МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №7
по дисциплине «Компьютерная графика»
Тема: Реализация трехмерного объекта
с использованием библиотеки OpenGL

Студент гр. 8383 _____ Киреев К.А.

Студент гр. 8383 _____ Муковский Д.В.

Преподаватель _____ Герасимова Т.В.

Санкт-Петербург
2021

Цель работы.

Разработать программу, реализующую представление разработанной вами трехмерной сцены с добавлением возможности формирования различного типа проекций, используя предложенные функции OpenGL и возможности использования различных видов источников света, используя предложенные функции OpenGL (матрицы видового преобразования, проецирование, модель освещения, типы источников света, свойства материалов).

Разработанная программа должна быть пополнена возможностями остановки интерактивно различных атрибутов через вызов соответствующих элементов интерфейса пользователя (замена типа источника света, управление положением камеры, изменение свойств материала модели, как с помощью мыши, так и с помощью диалоговых элементов)

Задания.

Разработать программу, реализующую представление трехмерной сцены с добавлением возможности формирования различного типа проекций, отражений, используя предложенные функции OpenGL (модель освещения, типы источников света, свойства материалов).

Разработанная программа должна быть пополнена возможностями остановки интерактивно различных атрибутов через вызов соответствующих элементов интерфейса пользователя (замена типа источника света, управление положением камеры, изменение свойств материала модели, как с помощью мыши, так и с помощью диалоговых элементов)

Теоретические сведения.

Использование источников света в OpenGL

Для создания реалистических изображений необходимо определить как свойства самого объекта, так и свойства среды, в которой он находится. Первая группа свойств включает в себя параметры материла, из которого сделан объект, способы нанесения текстуры на его поверхность, степень прозрачности объекта.

Ко второй группе можно отнести количество и свойства источников света, уровень прозрачности среды. Все эти свойства можно задавать, используя соответствующие команды *OpenGL*.

В *OpenGL* содержится целый ряд функций для установки и использования источников света, а также для придания поверхности свойств определенного материала. Довольно трудно охватить все их возможные варианты и детали, поэтому рассмотрим лишь основные - как устанавливать на сцене различные виды источников света.

Создание источника света

Добавить в сцену источник света можно с помощью команд

```
void glLight[i f](GLenum light, GLenum pname, GLfloat param);
void glLight[i f](GLenum light, GLenum pname, GLfloat *params).
```

Параметр light однозначно определяет источник, и выбирается из набора специальных символических имен вида *GL_LIGHTi,* где i должно лежать в диапазоне от *0* до *GL_MAX_LIGHT*, которое не превосходит восьми.

Рассмотрим назначение остальных двух параметров (вначале описываются параметры для первой команды, затем для второй) pname:

- *GL_AMBIENT* параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет фонового освещения. Значение по умолчанию: (0.0,0.0,0.0,1.0);
- *GL_DIFFUSE* параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет диффузного освещения. Значение по умолчанию: (1.0,1.0,1.0,1.0) для *LIGHT0* и (0.0,0.0,0.0,1.0) для остальных;
- *GL_SPECULAR* параметр params должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет зеркального отражения. Значение по умолчанию: (1.0,1.0,1.0,1.0) для *LIGHT0* и (0.0,0.0,0.0,1.0) для остальных;
- *GL_POSITION* параметр params должен содержать четыре целых или вещественных, которые определяют положение источника света. Если

значение компоненты w равно 0.0, то источник считается бесконечно удаленным и при расчете освещенности учитывается только направление на точку (x, y, z), в противном случае считается, что источник расположен в точке (x, y, z, w). Значение по умолчанию: (0.0,0.0,1.0,0.0);

Модель освещения в OpenGL

OpenGL предусматривает задание трех параметров, определяющих общие законы применения модели освещения. Эти параметры передаются в функцию *glLightModel()* и некоторые ее модификации. Для задания глобальных параметров освещения используются команды:

```
void glLightModel[i f](GLenum pname, GLenum param)
void glLightModel[i f]v(GLenum pname, const GLtype *params)
```

Аргумент pname определяет, какой параметр модели освещения будет настраиваться и может принимать следующие значения:

- *GL_LIGHT_MODEL_LOCAL_VIEWER* - является ли точка наблюдения локальной или удаленной. *OpenGL* вычисляет зеркальные отражения с помощью "промежуточного вектора" *h=s+v*, описанного ранее. Истинные направления s и v, различаются для каждой вершины сетки. Если источник света является направленным, то вектор s-величина постоянная, а v все же изменяется от вершины к вершине. Скорость визуализации возрастает, если сделать и вектор v постоянным для всех вершин. По умолчанию *OpenGL* использует значение *v=(0,0,1),* при этом вектор указывает в сторону положительной оси z в координатах камеры. В то же время можно принудительно заставить графический конвейер вычислять истинное значение вектора v для каждой вершины с помощью выполнения оператора:

  ```
  GlLightModel(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
  ```

  Параметр *param* должен быть булевским и задает положение наблюдателя. Если он равен *FALSE*, то направление обзора считается параллельным оси z, вне зависимости от положения в видовых координатах. Если же он равен *TRUE*, то наблюдатель находится в начале видовой системы координат. Это

может улучшить качество освещения, но усложняет его расчет. Значение по умолчанию: *FALSE*.;

- *GL_LIGHT_MODEL_TWO_SIDE* - правильно ли происходит закрашивание обеих сто-рон полигона. Параметр param должен быть булевским и управляет режимом расчета освещенности, как для лицевых, так и для обратных граней. Если он равен *FALSE*, то осве-щенность рассчитывается только для лицевых граней. Если же он равен *TRUE*, расчет про-водится и для обратных граней. Значение по умолчанию: *FALSE*. Каждая полигональная грань модели имеет две стороны. При моделировании можно рассматривать внутреннюю сторону и внешнюю. Принято заносить эти вершины в список против часовой стрелки, если смотреть с внешней стороны объекта.

Перемещение источников света

OpenGL обрабатывает положение и направление источника света так же, как он обрабатывает положение геометрического примитива. Другими словами, источник света подвергается тем же матричным преобразованиям, что и примитив. Более точно, при вызове команды *glLight..()* для задания позиции или направления источника света, эти позиции или направление преобразуются с помощью текущей матрицы видового преобразования и хранятся в видовой системе координат. Это означает, что можно управлять положением источника света или его направлением, изменяя содержимое матрицы видового преобразования.

Следовательно, источники света могут быть перемещены с помощью соответствующих вызовов функций преобразований. Функции преобразования - *glRotated(), glTranslated().*

Массив *position*, заданный подпрограммой *glLightfv(GL_LIGHT0, GL_POSITION, position),* изменяется с помощью матрицы моделирования-вида, что происходит в момент вызова функции *glLightfv().*

Работа со свойствами материалов в OpenGL

Влияние источника света можно увидеть только при отражении света от поверхности объекта. В *OpenGL* предусмотрены возможности задания различных коэффициентов отражения, фигурирующих в уравнении интенсивности отраженного света. Эти коэффициенты устанавливаются с помощью различных версий функции *glMaterial(),* причем коэффициенты можно устанавливать отдельно для лицевых и нелицевых граней.

Для задания параметров текущего материала используются команды:

```
void glMaterial[i f](GLenum face, GLenum pname, GLtype param);
void glMaterial[i f]v(GLenum face, GLenum pname, GLtype *params);
```

С их помощью можно определить рассеянный, диффузный и зеркальный цвета материала, а также цвет, степень зеркального отражения и интенсивность излучения света, если объект должен светиться. Какой именно параметр будет определяться значением *param*, зависит от значения *pname*:

- *GL_AMBIENT* параметр *params* должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют рассеянный цвет материала (цвет материала в тени). Значение по умолчанию: (0.2,0.2,0.2,1.0);
- *GL_DIFFUSE* параметр *params* должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет диффузного отражения материала. Значение по умолчанию: (0.8,0.8,0.8,1.0);
- *GL_SPECULAR* параметр *params* должен содержать четыре целых или вещественных значения цветов RGBA, которые определяют цвет зеркального отражения материала. Значение по умолчанию: (0.0,0.0,0.0,1.0);
- *GL_SHININESS* параметр *params* должен содержать одно целое или вещественное значение в диапазоне от 0 до 128, которое определяет степень зеркального отражения материала. Значение по умолчанию: 0;

Выполнение работы.

В результате выполнения лабораторной работы, к возможностям предыдущей программы была добавлена возможность изменять материал модели, менять модель освещения и тип источника света с помощью интерфейса пользователя. Был написан основной метод `setLight()`, который выполняет следующие функции:

- устанавливает параметры источника света
- определяет свойства материала поверхности
- задает источник света
- устанавливает параметры материала
- разрешает освещение и включает источник света

Описание используемых функций в методе `setLight()` представлено в листинге с комментариями ниже.

```
void Widget::setLight() {

    // Материал

    GLfloat material_ambient[] = {0.1f, 0.1f, 0.3f}; // рассеянный цвет материала (цвет материала в тени)
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, material_ambient);

    GLfloat material_diffuse[] = {0.2f, 0.2f, 0.2f}; // цвет диффузного отражения материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material_diffuse);

    GLfloat material_spec[] = {5.0f, 5.0f, 6.0f}; // цвет отраженного света
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, material_spec);

    GLfloat material_shininess[] = {20.0f}; // степень зеркального отражения материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, material_shininess);

    // Свет

    GLfloat ambient[] = {0.2, 0.4, 0.6, 1.0}; // "цвет тени"
    glLightfv(LIGHT_TYPE, GL_AMBIENT, ambient); // цвет фонового освещения

    GLfloat diffuse[] = {0.8, 0.9, 0.5, 1.0}; // "цвет фигуры"
    glLightfv(LIGHT_TYPE, GL_DIFFUSE, diffuse); // цвет диффузного освещения

    GLfloat spec[] = {1.0, 0.8, 1.0, 1.0}; // цвет зеркального отражения
    glLightfv(LIGHT_TYPE, GL_SPECULAR, spec);

    GLfloat position[] = {0.0, 4.0, 4.0, 1.0}; // положение источника света
    glLightfv(LIGHT_TYPE, GL_POSITION, position);

    glLightf(LIGHT_TYPE, GL_CONSTANT_ATTENUATION, 0.0); // постоянная в функции затухания f(d)
    glLightf(LIGHT_TYPE, GL_LINEAR_ATTENUATION, 0.09); // коэффициент при линейном члене в f(d)
    glLightf(LIGHT_TYPE, GL_QUADRATIC_ATTENUATION, 0.0); // коэффициент при квадрате расстояния в f(d)

    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE); // освещенность рассчитывается для лицевых и обратных граней

    glEnable(GL_LIGHTING); // включение света
    glEnable(LIGHT_TYPE); // включение конкретного света
}
```

Тип источника освещения

- параметр *GL_AMBIENT* определяет цвет фонового освещения
- параметр *GL_DIFFUSE* определяет цвет диффузного освещения
- параметр *GL_SPECULAR* определяет цвет зеркального отражения
- параметр *GL_POSITION* определяет положение источника света.

Пример функции изменения типа источника освещения представлен в листинге ниже.

```
void Widget::changeLightPosition2() {
    GLfloat ambient[] = {0.5, 0.2, 0.2, 1.0}; // цвет фонового освещения
    glLightfv(LIGHT_TYPE, GL_AMBIENT, ambient);

    GLfloat diffuse[] = {0.8, 0.9, 0.5, 1.0}; // цвет диффузного освещения
    glLightfv(LIGHT_TYPE, GL_DIFFUSE, diffuse);

    GLfloat spec[] = {1.0, 0.8, 1.0, 1.0}; // цвет зеркального отражения
    glLightfv(LIGHT_TYPE, GL_SPECULAR, spec);

    GLfloat position[] = {3.0, 6.5, 0.0, 1.0}; // положение источника света
    glLightfv(LIGHT_TYPE, GL_POSITION, position);
    update();
}
```

Модель освещения

- *GL_LIGHT_MODEL_LOCAL_VIEWER* - является ли точка наблюдения локальной или удаленной

Пример функции изменения модели освещения представлен в листинге ниже.

```
void Widget::changeModel(){
    flag++;
    if (flag % 2 == 1)
        glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE); // наблюдатель находится в начале видовой системы координат
    else
        glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE); // направление обзора параллельно оси z
    update();
}
```

Материал модели

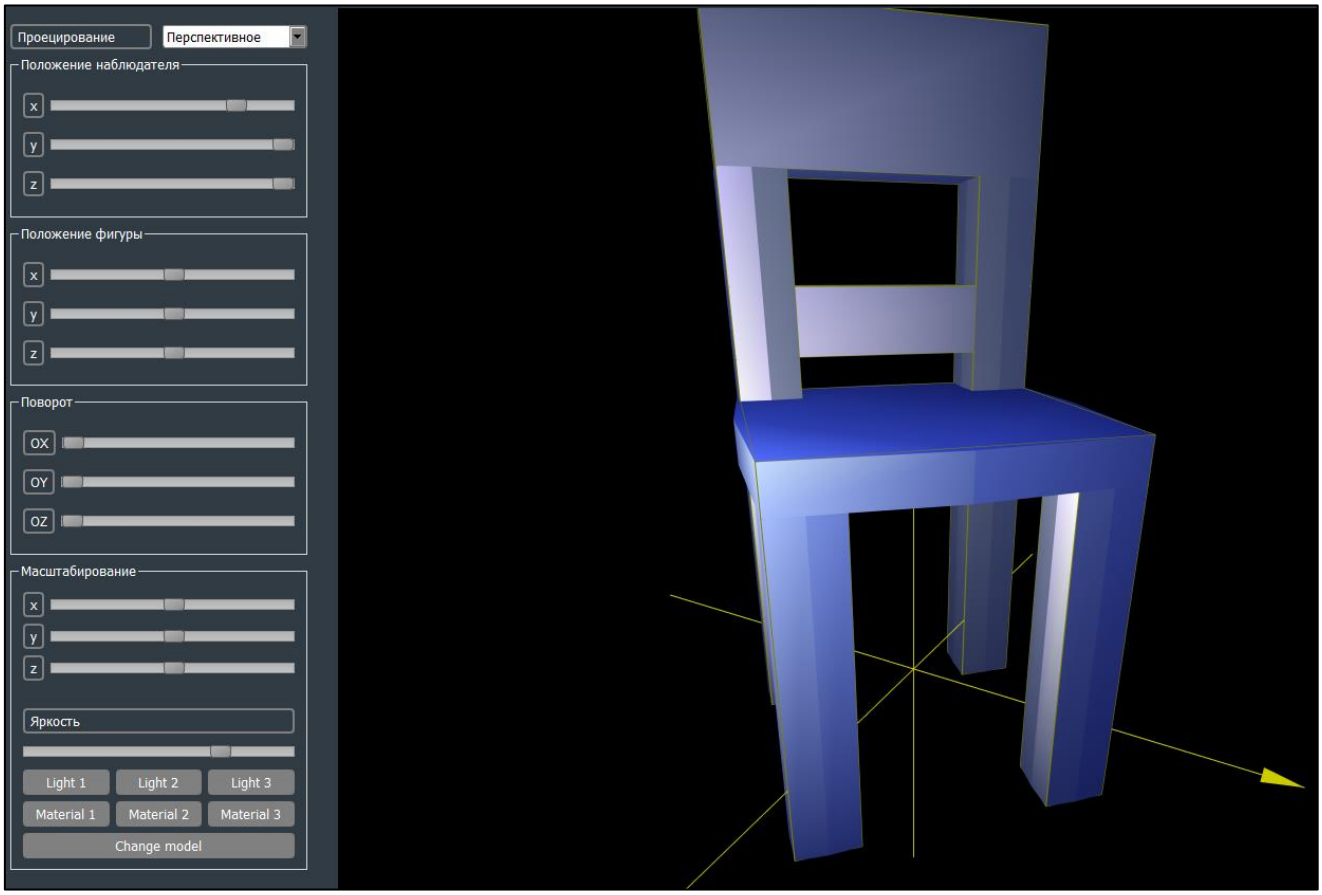
- параметр *GL_AMBIENT* определяет цвет материала в тени
- параметр *GL_DIFFUSE* определяет цвет диффузного отражения материала
- параметр *GL_SPECULAR* определяет цвет отраженного света
- параметр *GL_SHININESS* определяет степень зеркального отражения материала.

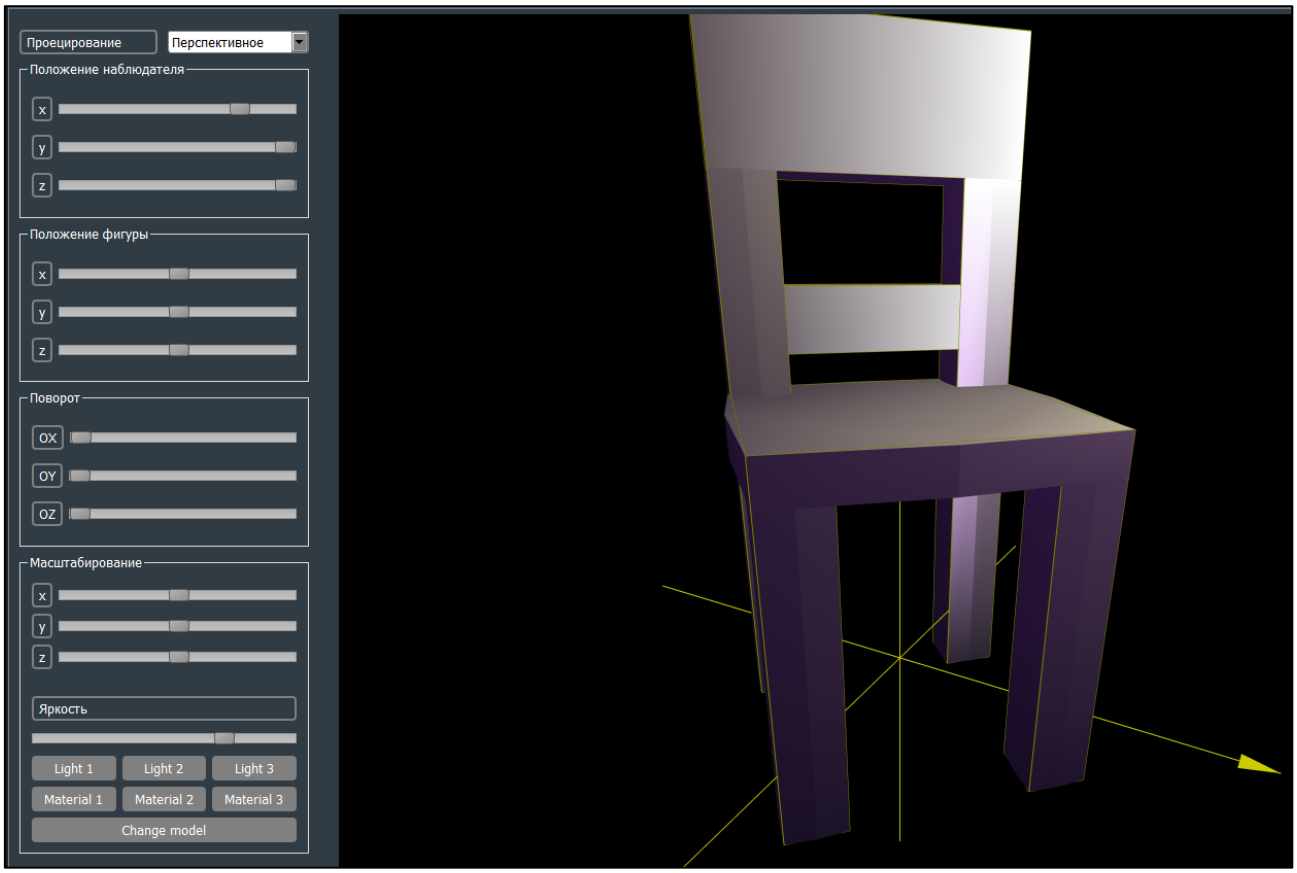
Пример функции изменения свойств материала модели представлен в листинге ниже.

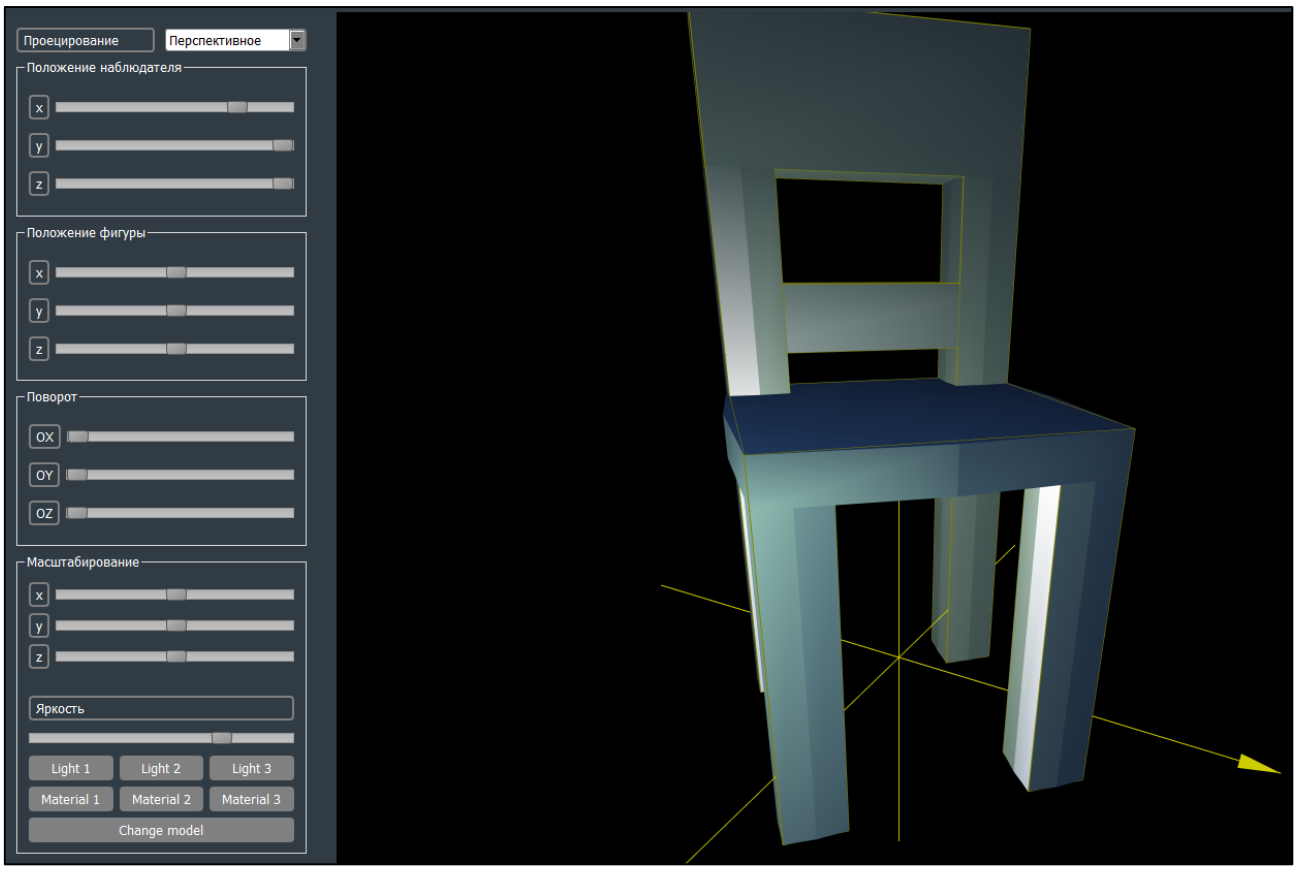
```
void Widget::changeMaterial3(){

    GLfloat material_ambient[] = {0.1f, 0.1f, 0.3f}; // рассеянный цвет материала (цвет материала в тени)
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, material_ambient);

    GLfloat material_diffuse[] = {0.2f, 0.2f, 0.2f}; // цвет диффузного отражения материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material_diffuse);

    GLfloat material_spec[] = {5.0f, 5.0f, 6.0f}; // цвет отраженного света
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, material_spec);

    GLfloat material_shininess[] = {40.0f}; // степень зеркального отражения материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, material_shininess);
    update();
}
```

Пример работы программы

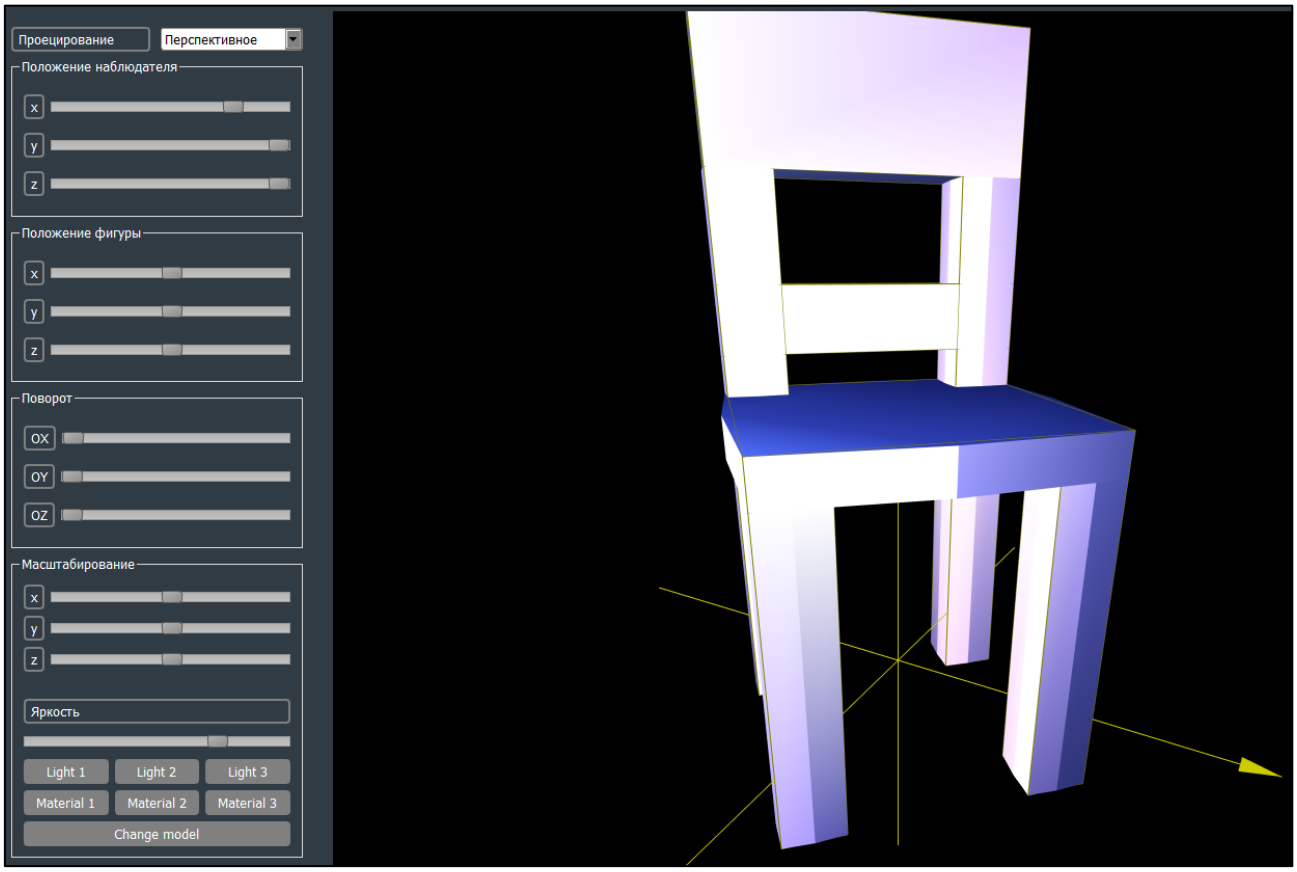
Изменение источника света

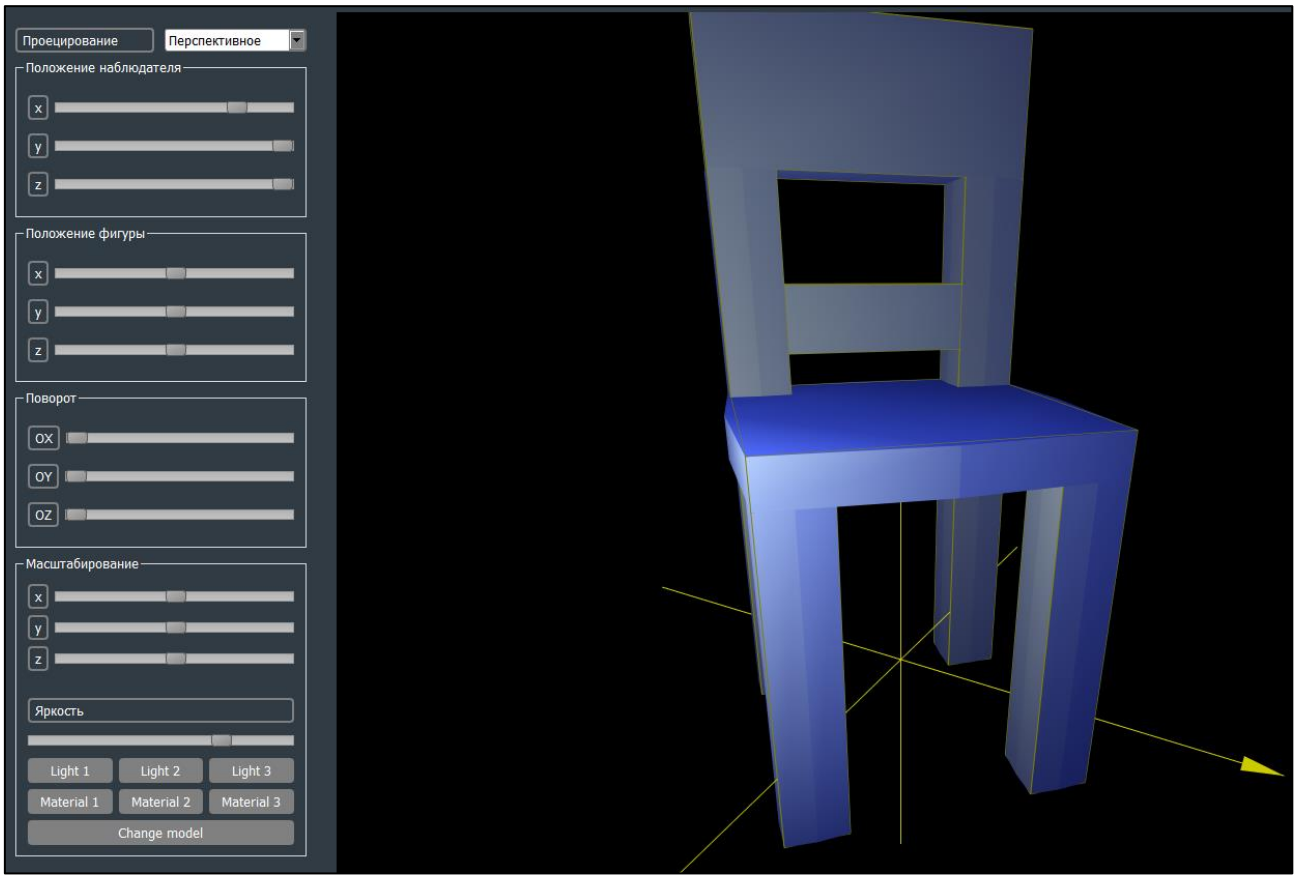



Проецирование
Перспективное
Положение наблюдателя
x
y
z
Положение фигуры
x
y
z
Поворот
OX
OY
OZ
Масштабирование
x
y
z
Яркость
Light 1
Light 2
Light 3
Material 1
Material 2
Material 3
Change model

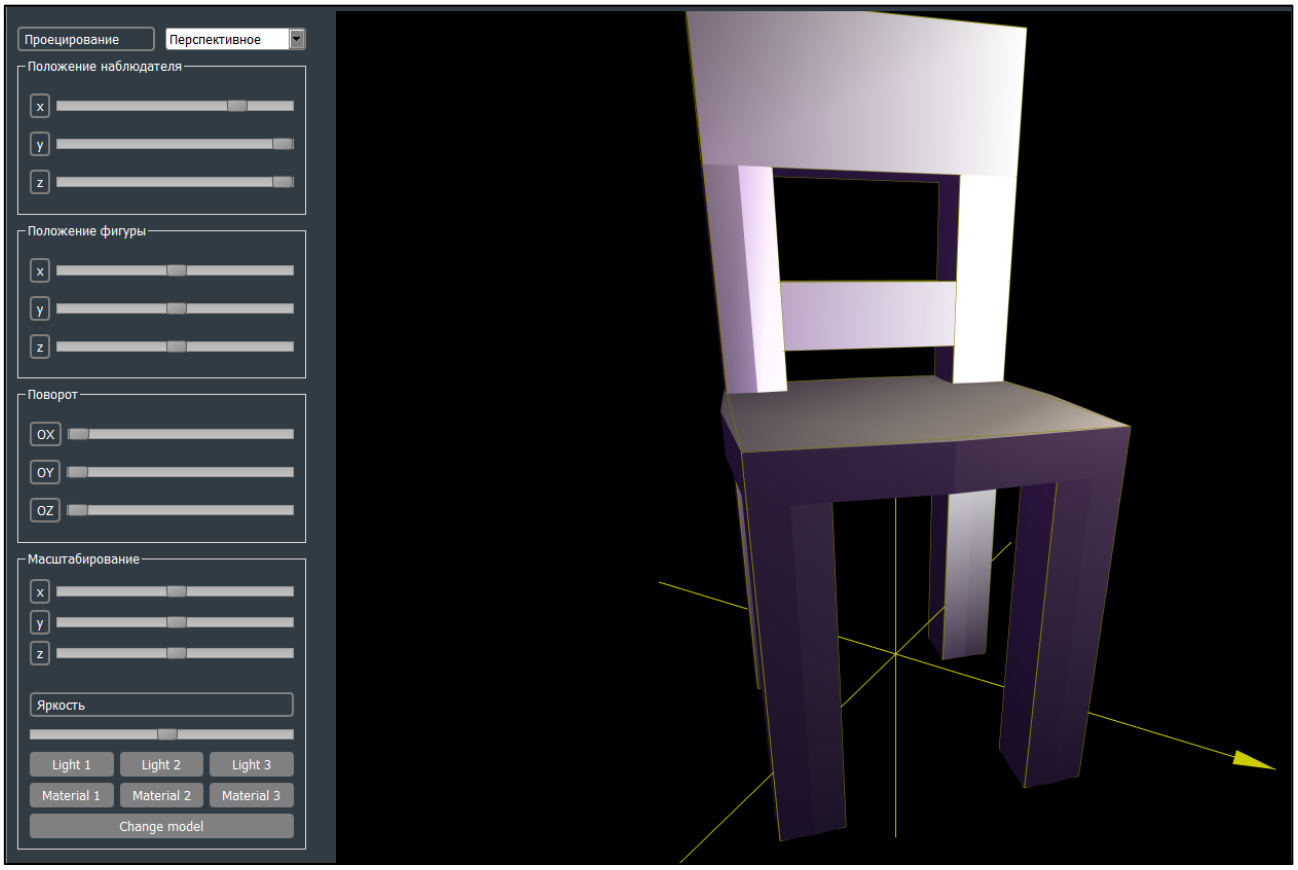

Проецирование
Перспективное
Положение наблюдателя
x
y
z
Положение фигуры
x
y
z
Поворот
OX
OY
OZ
Масштабирование
x
y
z
Яркость
Light 1
Light 2
Light 3
Material 1
Material 2
Material 3
Change model

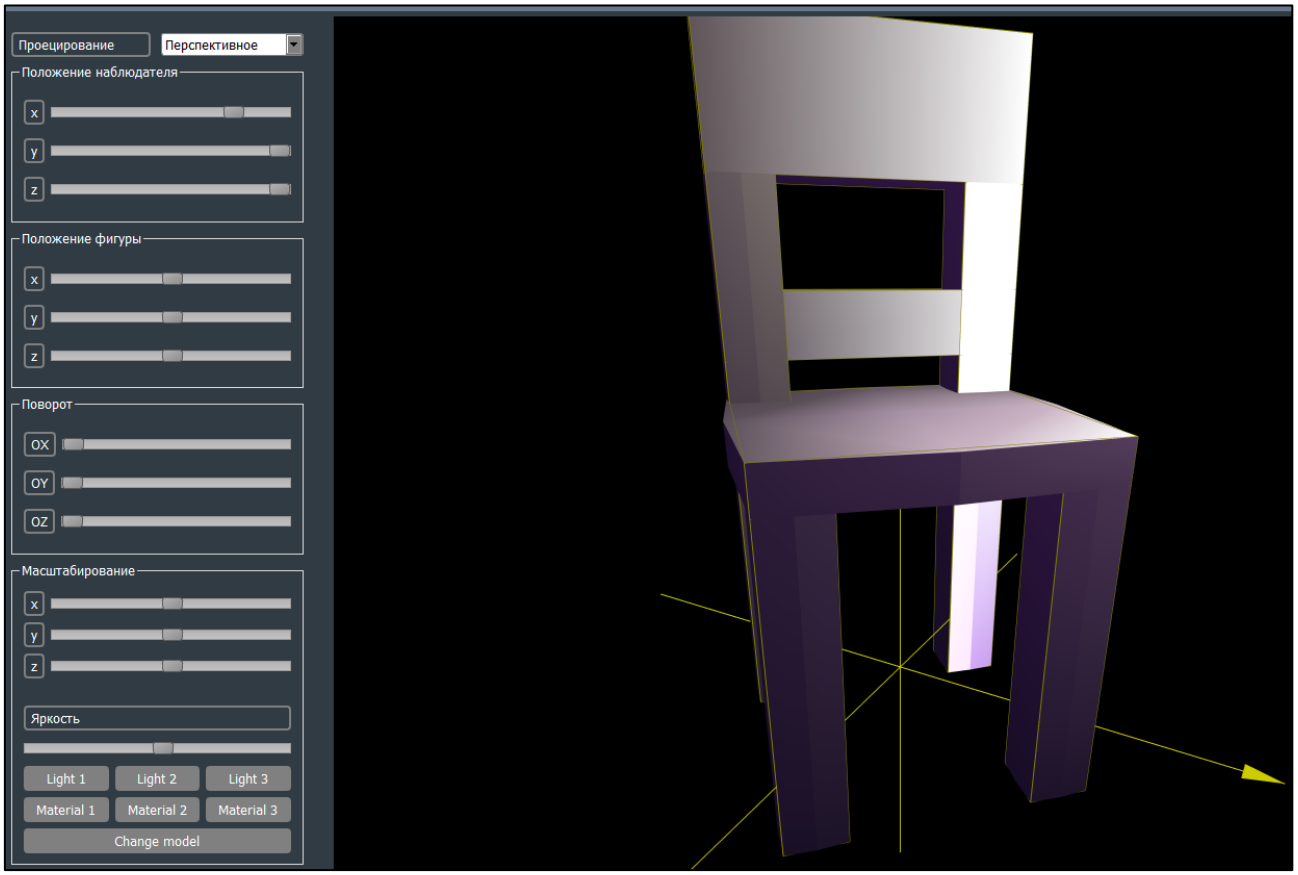
Изменение свойств материала





Изменение модели освещения





Выводы.

Разработана программа, реализующая представление разработанной трехмерной сцены с добавлением возможности использования различных видов источников света, используя предложенные функции OpenGL.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ WIDGET.CPP

```
#include "widget.h"
#include <QDebug>
#include <qmath.h>
#include <QString>
#include <QVector3D>
#include <GL/glu.h>
#include <QDebug>
#include <list>

# define LIGHT_TYPE GL_LIGHT1

Widget::Widget(QWidget *parent = 0) : QGLWidget(parent)
{
    angleX = 0, angleY = 0, angleZ = 0;
    cameraPosition = new QVector3D(3, 4, 5);
    figurePosition = new QVector3D(0, 0, 0);
    figureScale = new QVector3D(1, 1, 1);
    perspective = true;
    showWireframe = true;
    polygons = true;
    recountPoints();
}

void Widget::initializeGL()
{
    qglClearColor(Qt::black);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_NORMALIZE);
    setLight();
    setPerspectiveProjection();
}

void Widget::resizeGL(int nWidth, int nHeight)
{
    glViewport(0, 0, nWidth, nHeight);
}

void Widget::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    drawAxises();//оси координат

    //расположение света
//    glMatrixMode(GL_MODELVIEW);
//    glPushMatrix();
//    glLoadIdentity();
//    GLfloat position[] = {0.0, 4.5, 3.0, 1.0};
//    glLightfv(LIGHT_TYPE, GL_POSITION, position);
//    glPopMatrix();
```

```
    //положение фигуры
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    glRotatef(angleX, 1.0, 0.0, 0.0);
//    qDebug()<<angleX;
    glRotatef(angleY, 0.0, 1.0, 0.0);
    glRotatef(angleZ, 0.0, 0.0, 1.0);
    glTranslatef(figurePosition->x(), figurePosition->y(), figurePosition->z());
    glScalef(figureScale->x(), figureScale->y(), figureScale->z());

    if (polygons)
        drawPolygons();

    if (showWireframe){
        drawWireframe(wireframe);
        drawWireframe(wireframe2);
        drawWireframe(wireframe3);
        drawWireframe(wireframe4);
        drawWireframe(wireframe5);
        drawWireframe(wireframe6);
        drawWireframe(wireframe7);
        drawWireframe(wireframe8);
        drawWireframe(wireframe9);
    }
    glPopMatrix();

}

void Widget::drawWireframe(QList<QVector3D> wireframe) {

    glDisable(GL_LIGHTING);
    glColor3f(0.5, 0.5, 0.0);

    //нижний контур
    glBegin(GL_LINE_LOOP);
    for(int i = 0; i < vertices * 2; i++)
        glVertex3f(wireframe.at(i).x(), wireframe.at(i).y(), wireframe.at(i).z());
    glEnd();

    //верхний контур
    glBegin(GL_LINE_LOOP);
    for(int i = vertices * 2; i < vertices * 4; i++)
        glVertex3f(wireframe.at(i).x(), wireframe.at(i).y(), wireframe.at(i).z());
    glEnd();

    //стороны
    glBegin(GL_LINES);
    for(int i = 0; i < vertices * 2; i++) {
        glVertex3f(wireframe.at(i).x(), wireframe.at(i).y(), wireframe.at(i).z());
        int j = i + vertices * 2;
        glVertex3f(wireframe.at(j).x(), wireframe.at(j).y(), wireframe.at(j).z());
```

```
    }
    glEnd();

    glEnable(GL_LIGHTING);
}



void Widget::drawPolygons() {

    glBegin(GL_TRIANGLES);
    glColor3f(1, 1, 1);

    //верх
    glNormal3f(0, 1, 0);
    for (int i = 0; i < topPoints.size(); i++)
        glVertex3f(topPoints.at(i).x(), topPoints.at(i).y(), topPoints.at(i).z());

    //низ
    for (int i = 0; i < bottomPoints.size(); i++)
        glVertex3f(bottomPoints.at(i).x(), bottomPoints.at(i).y(),
bottomPoints.at(i).z());
    glEnd();

    for (int i = 0; i < shitPoints.size(); i+=4){

        glBegin(GL_POLYGON);
        glNormal3f(shitNormals.at(i/4).x(), shitNormals.at(i/4).y(),
shitNormals.at(i/4).z());
        for (int j = i; j < i + 4; j++) {
            glVertex3f(shitPoints.at(j).x(), shitPoints.at(j).y(),
shitPoints.at(j).z());

        }

        glEnd();
    }

    //стороны
    for (int i = 0; i < sidePoints.size(); i += 4) {
        glBegin(GL_POLYGON);
        glNormal3f(sideNormals.at(i/4).x(), sideNormals.at(i/4).y(),
sideNormals.at(i/4).z());
        for (int j = i; j < i + 4; j++) {
            glVertex3f(sidePoints.at(j).x(), sidePoints.at(j).y(),
sidePoints.at(j).z());
        }
        glEnd();
    }

}

void Widget::recountPoints() {
```

```
    wireframe.clear();
    wireframe2.clear();
    wireframe3.clear();
    wireframe4.clear();
    wireframe5.clear();
    wireframe6.clear();
    wireframe7.clear();
    wireframe8.clear();
    wireframe9.clear();
    //рассчет точек для контура

    for (int j = 0; j < 2; j++) {
            float angle = 0, step = 2 * M_PI / vertices, x, z;
            for (int i = 0; i < vertices; i++) {
                x = starSize * cos(angle);
                z = starSize * sin(angle);
                wireframe.append(QVector3D(x*0.2-1.6, j * starHeight*4+d, z*0.2));
                wireframe.append(QVector3D(x*0.2-1.6, j * starHeight*4+d, z*0.2));
                angle += step;
            }
        }

        for (int j = 0; j < 2; j++) {
            float angle = 0, step = 2 * M_PI / vertices, x, z;
            for (int i = 0; i < vertices; i++) {
                x = starSize * cos(angle);
                z = starSize * sin(angle);
                wireframe2.append(QVector3D(x, j * starHeight*0.5+4+d, z));
                wireframe2.append(QVector3D(x, j * starHeight*0.5+4+d, z));
                angle += step;
            }
        }


        for (int j = 0; j < 2; j++) {
                float angle = 0, step = 2 * M_PI / vertices, x, z;
                for (int i = 0; i < vertices; i++) {
                    x = starSize * cos(angle);
                    z = starSize * sin(angle);
                    wireframe6.append(QVector3D(x*0.2-1.6, j * starHeight*3+4+d,
z*0.2));
                    wireframe6.append(QVector3D(x*0.2-1.6, j * starHeight*3+4+d,
z*0.2));
                    angle += step;
                }
            }

        for (int j = 0; j < 2; j++) {
                float angle = 0, step = 2 * M_PI / vertices, x, z;
                for (int i = 0; i < vertices; i++) {
                    x = starSize * cos(angle);
                    z = starSize * sin(angle);
```

```
                    wireframe7.append(QVector3D(x*0.2, j * starHeight*3+4+d, z*0.2-
1.6));
                    wireframe7.append(QVector3D(x*0.2, j * starHeight*3+4+d, z*0.2-
1.6));
                    angle += step;
                }
            }


{

        wireframe8.append(QVector3D(0.4, 7+d,-1.6));
        wireframe8.append(QVector3D(0.4, 7+d,-1.6));

        wireframe8.append(QVector3D(-1.6, 7+d,0.4));
        wireframe8.append(QVector3D(-1.6, 7+d,0.4));


        wireframe8.append(QVector3D(-2, 7+d,0));
        wireframe8.append(QVector3D(-2, 7+d,0));


        wireframe8.append(QVector3D(0, 7+d,-2));
        wireframe8.append(QVector3D(0, 7+d,-2));

        wireframe8.append(QVector3D(0.4, 8.5+d,-1.6));
        wireframe8.append(QVector3D(0.4, 8.5+d,-1.6));

        wireframe8.append(QVector3D(-1.6, 8.5+d,0.4));
        wireframe8.append(QVector3D(-1.6, 8.5+d,0.4));


        wireframe8.append(QVector3D(-2, 8.5+d,0));
        wireframe8.append(QVector3D(-2, 8.5+d,0));


        wireframe8.append(QVector3D(0, 8.5+d,-2));
        wireframe8.append(QVector3D(0, 8.5+d,-2));

}




        {

                wireframe9.append(QVector3D(0.4, 5+d,-1.6));
                wireframe9.append(QVector3D(0.4, 5+d,-1.6));

                wireframe9.append(QVector3D(-1.6, 5+d,0.4));
                wireframe9.append(QVector3D(-1.6, 5+d,0.4));
```

```
        wireframe9.append(QVector3D(-2, 5+d,0));
        wireframe9.append(QVector3D(-2, 5+d,0));


        wireframe9.append(QVector3D(0, 5+d,-2));
        wireframe9.append(QVector3D(0, 5+d,-2));

        wireframe9.append(QVector3D(0.4, 5.8+d,-1.6));
        wireframe9.append(QVector3D(0.4, 5.8+d,-1.6));

        wireframe9.append(QVector3D(-1.6, 5.8+d,0.4));
        wireframe9.append(QVector3D(-1.6, 5.8+d,0.4));


        wireframe9.append(QVector3D(-2, 5.8+d,0));
        wireframe9.append(QVector3D(-2, 5.8+d,0));


        wireframe9.append(QVector3D(0, 5.8+d,-2));
        wireframe9.append(QVector3D(0, 5.8+d,-2));

}
for (int j = 0; j < 2; j++) {
        float angle = 0, step = 2 * M_PI / vertices, x, z;

        for (int i = 0; i < vertices; i++) {
            x = starSize * cos(angle);
            z = starSize * sin(angle);
            wireframe3.append(QVector3D(x*0.2+1.6, j * starHeight*4+d, z*0.2));
            wireframe3.append(QVector3D(x*0.2+1.6, j * starHeight*4+d, z*0.2));
            angle += step;
        }
    }

    for (int j = 0; j < 2; j++) {
        float angle = 0, step = 2 * M_PI / vertices, x, z;
        for (int i = 0; i < vertices; i++) {
            x = starSize * cos(angle);
            z = starSize * sin(angle);
            wireframe4.append(QVector3D(x*0.2, j * starHeight*4+d, z*0.2+1.6));
            wireframe4.append(QVector3D(x*0.2, j * starHeight*4+d, z*0.2+1.6));
            angle += step;
        }
    }

    for (int j = 0; j < 2; j++) {
        float angle = 0, step = 2 * M_PI / vertices, x, z;
        for (int i = 0; i < vertices; i++) {
            x = starSize * cos(angle);
            z = starSize * sin(angle);
            wireframe5.append(QVector3D(x*0.2, j * starHeight*4+d, z*0.2-1.6));
            wireframe5.append(QVector3D(x*0.2, j * starHeight*4+d, z*0.2-1.6));
```

```
                angle += step;
            }
        }
//для полигонов
topPoints.clear();
sidePoints.clear();
sideNormals.clear();
shitPoints.clear();
shitNormals.clear();
float step = 2 * M_PI / vertices, angle = step/2, x, z, r = 0;
float koefs[7][6] = {{0.2, -1.6, 4, d, 0.2, 0}, {1, 0, 0.5, 4+d, 1, 0},
                        {0.2, -1.6, 3, 4+d, 0.2, 0}, {0.2, 0, 3, 4+d, 0.2, -1.6},
                        {0.2, 1.6, 4, d, 0.2, 0}, {0.2, 0, 4, d, 0.2, 1.6},
                        {0.2, 0, 4, d, 0.2, -1.6}};


for (int i = 0; i<2;i++){
    float difH = 0;
    float difL = 0;
    if (i == 1){
        difL = -2;
        difH = -2.7;
    }


    //down
    shitPoints.append(QVector3D(0.4, 7+d+difL,-1.6));
    shitPoints.append(QVector3D(-1.6, 7+d+difL,0.4));
    shitPoints.append(QVector3D(-2, 7+d+difL,0));
    shitPoints.append(QVector3D(0, 7+d+difL,-2));

    shitNormals.append(QVector3D(0,-1,0));

    //top
    shitPoints.append(QVector3D(-1.6, 8.5+d+difH,0.4));
    shitPoints.append(QVector3D(0.4, 8.5+d+difH,-1.6));
    shitPoints.append(QVector3D(0, 8.5+d+difH,-2));
    shitPoints.append(QVector3D(-2, 8.5+d+difH,0));

    shitNormals.append(QVector3D(0,1,0));


    //боковая от нас
    shitPoints.append(QVector3D(-2, 8.5+d+difH,0));
    shitPoints.append(QVector3D(0, 8.5+d+difH,-2));
    shitPoints.append(QVector3D(0, 7+d+difL,-2));
    shitPoints.append(QVector3D(-2, 7+d+difL,0));

    shitNormals.append(QVector3D(-1, 0, -1));

    //боковая на нас
    shitPoints.append(QVector3D(0.4, 8.5+d+difH,-1.6));
    shitPoints.append(QVector3D(-1.6, 8.5+d+difH,0.4));
```

```
        shitPoints.append(QVector3D(-1.6, 7+d+difL,0.4));
        shitPoints.append(QVector3D(0.4, 7+d+difL,-1.6));

        shitNormals.append(QVector3D(1, 0, 1));

        //боковая справа
        shitPoints.append(QVector3D(0.4, 7+d+difL,-1.6));
        shitPoints.append(QVector3D(0, 7+d+difL,-2));
        shitPoints.append(QVector3D(0, 8.5+d+difH,-2));
        shitPoints.append(QVector3D(0.4, 8.5+d+difH,-1.6));

        shitNormals.append(QVector3D(1, 0, -1));


        //боковая слева
        shitPoints.append(QVector3D(-1.6, 8.5+d+difH,0.4));
        shitPoints.append(QVector3D(-2, 8.5+d+difH,0));
        shitPoints.append(QVector3D(-2, 7+d+difL,0));
        shitPoints.append(QVector3D(-1.6, 7+d+difL,0.4));

        shitNormals.append(QVector3D(-1, 0, 1));


    }
        for (int ko = 0; ko < 7; ko++){
            for (int i = 0; i < vertices; i++) {

                float x1 = ((starSize - endSize) *
cos(angle))*koefs[ko][0]+koefs[ko][1];
                float z1 = (starSize - endSize) * sin(angle)*koefs[ko][4]+koefs[ko][5];
                float x2 = ((starSize - endSize) * cos(angle +
step))*koefs[ko][0]+koefs[ko][1];
                float z2 = (starSize - endSize) * sin(angle +
step)*koefs[ko][4]+koefs[ko][5];
                float x3 = ((starSize) * cos(angle +
step/2))*koefs[ko][0]+koefs[ko][1];
                float z3 = (starSize) * sin(angle + step/2)*koefs[ko][4]+koefs[ko][5];

                topPoints.append(QVector3D(0+koefs[ko][1],
starHeight*koefs[ko][2]+koefs[ko][3], 0+koefs[ko][5]));
                topPoints.append(QVector3D(x2, starHeight*koefs[ko][2]+koefs[ko][3],
z2));
                topPoints.append(QVector3D(x1, starHeight*koefs[ko][2]+koefs[ko][3],
z1));
                topPoints.append(QVector3D(x3, starHeight*koefs[ko][2]+koefs[ko][3],
z3));
                topPoints.append(QVector3D(x1, starHeight*koefs[ko][2]+koefs[ko][3],
z1));
                topPoints.append(QVector3D(x2, starHeight*koefs[ko][2]+koefs[ko][3],
z2));

                sidePoints.append(QVector3D(x3, 0*koefs[ko][2]+koefs[ko][3], z3));
                sidePoints.append(QVector3D(x1, 0*koefs[ko][2]+koefs[ko][3], z1));
```

```
                sidePoints.append(QVector3D(x1, starHeight*koefs[ko][2]+koefs[ko][3],
z1));
                sidePoints.append(QVector3D(x3, starHeight*koefs[ko][2]+koefs[ko][3],
z3));

                sidePoints.append(QVector3D(x3, starHeight*koefs[ko][2]+koefs[ko][3],
z3));
                sidePoints.append(QVector3D(x2, starHeight*koefs[ko][2]+koefs[ko][3],
z2));
                sidePoints.append(QVector3D(x2, 0*koefs[ko][2]+koefs[ko][3], z2));
                sidePoints.append(QVector3D(x3, 0*koefs[ko][2]+koefs[ko][3], z3));

                sideNormals.append(vectorProduct(x1 - x3, 0, z1 - z3, 0, 1, 0));
                sideNormals.append(vectorProduct(x3 - x2, 0, z3 - z2, 0, 1, 0));

                angle += step;
            }

            bottomPoints = topPoints;
            for (int i = 0; i < bottomPoints.size(); i++)
                bottomPoints[i].setY(starHeight*koefs[ko][2]+koefs[ko][3]);
        }



}

QVector3D Widget::vectorProduct(float ax, float ay, float az, float bx, float by, float
bz) {
    return QVector3D(ay * bz - az * by, az * bx - ax * bz, ax * by - ay * bx);
}

void Widget::setLight() {

    // Материал

    GLfloat material_ambient[] = {0.1f, 0.1f, 0.3f}; // рассеянный цвет материала (цвет
материала в тени)
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, material_ambient);

    GLfloat material_diffuse[] = {0.2f, 0.2f, 0.2f}; // цвет диффузного отражения
материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material_diffuse);

    GLfloat material_spec[] = {5.0f, 5.0f, 6.0f}; // цвет отраженного света
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, material_spec);

    GLfloat material_shininess[] = {20.0f}; // степень зеркального отражения материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, material_shininess);


    // Свет
```

```
    GLfloat ambient[] = {0.2, 0.4, 0.6, 1.0}; // "цвет тени"
    glLightfv(LIGHT_TYPE, GL_AMBIENT, ambient); // цвет фонового освещения

    GLfloat diffuse[] = {0.8, 0.9, 0.5, 1.0}; // "цвет фигуры"
    glLightfv(LIGHT_TYPE, GL_DIFFUSE, diffuse); // цвет диффузного освещения

    GLfloat spec[] = {1.0, 0.8, 1.0, 1.0}; // цвет зеркального отражения
    glLightfv(LIGHT_TYPE, GL_SPECULAR, spec);

    GLfloat position[] = {0.0, 4.0, 4.0, 1.0}; // положение источника света
    glLightfv(LIGHT_TYPE, GL_POSITION, position);

    glLightf(LIGHT_TYPE, GL_CONSTANT_ATTENUATION, 0.0); // постоянная в функции
затухания f(d)
    glLightf(LIGHT_TYPE, GL_LINEAR_ATTENUATION, 0.09); // коэффициент при линейном
члене в f(d)
    glLightf(LIGHT_TYPE, GL_QUADRATIC_ATTENUATION, 0.0); // коэффициент при квадрате
расстояния в f(d)

    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE); // освещенность рассчитывается для
лицевых и обратных граней

    glEnable(GL_LIGHTING); // включение света
    glEnable(LIGHT_TYPE); // включение конкретного света
}

void Widget::drawAxises() {

    glDisable(GL_LIGHTING);
    //x координата
//    qglColor(Qt::blue);
    glColor3f(0.8, 0.8, 0.0);
    glBegin(GL_LINES);
    glVertex3f(-axisSize, 0.0, 0.0);
    glVertex3f(axisSize, 0.0, 0.0);
    glEnd();
    glBegin(GL_TRIANGLES);
    glVertex3f(axisSize, 0.0, 0.0);
    glVertex3f(axisSize - arrowSize*3, arrowSize, 0.0);
    glVertex3f(axisSize - arrowSize*3, -arrowSize, 0.0);
    glEnd();
    //y координата
//    qglColor(Qt::red);
    glColor3f(0.8, 0.8, 0.0);
    glBegin(GL_LINES);
    glVertex3f(0.0, -axisSize, 0.0);
    glVertex3f(0.0, axisSize, 0.0);
    glEnd();
    glBegin(GL_TRIANGLES);
    glVertex3f(0.0, axisSize, 0.0);
    glVertex3f(-arrowSize, axisSize - arrowSize*3, 0.0);
    glVertex3f(arrowSize, axisSize - arrowSize*3, 0.0);
    glEnd();
```

```
    //z координата
//    qglColor(Qt::green);
    glColor3f(0.8, 0.8, 0.0);
    glBegin(GL_LINES);
    glVertex3f(0.0, 0.0, -axisSize);
    glVertex3f(0.0, 0.0, axisSize);
    glEnd();
    glBegin(GL_TRIANGLES);
    glVertex3f(0.0, 0.0, axisSize);
    glVertex3f(-arrowSize, 0.0, axisSize - arrowSize*3);
    glVertex3f(arrowSize, 0.0, axisSize - arrowSize*3);
    glEnd();

    glEnable(GL_LIGHTING);
}

void Widget::setPerspectiveProjection() {


    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(80.0, 1, 1, 100);
    gluLookAt(cameraPosition->x(), cameraPosition->y(), cameraPosition->z(), 0, 3.0, 0,
0, 1, 0);
//    qDebug()<<cameraPosition->x();
//    qDebug()<<cameraPosition->y();
//    qDebug()<<cameraPosition->z();

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    update();
}

void Widget::setOrthoProjection() {

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-8, 8, -8, 8, 0.5, 100);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(-cameraPosition->x(), -cameraPosition->y(), -cameraPosition->z());

    update();
}

void Widget::changeCameraPositionX(float x) {
    cameraPosition->setX(x);
    if (perspective) setPerspectiveProjection();
    else setOrthoProjection();
}
```

```
void Widget::changeCameraPositionY(float y) {
    cameraPosition->setY(y);
    if (perspective) setPerspectiveProjection();
    else setOrthoProjection();
}

void Widget::changeCameraPositionZ(float z) {
    cameraPosition->setZ(z);
    if (perspective) setPerspectiveProjection();
    else setOrthoProjection();
}

void Widget::changeFigurePositionX(float x) {
    figurePosition->setX(x);
    if (perspective) setPerspectiveProjection();
    else setOrthoProjection();
}

void Widget::changeFigurePositionY(float y) {
    figurePosition->setY(y);
    if (perspective) setPerspectiveProjection();
    else setOrthoProjection();
}

void Widget::changeFigurePositionZ(float z) {
    figurePosition->setZ(z);
    if (perspective) setPerspectiveProjection();
    else setOrthoProjection();
}

void Widget::changeFigureScaleX(float x) {
    figureScale->setX(x);
    if (perspective) setPerspectiveProjection();
    else setOrthoProjection();
}

void Widget::changeFigureScaleY(float y) {
    figureScale->setY(y);
    if (perspective) setPerspectiveProjection();
    else setOrthoProjection();
}

void Widget::changeFigureScaleZ(float z) {
    figureScale->setZ(z);
    if (perspective) setPerspectiveProjection();
    else setOrthoProjection();
}

void Widget::changeBrightness(float b) {
    GLfloat d[] = {b, b, 0.0, 1.0};
    glLightfv(LIGHT_TYPE, GL_DIFFUSE, d);
    update();
}
```

```
void Widget::setShowWireframe(bool s) {
    showWireframe = s;
    update();
}

void Widget::setShowPolygons(bool s) {
    polygons = s;
    update();
}

void Widget::changeLightPosition1() {

    GLfloat ambient[] = {0.2, 0.4, 0.6, 1.0}; // "цвет тени"
    glLightfv(LIGHT_TYPE, GL_AMBIENT, ambient); // цвет фонового освещения

    GLfloat diffuse[] = {0.8, 0.9, 0.5, 1.0}; // "цвет фигуры"
    glLightfv(LIGHT_TYPE, GL_DIFFUSE, diffuse); // цвет диффузного освещения

    GLfloat spec[] = {1.0, 0.8, 1.0, 1.0}; // цвет зеркального отражения
    glLightfv(LIGHT_TYPE, GL_SPECULAR, spec);

    GLfloat position[] = {0.0, 4.0, 4.0, 1.0}; // положение источника света
    glLightfv(LIGHT_TYPE, GL_POSITION, position);

    update();
}

void Widget::changeLightPosition2() {

    GLfloat ambient[] = {0.5, 0.2, 0.2, 1.0}; // "цвет тени"
    glLightfv(LIGHT_TYPE, GL_AMBIENT, ambient); // цвет фонового освещения

    GLfloat diffuse[] = {0.8, 0.9, 0.5, 1.0}; // "цвет фигуры"
    glLightfv(LIGHT_TYPE, GL_DIFFUSE, diffuse); // цвет диффузного освещения

    GLfloat spec[] = {1.0, 0.8, 1.0, 1.0}; // цвет зеркального отражения
    glLightfv(LIGHT_TYPE, GL_SPECULAR, spec);

    GLfloat position[] = {3.0, 6.5, 0.0, 1.0}; // положение источника света
    glLightfv(LIGHT_TYPE, GL_POSITION, position);

    update();
}

void Widget::changeLightPosition3() {

    GLfloat ambient[] = {0.2, 0.4, 0.2 , 1.0}; // "цвет тени"
    glLightfv(LIGHT_TYPE, GL_AMBIENT, ambient); // цвет фонового освещения

    GLfloat diffuse[] = {0.8, 0.9, 0.5, 1.0}; // "цвет фигуры"
    glLightfv(LIGHT_TYPE, GL_DIFFUSE, diffuse); // цвет диффузного освещения
```

```
    GLfloat spec[] = {1.0, 0.8, 1.0, 1.0}; // цвет зеркального отражения
    glLightfv(LIGHT_TYPE, GL_SPECULAR, spec);

    GLfloat position[] = {0.0, 1.5, 3.0, 1.0}; // положение источника света
    glLightfv(LIGHT_TYPE, GL_POSITION, position);

    update();
}

void Widget::changeMaterial1(){

    GLfloat material_ambient[] = {0.1f, 0.1f, 0.3f}; // рассеянный цвет материала (цвет
материала в тени)
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, material_ambient);

    GLfloat material_diffuse[] = {0.2f, 0.2f, 0.2f}; // цвет диффузного отражения
материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material_diffuse);

    GLfloat material_spec[] = {5.0f, 5.0f, 6.0f}; // цвет отраженного света
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, material_spec);

    GLfloat material_shininess[] = {20.0f}; // степень зеркального отражения материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, material_shininess);

    update();
}

void Widget::changeMaterial2(){

    GLfloat material_ambient[] = {0.1f, 0.1f, 0.3f}; // рассеянный цвет материала (цвет
материала в тени)
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, material_ambient);

    GLfloat material_diffuse[] = {0.2f, 0.2f, 0.2f}; // цвет диффузного отражения
материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material_diffuse);

    GLfloat material_spec[] = {5.0f, 5.0f, 6.0f}; // цвет отраженного света
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, material_spec);

    GLfloat material_shininess[] = {6.0f}; // степень зеркального отражения материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, material_shininess);

    update();
}

void Widget::changeMaterial3(){

    GLfloat material_ambient[] = {0.1f, 0.1f, 0.3f}; // рассеянный цвет материала (цвет
материала в тени)
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, material_ambient);
```

```
    GLfloat material_diffuse[] = {0.2f, 0.2f, 0.2f}; // цвет диффузного отражения
материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material_diffuse);

    GLfloat material_spec[] = {5.0f, 5.0f, 6.0f}; // цвет отраженного света
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, material_spec);

    GLfloat material_shininess[] = {40.0f}; // степень зеркального отражения материала
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, material_shininess);

    update();
}

void Widget::changeModel(){
    flag++;
    if (flag % 2 == 1)
        glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE); // наблюдатель находится в
начале видовой системы координат
    else
        glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE); // направление обзора
параллельно оси z
    update();
}
```