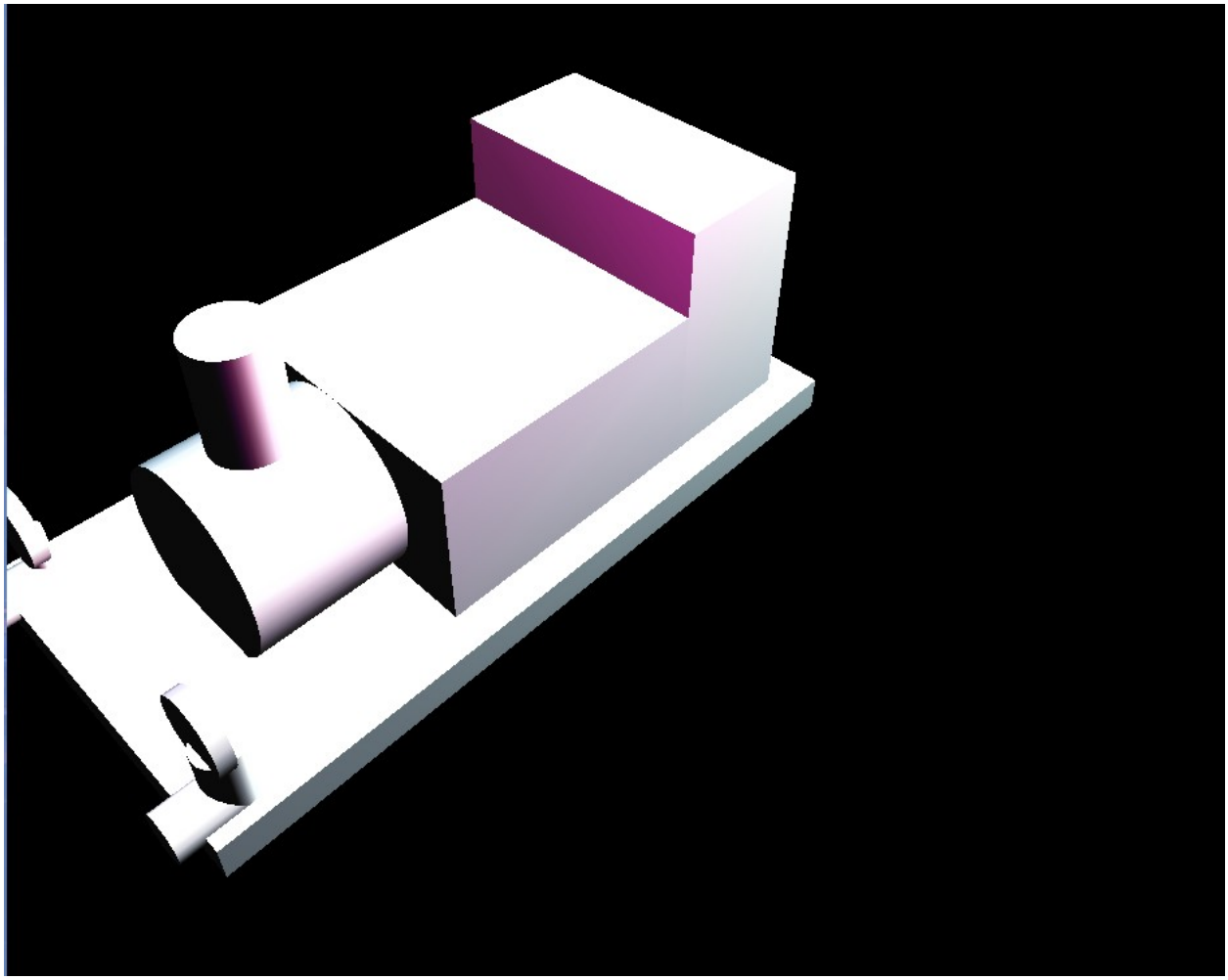
```
  glNormal3f(0.0f, 0.0f, 1.0f);
  glVertex3f(0.0f, 0.0f, height);
  for (int i = 0; i <= segments; ++i) {
    float theta =
        2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
    float x = radius * cos(theta);
    float y = radius * sin(theta);

    glVertex3f(height, y, x);
  }
  glEnd();
  glPopMatrix();
}

void drawHorizontalCylinder(float position[], float radius, float height,
                            int segments) {
  glPushMatrix();
  glTranslatef(position[0], position[1], position[2]);

  GLfloat mat_diffuse[] = {0.8, 0.8, 0.8, 1.0};
  GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
  GLfloat mat_shininess[] = {50.0};
  glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

  glBegin(GL_QUAD_STRIP);
  for (int i = 0; i <= segments; ++i) {
    float theta =
        2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
    float x = radius * cos(theta);
    float y = radius * sin(theta);

    glNormal3f(x / radius, y / radius, 0.0f);
    glVertex3f(x, y, 0.0f);
    glVertex3f(x, y, height);
  }
  glEnd();

  glBegin(GL_TRIANGLE_FAN);
  glNormal3f(0.0f, 0.0f, -1.0f);
  glVertex3f(0.0f, 0.0f, 0.0f);
  for (int i = 0; i <= segments; ++i) {
    float theta =
        2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
    float x = radius * cos(theta);
    float y = radius * sin(theta);

    glVertex3f(x, y, 0.0f);
  }
  glEnd();

  glBegin(GL_TRIANGLE_FAN);
  glNormal3f(0.0f, 0.0f, 1.0f);
  glVertex3f(0.0f, 0.0f, height);
  for (int i = 0; i <= segments; ++i) {
    float theta =
        2.0f * PI * static_cast<float>(i) / static_cast<float>(segments);
    float x = radius * cos(theta);
```

```
      float y = radius * sin(theta);

      glVertex3f(x, y, height);
    }
    glEnd();
    glPopMatrix();
  }

  void drawTrain() {
    drawCube(new double[]{300, 75, 900}, new double[]{100, -75, 50});
    drawCube(new double[]{500, 50, 1000}, new double[]{0, 0, 0});
    drawCube(new double[]{400, 350, 200}, new double[]{50, 50, 50});
    drawCube(new double[]{400, 225, 400}, new double[]{50, 50, 250});
    drawHorizontalCylinder(new float[]{250, 150, 650}, 125, 200, 50);
    drawVerticalCylinder(new float[]{250, 275, 750}, 50, 150, 40);
    drawHorizontalCylinder(new float[]{50, 50, 950}, 30, 80, 20);
    drawHorizontalCylinder(new float[]{450, 50, 950}, 30, 80, 20);
    drawVerticalCylinder(new float[]{50, 50, 950}, 15, 80, 20);
    drawVerticalCylinder(new float[]{450, 50, 950}, 30, 80, 20);
    drawHorizontalCylinder(new float[]{60, 150, 950}, 50, 20, 20);
    drawHorizontalCylinder(new float[]{440, 150, 950}, 50, 20, 20);
    drawWheel(new float[]{75, -60, 200}, 75, 20, 20);
    drawWheel(new float[]{400, -60, 200}, 75, 20, 20);
    drawWheel(new float[]{75, -60, 400}, 75, 20, 20);
    drawWheel(new float[]{400, -60, 400}, 75, 20, 20);
    drawWheel(new float[]{75, -60, 800}, 75, 20, 20);
    drawWheel(new float[]{400, -60, 800}, 75, 20, 20);
  }
} train;

void init() {
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  glEnable(GL_DEPTH_TEST);

  glEnable(GL_LIGHTING);

  GLfloat light_position1[] = {1000.0f, 1000.0f, 1000.0f, 1.0f};
  GLfloat light_color1[] = {0.0f, 1.0f, 1.0f, 1.0f}; // cyan

  glLightfv(GL_LIGHT0, GL_POSITION, light_position1);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light_color1);
  glEnable(GL_LIGHT0);

  GLfloat light_position2[] = {100.0f, 100.0f, 1000.0f, 1.0f};
  GLfloat light_color2[] = {1.0f, 0.0f, 1.0f, 1.0f}; // pink
  glLightfv(GL_LIGHT1, GL_POSITION, light_position2);
  glLightfv(GL_LIGHT1, GL_DIFFUSE, light_color2);
  glEnable(GL_LIGHT1);

  GLfloat light_position3[] = {200.0f, 300.0f, 1000.0f, 1.0f};
  GLfloat light_color3[] = {1.0f, 1.0f, 0.0f, 1.0f}; // yellow-ish?
  glLightfv(GL_LIGHT2, GL_POSITION, light_position3);
  glLightfv(GL_LIGHT2, GL_DIFFUSE, light_color3);
  glEnable(GL_LIGHT2);

  GLfloat light_position4[] = {200.0f, 600.0f, 500.0f, 1.0f};
  GLfloat light_color4[] = {0.5f, 0.5f, 1.0f, 1.0f}; // dark blue?
  glLightfv(GL_LIGHT3, GL_POSITION, light_position4);
  glLightfv(GL_LIGHT3, GL_DIFFUSE, light_color4);
```

```
  glEnable(GL_LIGHT3);

  GLfloat light_position5[] = {700.0f, 200.0f, 500.0f, 1.0f};
  GLfloat light_color5[] = {1.0f, 1.0f, 1.0f, 1.0f}; // white
  glLightfv(GL_LIGHT4, GL_POSITION, light_position5);
  glLightfv(GL_LIGHT4, GL_DIFFUSE, light_color5);
  glEnable(GL_LIGHT4);
}

void menu(int value) {
  switch (value) {
  case 1:
    glEnable(GL_LIGHT0);
    break;
  case 2:
    glDisable(GL_LIGHT0);
    break;
  case 3:
    glEnable(GL_LIGHT1);
    break;
  case 4:
    glDisable(GL_LIGHT1);
    break;
  case 5:
    glEnable(GL_LIGHT2);
    break;
  case 6:
    glDisable(GL_LIGHT2);
    break;
  case 7:
    glEnable(GL_LIGHT3);
    break;
  case 8:
    glDisable(GL_LIGHT3);
    break;
  case 9:
    glEnable(GL_LIGHT4);
    break;
  case 10:
    glDisable(GL_LIGHT4);
    break;
  }
  glutPostRedisplay();
}

void createMenu() {
  int menu_id = glutCreateMenu(menu);
  glutAddMenuEntry("Enable Light 1", 1);
  glutAddMenuEntry("Disable Light 1", 2);
  glutAddMenuEntry("Enable Light 2", 3);
  glutAddMenuEntry("Disable Light 2", 4);
  glutAddMenuEntry("Enable Light 3", 5);
  glutAddMenuEntry("Disable Light 3", 6);
  glutAddMenuEntry("Enable Light 4", 7);
  glutAddMenuEntry("Disable Light 4", 8);
  glutAddMenuEntry("Enable Light 5", 9);
  glutAddMenuEntry("Disable Light 5", 10);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

```
void display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();

  glRotatef(angle, 0.0f, 1.0f, 1.0f);
  gluLookAt(5.0, 5.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  train.drawTrain();

  glutSwapBuffers();
}

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  float aspectRatio = (float)w / (float)h;

  float distanceToObject = 2000;
  float fov = 60.0f;
  float nearPlane = 1000.0f;
  float farPlane = distanceToObject + 2000;

  gluPerspective(fov, aspectRatio, nearPlane, farPlane);
  gluLookAt(0.0f, 0.0f, distanceToObject, 100, -200, -500, 0.0f, 1.0f, 0.0f);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void spinView(int key, int x, int y) {
  switch (key) {
  case GLUT_KEY_LEFT:
    angle += 5.0f;
    break;
  case GLUT_KEY_RIGHT:
    angle -= 5.0f;
    break;
  }
  glutPostRedisplay();
}

int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(100, 100);
  glutCreateWindow("3D Train");
  init();
  createMenu();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutSpecialFunc(spinView);
  glutMainLoop();
  return 0;
}
```

Результаты



Задание 4.

Создать простой объект (или доработать Задание 1), который находится над поверхностью и отбрасывает тень на эту поверхность. Объект должен иметь возможность перемещаться.

8) glutSolidTeapot

Задание 5.

На основе Задания 4 добавить в проект туман режимов: GL_LINEAR , GL_EXP и GL_EXP2.

Листинг

```
#include <GL/freeglut.h>
#include <GL/gl.h>
#include <math.h>

static GLfloat xRot = 0.0f;
static GLfloat yRot = 0.0f;
// Глобальные переменные
GLfloat fLowLight[4] = {0.25, 0.25, 0.25, 1.0}; // Цвет тумана
// Коды и координаты источников света
GLfloat ambientLight[] = {0.3f, 0.3f, 0.3f, 1.0f};
GLfloat diffuseLight[] = {0.7f, 0.7f, 0.7f, 1.0f};
GLfloat specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat lightPos[] = {-75.0f, 150.0f, -50.0f, 0.0f};
GLfloat specref[] = {1.0f, 1.0f, 1.0f, 1.0f};
typedef GLfloat GLTVector3[3];
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 4x4 с плавающей //запятой
// Матрица преобразования, дающая проекцию тени
GLTMatrix shadowMat;

// Масштабирование скалярного 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));
}

// Вычитание одного вектора 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];
}

// 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];
```

```
}

// Масштабирование вектора по длине - создание единичсного вектора
void gltNormalizeVector(GLTVector3 vNormal) {
  GLfloat fLength = 1.0f / gltGetVectorLength(vNormal);
  gltScaleVector(vNormal, fLength);
}

// Три точки на плоскости расположены против часовой стрелки, вычисление
// нормали

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);
}

// 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 подстановкой
  vPlane[3] = -(vPlane[0] * vPoint3[0] + vPlane[1] * vPoint3[1] +
                vPlane[2] * vPoint3[2]);
}

// Создание матрицы теневой проекции из коэффициентов уравнения плоскости
и //
// положениесвета.Возвращаемое значение хранится в
void gltMakeShadowMatrix(GLTVector3 vPoints[3], GLTVector4 vLightPos,
                         GLTMatrix destMat) {
  GLTVector4 vPlaneEquation;
  GLfloat dot;
  gltGetPlaneEquation(vPoints[0], vPoints[1], vPoints[2], vPlaneEquation);
  // Скалярное произведение положение конуса и света
  dot = vPlaneEquation[0] * vLightPos[0] + vPlaneEquation[1] * vLightPos[1] +
        vPlaneEquation[2] * vLightPos[2] + vPlaneEquation[3] * vLightPos[3];
  // Проецируем
  // 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];
}

//////////////////////////////////////////////
// Функция, специально прорисовывающая конус
void DrawTeapot(int nShadow) {
  if (nShadow == 0)
    glColor3ub(255, 0, 255);
  else
    glColor3ub(0, 0, 0);
  glScalef(30.0, 30.0, 30.0);
  glPushMatrix();
  glutSolidTeapot(1.0f);
  glPopMatrix();
}

void RenderScene(void) {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glBegin(GL_QUADS);
  glColor3ub(0, 0, 80);
  glVertex3f(400.0f, -150.0f, -200.0f);
  glVertex3f(-400.0f, -150.0f, -200.0f);
  glColor3ub(0, 0, 255);
  glVertex3f(-400.0f, -150.0f, 200.0f);
  glVertex3f(400.0f, -150.0f, 200.0f);
  glEnd();
  glPushMatrix();
  glEnable(GL_LIGHTING);
  glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
  glRotatef(xRot, 1.0f, 0.0f, 0.0f);
  glRotatef(yRot, 0.0f, 1.0f, 0.0f);
  DrawTeapot(0);
  glPopMatrix();
  glDisable(GL_DEPTH_TEST);
  glDisable(GL_LIGHTING);
  glPushMatrix();
  glMultMatrixf((GLfloat *)shadowMat);
  glRotatef(xRot, 1.0f, 0.0f, 0.0f);
  glRotatef(yRot, 0.0f, 1.0f, 0.0f);
  DrawTeapot(1);
  glPopMatrix();
  glPushMatrix();
  glTranslatef(lightPos[0], lightPos[1], lightPos[2]);
  glColor3ub(255, 255, 0);
  glutSolidSphere(5.0f, 10, 10);
  glPopMatrix();
  glEnable(GL_DEPTH_TEST);
  glutSwapBuffers();
}

void SetupRC() {
  GLTVector3 points[3] = {{-30.0f, -149.0f, -20.0f},
                          {-30.0f, -149.0f, 20.0f},
                          {40.0f, -149.0f, 20.0f}};
  glEnable(GL_DEPTH_TEST);
```

```
  glFrontFace(GL_CCW);
  glEnable(GL_CULL_FACE);
  glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
  glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
  glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
  glEnable(GL_LIGHT0);
  glEnable(GL_COLOR_MATERIAL);
  glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
  glMaterialfv(GL_FRONT, GL_SPECULAR, specref);
  glMateriali(GL_FRONT, GL_SHININESS, 128);
  glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
  gltMakeShadowMatrix(points, lightPos, shadowMat);
}

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 (key > 356.0f)
    xRot = 0.0f;
  if (key < -1.0f)
    xRot = 355.0f;
  if (key > 356.0f)
    yRot = 0.0f;
  if (key < -1.0f)
    yRot = 355.0f;
  glutPostRedisplay();
}

void ChangeSize(int w, int h) {
  GLfloat fAspect;
  if (h == 0)
    h = 1;
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  fAspect = (GLfloat)w / (GLfloat)h;
  gluPerspective(60.0f, fAspect, 200.0, 500.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0.0f, 0.0f, -400.0f);
  glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
}

void changefog(int id) {
  switch (id) {
  case 0:
    glDisable(GL_FOG);
    glClearColor(fLowLight[0], fLowLight[1], fLowLight[2], fLowLight[3]);
    glEnable(GL_FOG);
    glFogfv(GL_FOG_COLOR, fLowLight);
    glFogf(GL_FOG_START, 100.0f);
    glFogf(GL_FOG_END, 650.0f);
```

```
    glFogi(GL_FOG_MODE, GL_LINEAR);
    break;
  case 1:
    glDisable(GL_FOG);
    glClearColor(fLowLight[0], fLowLight[1], fLowLight[2], fLowLight[3]);
    glFogfv(GL_FOG_COLOR, fLowLight);
    glFogf(GL_FOG_DENSITY, 0.005f);
    glFogi(GL_FOG_MODE, GL_EXP);
    glEnable(GL_FOG);
    break;
  case 2:
    glDisable(GL_FOG);
    glClearColor(fLowLight[0], fLowLight[1], fLowLight[2], fLowLight[3]);
    glEnable(GL_FOG);
    glFogf(GL_FOG_DENSITY, 0.005f);
    glFogfv(GL_FOG_COLOR, fLowLight);
    glFogi(GL_FOG_MODE, GL_EXP2);
    break;
  }
  glutPostRedisplay();
}

void createMenu() {
  glutCreateMenu(changefog);
  glutAddMenuEntry("GL_LINEAR on", 0);
  glutAddMenuEntry("GL_EXP on", 1);
  glutAddMenuEntry("GL_EXP2 on", 2);
  glutAttachMenu(GLUT_RIGHT_BUTTON);
}

int main(int argc, char *argv[]) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Almazova");
  glutReshapeFunc(ChangeSize);
  glutSpecialFunc(SpecialKeys);
  glutDisplayFunc(RenderScene);
  createMenu();
  SetupRC();
  glutMainLoop();
  return 0;
}
```

Результаты

Вывод: были сформированы практически навыки по работе с освещением средствами OpenGL, а также созданию настройки свойств материалов объектов и применения эффектов тени и тумана для большей реалистичности изображений.