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

ОТЧЕТ

по лабораторной работе №6
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 разработанного вами трехмерного рисунка, используя предложенные функции библиотеки OpenGL (матрицы видового преобразования, проецирование) и язык GLSL.

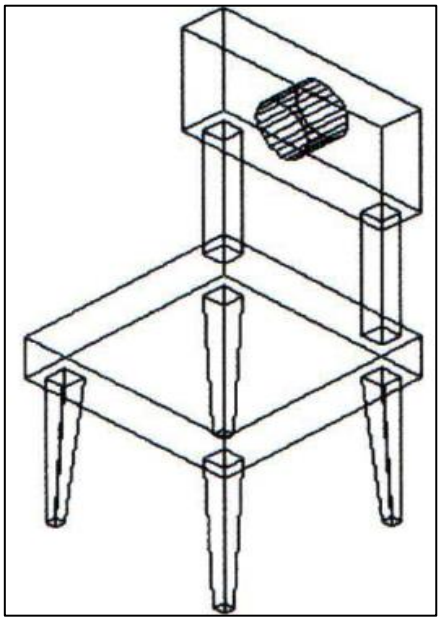
Разработанная программа должна быть пополнена возможностями остановки интерактивно различных атрибутов через вызов соответствующих элементов интерфейса пользователя, замена типа проекции, управление преобразованиями, как с помощью мыши, так и с помощью диалоговых элементов.

Написать программу, рисующую проекцию трехмерного каркасного объекта.

Требования.

- Грани объекта рисуются с помощью доступных функций рисования отрезка в координатах окна. При этом использовать шейдеры *GLSL* и *OpenGL*
- Вывод многогранника с удалением или прорисовкой невидимых граней;
- Ортогональное и перспективное проецирование;
- Перемещения, повороты и масштабирование многогранника по каждой из осей независимо от остальных.
- Генерация многогранника с заданной мелкостью разбиения.
- Д.б. установлено изменение свойств источника света (интенсивность).
- При запуске программы объект сразу должно быть хорошо виден.
- Пользователь имеет возможность вращать фигуру (2 степени свободы) и изменять параметры фигуры.
- Возможно изменять положение наблюдателя.
- Нарисовать оси системы координат.
- Все варианты требований могут быть выбраны интерактивно.

Вариант 30.



Теоретические сведения.

Основная задача, которую необходимо решить при выводе трехмерной графической информации, заключается в том, что объекты, описанные в мировых координатах, необходимо изобразить на плоской области вывода экрана, т.е. требуется преобразовать координаты точки из мировых координат (x,y,z) в оконные координаты (X,Y) ее центральной проекции. Это отображение выполняют в несколько этапов.

Первый этап – *видовое преобразование* – преобразование мировых координат в видовые (видовая матрица);

второй этап – *перспективное преобразование* – преобразование видовых координат в усеченные (матрица проекции).

Для того, чтобы активизировать какую-либо матрицу, надо установить текущий режим матрицы, для чего служит команда :

void glMatrixMode (GLenum mode).

Параметр mode определяет, с каким набором матриц будет выполняться последовательность операций, и может принимать одно из трех значений: *GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE*. Для определения элементов матрицы используются следующие команды: *glLoadMatrix, glLoadIdentity*.

Преобразование объектов выполняется при помощи следующих операций над матрицами.

void glRotate[f d](GLdouble angle, GLdouble x, GLdouble y, GLdouble z)

Эта команда рассчитывает матрицу для выполнения вращения вектора против часовой стрелки на угол, определяемый параметром angle, осуществляемого относительно точки (x,y,z). После выполнения этой команды все объекты изображаются повернутыми.

void glTranslate[f d](GLdouble x, GLdouble y, GLdouble z);

При помощи этой команды осуществляется перенос объекта на расстояние x по оси X, на расстояние y по оси У и на z по оси Z.

Проекции.

Несоответствие между пространственными объектами и плоским изображением устраняется путем введения проекций, которые отображают объекты на двумерной проекционной картинной плоскости. Очень важное значение имеет расстояние между наблюдателем и объектом, поскольку "эффект перспективы" обратно пропорционален этому расстоянию. Таким образом, вид проекции зависит от расстояния между наблюдателем и картинной плоскостью, в зависимости от которой различают два основных класса проекций: *параллельные* и *центральные*. В работе использовалась центральная проекция, а именно перспективная проекция. Для задания проекции использовалась команда *gluPerspective(Gldouble angley,Gldouble aspect,Gldouble znear,Gldouble zfar).*

Предполагается, что точка схода имеет координаты (0, 0, 0) в видовой системе координат. Параметр angley задает угол видимости (в градусах) в направлении оси y. В направлении оси x угол видимости задается через отношение сторон aspect, которое обычно определяется отношением сторон области вывода. Два других параметра задают расстояние от наблюдателя (точки схода) до ближней и дальней плоскости отсечения. Ориентация объекта задана при помощи команды

gluLookAp(Gldouble eyex,Gldouble eyey,Gldouble eyez,Gldouble eyex,Gldouble centerx,Gldouble centery,Gldouble centerz,Gldouble upx,Gldouble upy,Gldouble upz)

точка наблюдения задается группой параметров eye, центр сцены – center, верх сцены – up.

Выполнение работы.

Фигура

Сама фигура рисуется при помощи того, что мы рисуем несколько прямоугольников разных размеров и попарно соединяем соответствующие точки между этими прямоугольниками:

```
void Widget::drawframe(QList <QVector3D> wireframe) {
    glDisable(GL_LIGHTING);
    glColor3f(0.5, 0.5, 0.0);
    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();
    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();
    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);
}
```

Пример расчёта точек для контура “ножки” стула:

```
void Widget::recountPoints() {
    wireframe.clear();
    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, z*0.2));
            wireframe.append(QVector3D(x*0.2-1.6, j * starHeight*4, z*0.2));
            angle += step;
        }
    }
}
```

Поворот и масштабирование

Данные преобразования объекта расположены в методе *paintGL()*, который занимается отрисовкой. Мы заносим в стек матрицу преобразований, затем поворачиваем или масштабируем нашу фигуру по всем осям, в зависимости от выбранных пользователем параметров. После чего рисуем сам объект, затем достаем из стека матрицу преобразований в том виде, в котором она была до поворотов и масштабирования. Данные преобразования объекта происходят следующим образом:

```
void Widget::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    drawAxises();
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    GLfloat position[] = {0.0, 2.5, 3.0, 1.0};
    glLightfv(GL_LIGHT1, GL_POSITION, position);
    glPopMatrix();
    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);
        drawWireframe(wireframe10);
    }
    glPopMatrix();

}
```

Положение камеры

Имеется три метода для изменений положения камеры для каждой из осей, а также имеются переменные позиции камеры в трехмерном пространстве, которые мы заменяем в этих методах на новые значения.

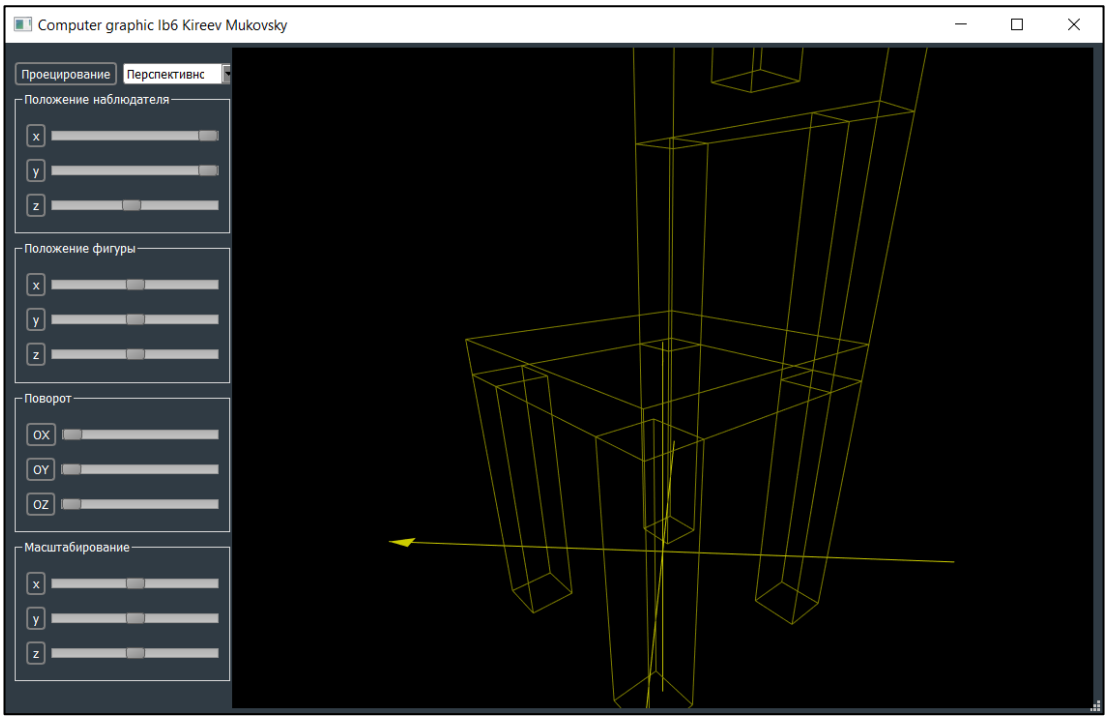
```
void Widget::changeCameraPositionX(float x) {
    cameraPosition->setX(x);
    if (perspective) setPerspectiveProjection();
    else setOrthoProjection();
}

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();
}
```

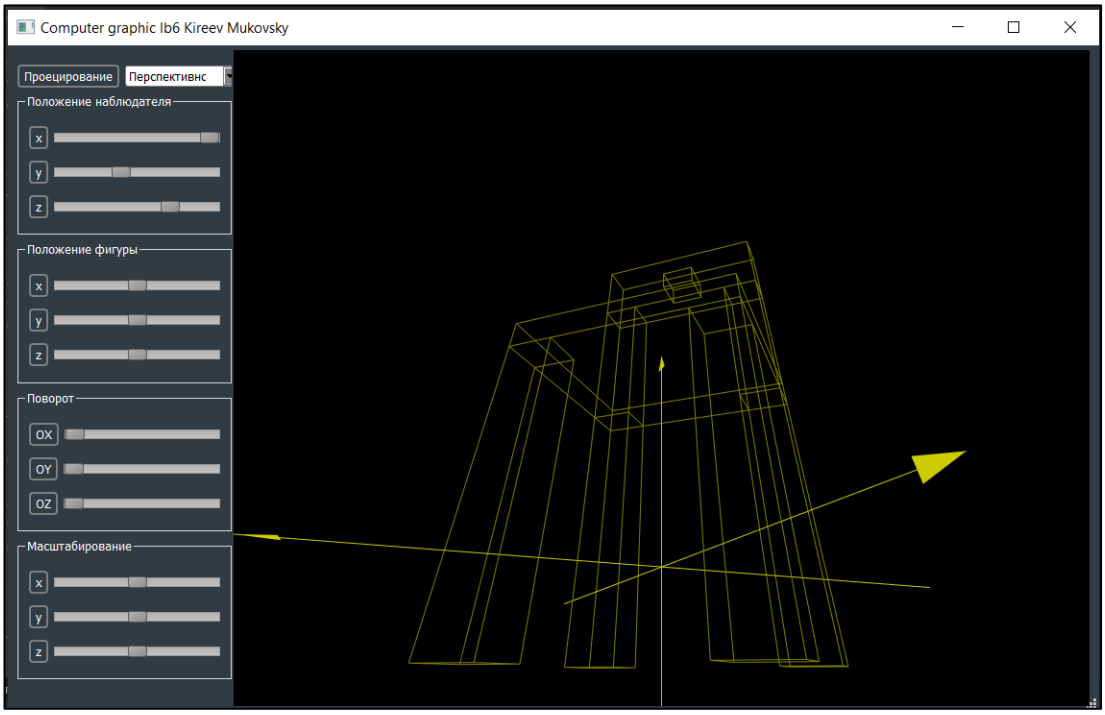
После чего в методе *setPerspectiveProjection()* изменяется положение камеры:

```
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();
}
```

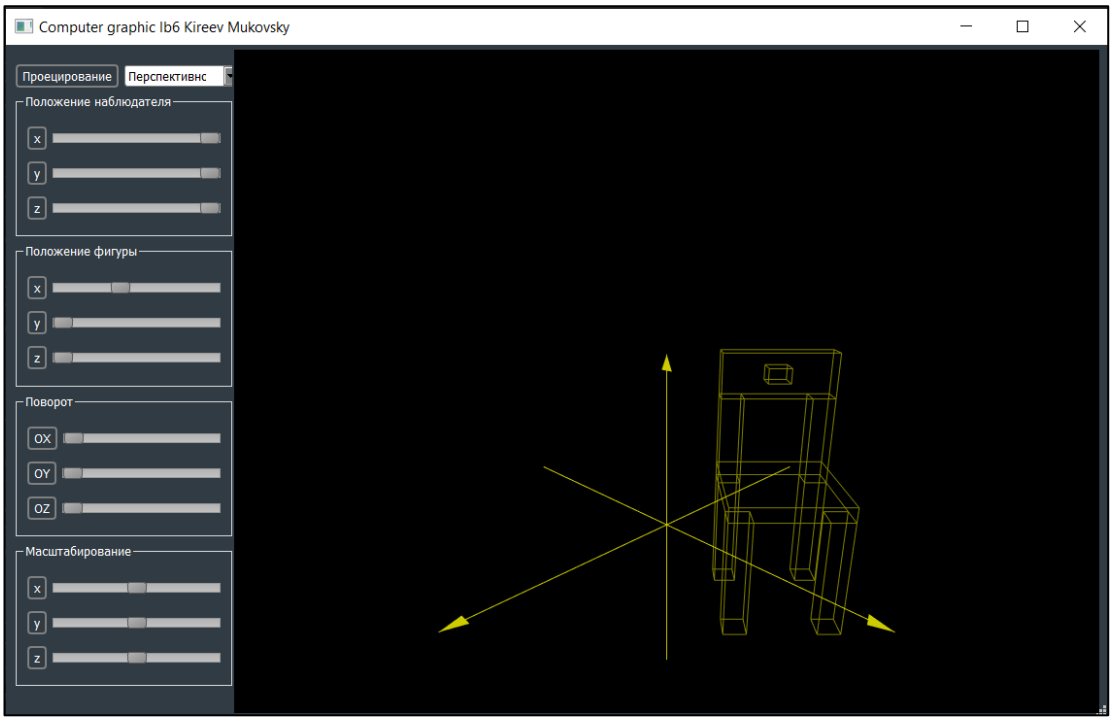
Пример работы программы

Изменяем положение наблюдателя

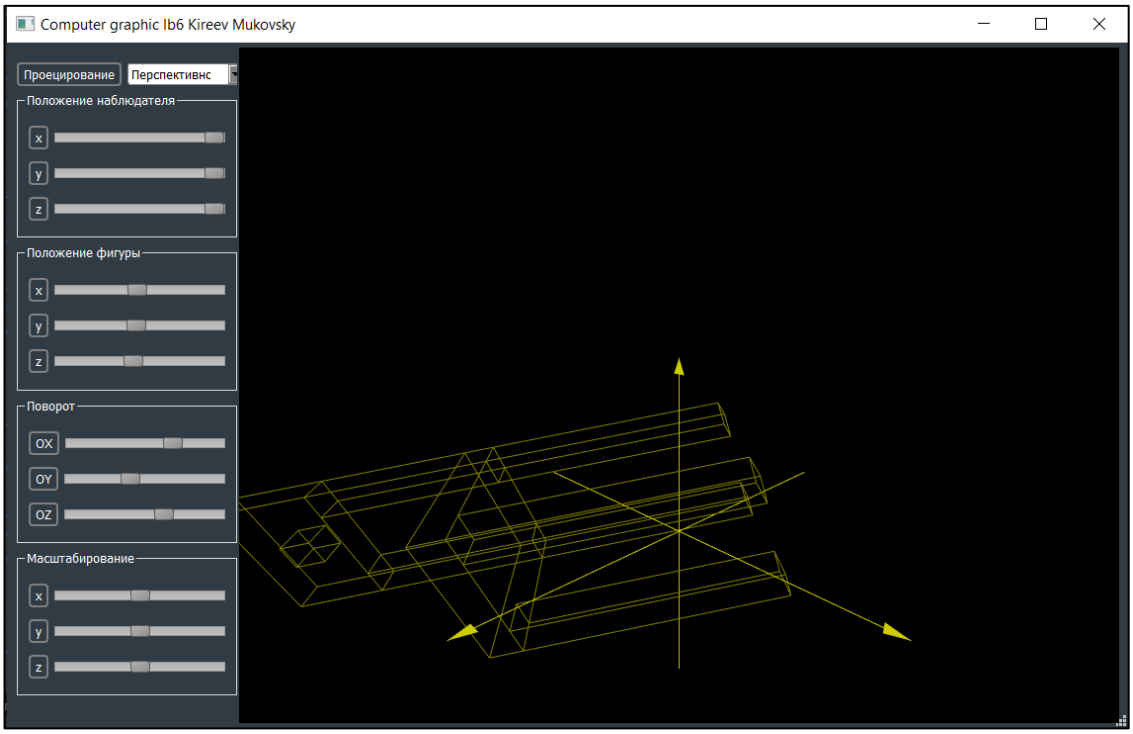




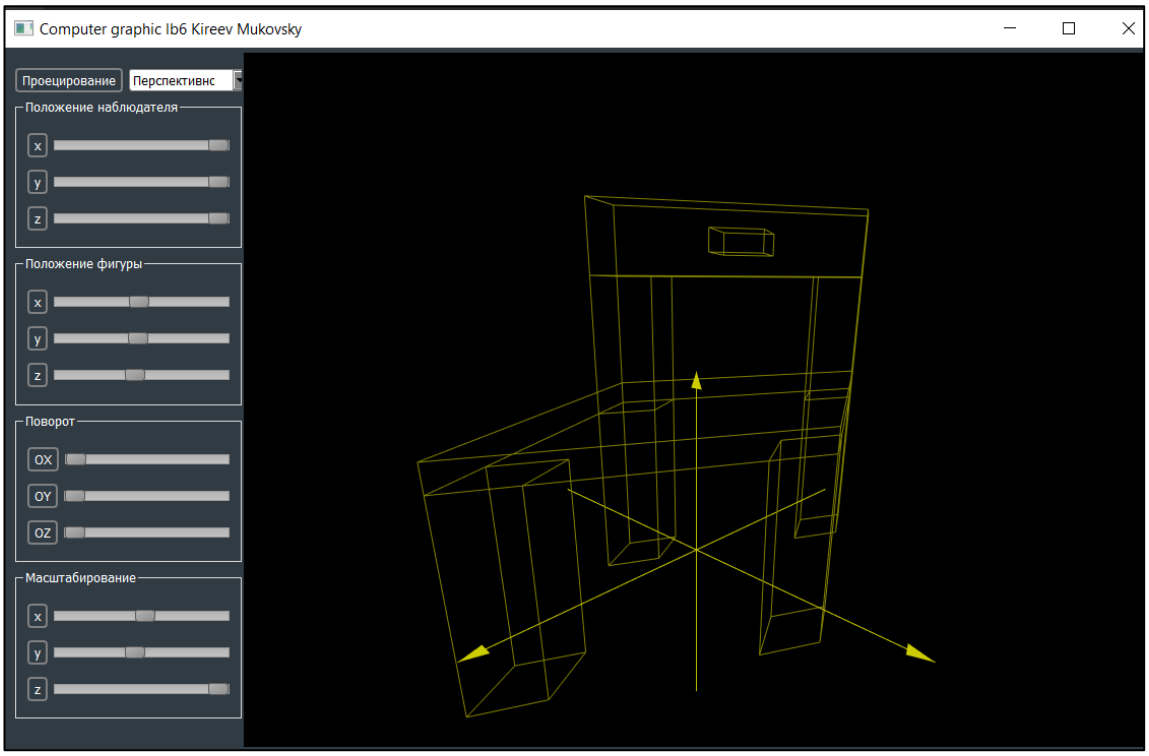
Изменяем положение фигуры



Поворачиваем фигуру



Масштабируем



Выводы.

В результате выполнения лабораторной работы была разработана программа, отображающая на экране заданный трехмерный объект (стул).

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>

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::white);
    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, 2.5, 3.0, 1.0};
```

```
    glLightfv(GL_LIGHT1, 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(wireframe10);
//        drawWireframe(wireframe11);
    }

    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();

    //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();

    //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, z*0.2));
            wireframe.append(QVector3D(x*0.2-1.6, j * starHeight*4, 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);
            wireframe3.append(QVector3D(x*0.2+1.6, j * starHeight*4, z*0.2));
            wireframe3.append(QVector3D(x*0.2+1.6, j * starHeight*4, 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, z*0.2+1.6));
            wireframe4.append(QVector3D(x*0.2, j * starHeight*4, 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, z*0.2-1.6));
            wireframe5.append(QVector3D(x*0.2, j * starHeight*4, 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);
            wireframe2.append(QVector3D(x, j * starHeight*0.5+4, z));
            wireframe2.append(QVector3D(x, j * starHeight*0.5+4, 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 < 4; i++) {
                x = starSize * cos(angle);
                z = starSize * sin(angle);
                wireframe10.append(QVector3D(x, j * starHeight, z));
                x = cos(angle + step/2);
                z = sin(angle + step/2);
                wireframe10.append(QVector3D(x, j * starHeight, z));
                angle += step;
            }
        }


//    for (int j = 0; j < 2; j++) {
//            float angle = 0, step = 2 * M_PI / 11, x, z;
//            for (int i = 0; i < 11; i++) {
//                x = starSize * cos(angle);
//                z = starSize * sin(angle);
//                wireframe11.append(QVector3D(x, j * starHeight, z));
//                x = (starSize - endSize) * cos(angle + step/2);
//                z = (starSize - endSize) * sin(angle + step/2);
//                wireframe11.append(QVector3D(x, j * starHeight, z));
//                angle += step;
//            }
//        }

    bottomPoints = topPoints;
    for (int i = 0; i < bottomPoints.size(); i++)
        bottomPoints[i].setY(0);
}

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() {

    //material
    GLfloat material_ambient[] = {0.5, 0.5, 0.5, 1.0};
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, material_ambient);

    GLfloat material_diffuse[] = {0.5, 0.5, 0.5, 1.0};
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material_diffuse);

    //light
    GLfloat ambient[] = {0.5, 0.5, 0.0, 0.5};//тень
    glLightfv(GL_LIGHT1, GL_AMBIENT, ambient);

    GLfloat diffuse[] = {1.0, 1.0, 0.0, 1.0};//цвет фигуры
    glLightfv(GL_LIGHT1, GL_DIFFUSE, diffuse);

    GLfloat position[] = {0.0, 2.5, 5.0, 1.0};
    glLightfv(GL_LIGHT1, GL_POSITION, position);


    glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, 0.0);
    glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, 0.09);
    glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, 0.0);
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT1);
    glEnable(GL_COLOR_MATERIAL);
}

void Widget::drawAxises() {

    glDisable(GL_LIGHTING);
    //x координата
    qglColor(Qt::blue);
    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);
    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);
    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(GL_LIGHT1, GL_DIFFUSE, d);
    update();
}

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

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