Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет: “Информационные технологии и прикладная математика”
Кафедра: 806 “Вычислительная математика и программирование”

Лабораторная работа №2

по курсу “Компьютерная графика”

Студент	Полей-Добронравова А.В.
Группа	М8О-307Б-18
Преподаватель	Г.С.Филиппов
Вариант	4
Дата	
Оценка	

Москва, 2020

Каркасная визуализация выпуклого многогранника. Удаление невидимых линий

Задача:
Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортогональной и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размера окна.
Вариант 4
Многогранник клин.

Описание

Программа написана на языке C++ в QTCreator, используется OpenGL. Создано два класса:

1) mainwindow - главное окно приложения;
2) myglwidget - виджет для отрисовки многогранника (файлы записаны с названием myglwidjet)

В **mainwindow** подключаются виджет, три слайдера для поворота фигуры по осям X, Y, Z, а также масштабирующий слайдер.
Существует кнопка **mode** при нажатии на которую можно показать задние линии или скрыть, закрашивая фигуру в красный цвет. При нажатии на **mode** срабатывает слот void MainWindow::sl1(), изменяющий значение поля виджета **modnow** и вызывающий *updateGL*() для перерисовки виджета. Внутри функции void MyGLWidget::draw() уже идет настройка отрисовки через

```
if (modenow == true) {
      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL); //
непрозрачность граней
   }
   else {
      glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   }
```

Клин - многогранник, имеющий две треугольные и три трапециевидные грани. Фигуры, образующие многогранник, я задаю в функции void MyGLWidget::draw() через конструкцию

```
qglColor(QT::color);
   glBegin(type_of_primitive);
      glVertex3f(x, y, z); ...
   glEnd();
```

Процесс масштабирования идет внутри функции void MyGLWidget::
resizeGL().
Совершение поворота в функциях: void setXRotation(int angle);
void setYRotation(int angle);
void setZRotation(int angle);

Код

main.cpp

```
#include <QApplication>
#include <QDesktopWidget>

#include "mainwindow.h"

int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  MainWindow window;
  window.resize(window.sizeHint());
  window.show();
  return app.exec();
}
```

mainwindow.h

```
#ifndef MAIN_WINDOW_H
#define MAIN_WINDOW_H

#include <QWidget>
#include <QSlider>
#include <QMainWindow>
#include <QPushButton>
#include "myglwidjet.h"

namespace Ui {
  class MainWindow;
}
class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  explicit MainWindow(QWidget *parent = 0);
  ~MainWindow();
private:
  Ui::MainWindow *ui;
  MyGLWidget* widget;
  QSlider *rotXSlider;
```

```
    QSlider *rotYSlider;
    QSlider *rotZSlider;
    QSlider * NSlider;
    QSlider *ScaleSlider;
    QPushButton* mode;
private slots:
    void sl1();
};
#endif
```

mainwindow.cpp

```
#include <QtWidgets>
#include <QGLWidget>
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "myglwidjet.h"

QSlider* createSlider()
{
    QSlider *slider = new QSlider(Qt::Vertical);
    slider->setRange(0, 360 * 16);
    slider->setSingleStep(16);
    slider->setPageStep(15 * 16);
    slider->setTickInterval(15 * 16);
    slider->setTickPosition(QSlider::TicksRight);
    return slider;
}

QSlider* createSlider1()
{
    QSlider *slider = new QSlider(Qt::Vertical);
    slider->setRange(6, 36);
    slider->setSingleStep(1);
    slider->setPageStep(1);
    slider->setTickInterval(1);
    slider->setTickPosition(QSlider::TicksRight);
    return slider;
}

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    widget = new MyGLWidget();

    rotXSlider = createSlider();
    rotYSlider = createSlider();
    rotZSlider = createSlider();
    ScaleSlider = createSlider1();

    connect(widget, SIGNAL(xRotationChanged(int)), rotXSlider, SLOT(setValue(int)));
    connect(rotXSlider, SIGNAL(valueChanged(int)), widget, SLOT(setXRotation(int)));
```

```
  connect(widget, SIGNAL(yRotationChanged(int)), rotYSlider, SLOT(setValue(int)));
  connect(rotYSlider, SIGNAL(valueChanged(int)), widget, SLOT(setYRotation(int)));

  connect(widget, SIGNAL(zRotationChanged(int)), rotZSlider, SLOT(setValue(int)));
  connect(rotZSlider, SIGNAL(valueChanged(int)), widget, SLOT(setZRotation(int)));

  connect(widget, SIGNAL(ScaleChanged(int)), ScaleSlider, SLOT(setValue(int)));
  connect(ScaleSlider, SIGNAL(valueChanged(int)), widget, SLOT(setScale(int)));
  QGridLayout *mainLayout = new QGridLayout;
  QWidget *centralWidget = new QWidget(this);

  mode = new QPushButton("change mode");
  connect(mode, SIGNAL(clicked()), SLOT(sl1()));
     mainLayout->addWidget(widget);
     mainLayout->addWidget(mode,2,0);
     mainLayout->addWidget(rotXSlider, 0, 1);
     mainLayout->addWidget(rotYSlider,0,2);
     mainLayout->addWidget(rotZSlider,0,3);

     //mainLayout->addWidget(NSlider);
     mainLayout->addWidget(ScaleSlider,0,4);
     centralWidget->setLayout(mainLayout);
     setCentralWidget(centralWidget);
}

void MainWindow::sl1() {
  if (widget->modenow == true) {
     widget->modenow = false;
  }
  else {
     widget->modenow = true;
  }
  widget->updateGL();
}

MainWindow::~MainWindow()
{
  delete rotXSlider;
  delete rotYSlider;
  delete rotZSlider;
  delete NSlider;
  delete ScaleSlider;
  delete widget;
}
```

myglwigjet.h

```
#ifndef MYGLWIDGET_H
#define MYGLWIDGET_H

#include <QGLWidget>

class MyGLWidget : public QGLWidget
{
```

```
    Q_OBJECT
public:
    explicit MyGLWidget(QWidget *parent = 0);
    bool modenow;
    double scale;
    void draw();
    void resizeGL(int width, int height);
protected:
    void initializeGL();
    void paintGL();
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *event);

public slots:
    // slots for xyz-rotation slider
    void setXRotation(int angle);
    void setYRotation(int angle);
    void setZRotation(int angle);
    //void setN(int value);
    void setScale(int value);

signals:
    // signaling rotation from mouse movement
    void xRotationChanged(int angle);
    void yRotationChanged(int angle);
    void zRotationChanged(int angle);
    //void NChanged(int value);
    void ScaleChanged(int value);
    void clicked();

private:
    QSize minimumSizeHint() const;
    QSize sizeHint() const;
    int xRot;
    int yRot;
    int zRot;
    //int N;
    QPoint lastPos;
};

#endif
```

myglwigjet.cpp

```
#include <QtWidgets>
#include <QtOpenGL>
#include <iostream>
#include <QDebug>

#include "myglwidjet.h"
```

```
MyGLWidget::MyGLWidget(QWidget *parent)
  : QGLWidget(QGLFormat(QGL::SampleBuffers), parent)
{
  // начальные параметры
  xRot = yRot = zRot = 0;
  modenow = true;
  scale = 0.06;
}

QSize MyGLWidget::minimumSizeHint() const
{
  return QSize(50, 50);
}

QSize MyGLWidget::sizeHint() const
{
  return QSize(400, 400);
}

// обнуление периода
static void qNormalizeAngle(int &angle)
{
  while (angle < 0)
     angle += 360;
  while (angle > 360)
     angle -= 360;
}

// поворот меша на угол angle, относительно оси oX
void MyGLWidget::setXRotation(int angle)
{
  qNormalizeAngle(angle);
  if (angle != xRot) {
     xRot = angle;
     emit xRotationChanged(angle);
     updateGL();
  }
}

// поворот меша на угол angle, относительно оси oY
void MyGLWidget::setYRotation(int angle)
{
  qNormalizeAngle(angle);
  if (angle != yRot) {
     yRot = angle;
     emit yRotationChanged(angle);
     updateGL();
  }
}

// поворот меша на угол angle, относительно оси oZ
void MyGLWidget::setZRotation(int angle)
{
  qNormalizeAngle(angle);
  if (angle != zRot) {
```

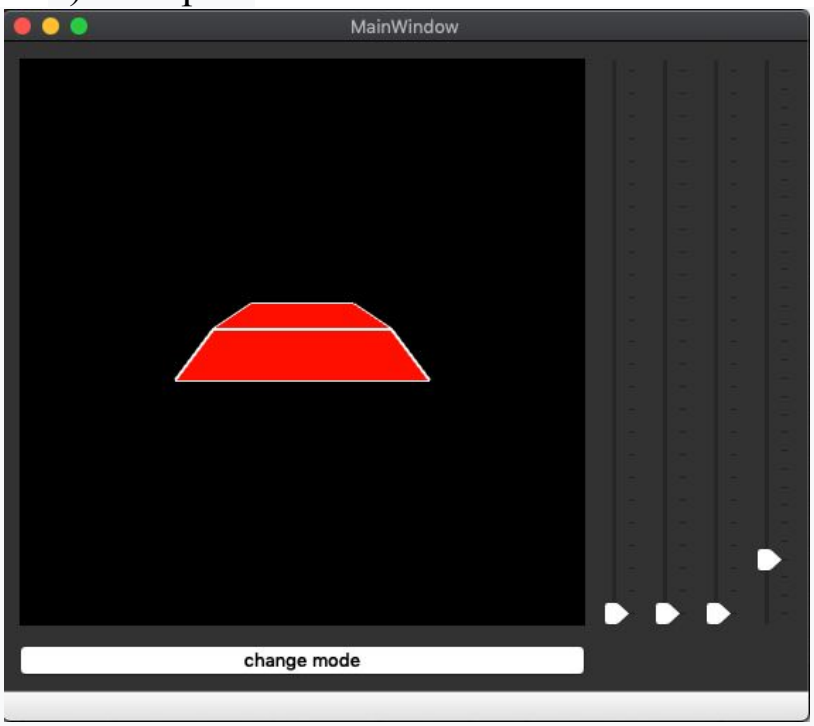
```
        zRot = angle;
        emit zRotationChanged(angle);
        updateGL();
    }
}

// задание масштаба
void MyGLWidget::setScale(int value)
{
    scale = value / 100.0;
    emit ScaleChanged(value);
    updateGL();
}

// инициализация OpenGL
void MyGLWidget::initializeGL()
{
    qglClearColor(Qt::black);

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glShadeModel(GL_SMOOTH);
}

// функция отрисовки
void MyGLWidget::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -10.0);
    glRotatef(xRot, 1.0, 0.0, 0.0);
    glRotatef(yRot, 0.0, 1.0, 0.0);
    glRotatef(zRot, 0.0, 0.0, 1.0);
    glScaled(scale,scale,scale);
    draw();
}

// параметры окна отрисовки
void MyGLWidget::resizeGL(int width, int height)
{
    int side = qMin(width, height);
    glViewport((width - side) / 2, (height - side) / 2, side, side);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
#ifdef QT_OPENGL_ES_1
    glOrthof(-2, +2, -2, +2, 1.0, 15.0);
#else
    glOrtho(-2, +2, -2, +2, 1.0, 15.0);
#endif
    glMatrixMode(GL_MODELVIEW);
}

// нажатие на кнопки мыши
void MyGLWidget::mousePressEvent(QMouseEvent *event)
```

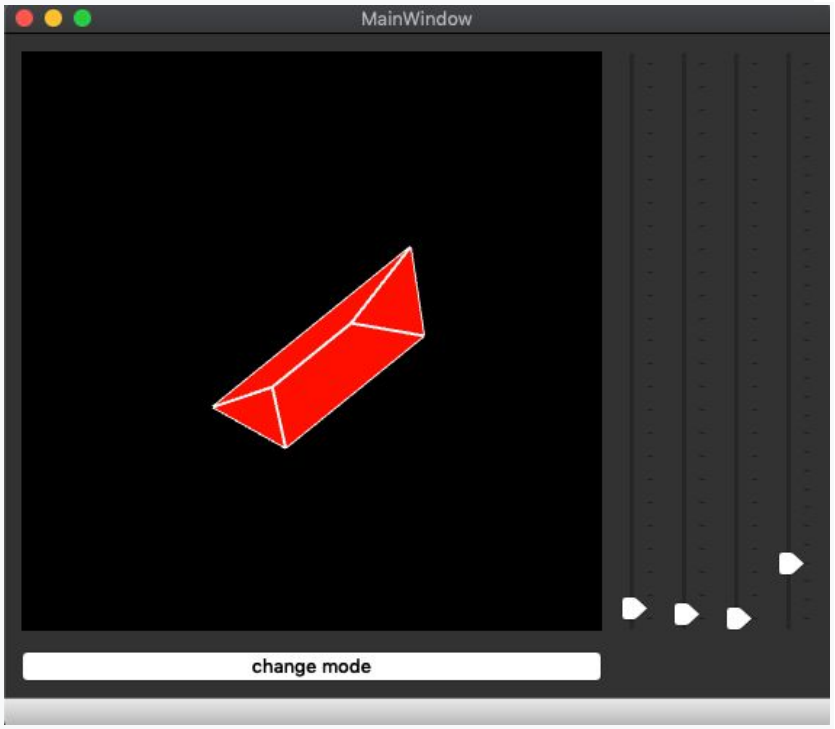
```
{
  lastPos = event->pos();
}

// перемещение мыши
void MyGLWidget::mouseMoveEvent(QMouseEvent *event)
{
  int dx = event->x() - lastPos.x();
  int dy = event->y() - lastPos.y();

  if (event->buttons() & Qt::LeftButton) {
     setXRotation(xRot + dy);
     setYRotation(yRot + dx);
  }
  if (event->buttons() & Qt::RightButton) {
     setXRotation(xRot + dy);
     setZRotation(zRot + dx);
  }
  lastPos = event->pos();
}

// отрисовка пирамиды
void MyGLWidget::draw()
{
  glLineWidth(2); // ширина ребер
  //glEnable(GL_CULL_FACE);
  if (modenow == true) {
     glPolygonMode(GL_FRONT_AND_BACK, GL_FILL); // непрозрачность граней
  }
  else {
     glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  }

  qglColor(Qt::white);
  glBegin(GL_LINE_STRIP);
     glVertex3f(10, -3, 0);
     glVertex3f(-10, -3, 0);
     glVertex3f(-7, 1, 5);
     glVertex3f(7, 1, 5);
     glVertex3f(10, -3, 0);
     glVertex3f(4, 3, 0);
     glVertex3f(-4, 3, 0);
  glEnd();
  glBegin(GL_LINE_STRIP);
     glVertex3f(-4, 3, 0);
     glVertex3f(-10, -3, 0);
  glEnd();
  glBegin(GL_LINE_STRIP);
     glVertex3f(-4, 3, 0);
     glVertex3f(-7, 1, 5);
  glEnd();
  glBegin(GL_LINE_STRIP);
     glVertex3f(4, 3, 0);
     glVertex3f(7, 1, 5);
  glEnd();
```

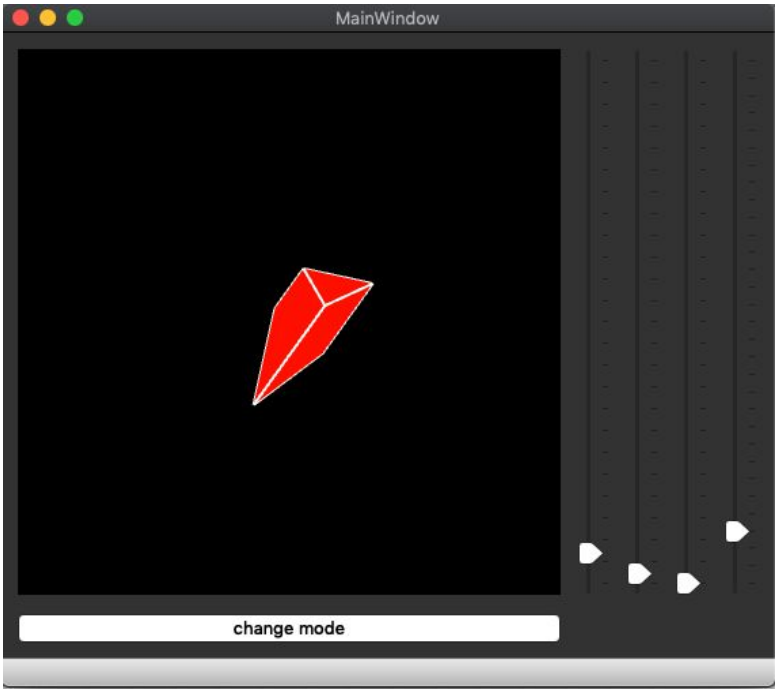
```
  qglColor(Qt::red);
     // основание клина

     glBegin(GL_QUADS);
        glVertex3f(-4, 3, 0);
        glVertex3f(4, 3, 0);
        glVertex3f(10, -3, 0);
        glVertex3f(-10, -3, 0);
     glEnd();
     //боковина клина 1
     qglColor(Qt::red);
     glBegin(GL_QUADS);
        glVertex3f(-7, 1, 5);
        glVertex3f(7, 1, 5);
        glVertex3f(4, 3, 0);
        glVertex3f(-4, 3, 0);
     glEnd();
     qglColor(Qt::red);
     //боковина клина 2
     glBegin(GL  QUADS);
        glVertex3f(-7, 1, 5);
        glVertex3f(-10, -3, 0);
        glVertex3f(10, -3, 0);
        glVertex3f(7, 1, 5);
     glEnd();
     qglColor(Qt::red);
     // треугольник 1
     glBegin(GL  TRIANGLE_STRIP);
        glVertex3f(-10, -3, 0);
        glVertex3f(-7, 1, 5);
        glVertex3f(-4, 3, 0);
        glVertex3f(-10, -3, 0);
     glEnd();
     // треугольник 2
     glBegin(GL_TRIANGLE_STRIP);
        glVertex3f(10, -3, 0);
        glVertex3f(7, 1, 5);
        glVertex3f(4, 3, 0);
        glVertex3f(10, -3, 0);
     glEnd();
}
```

Пример работы

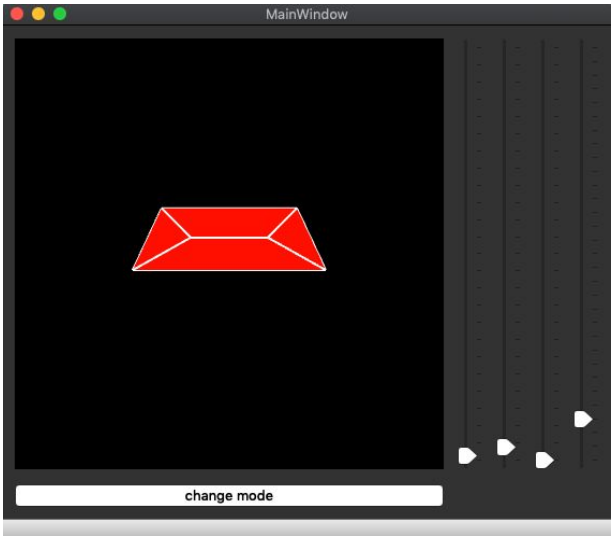
1) поворот:

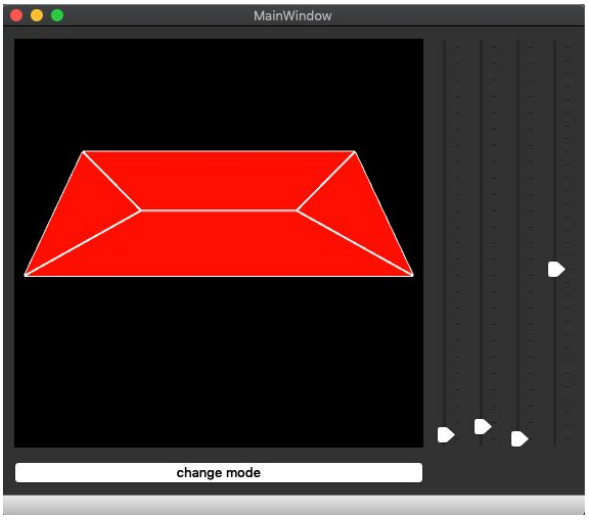




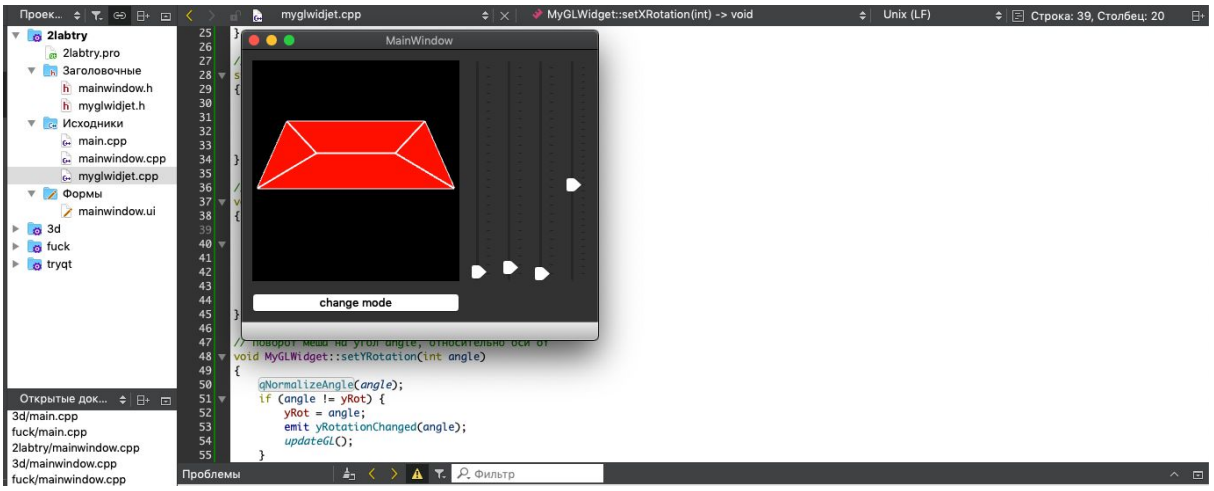

MainWindow
change mode

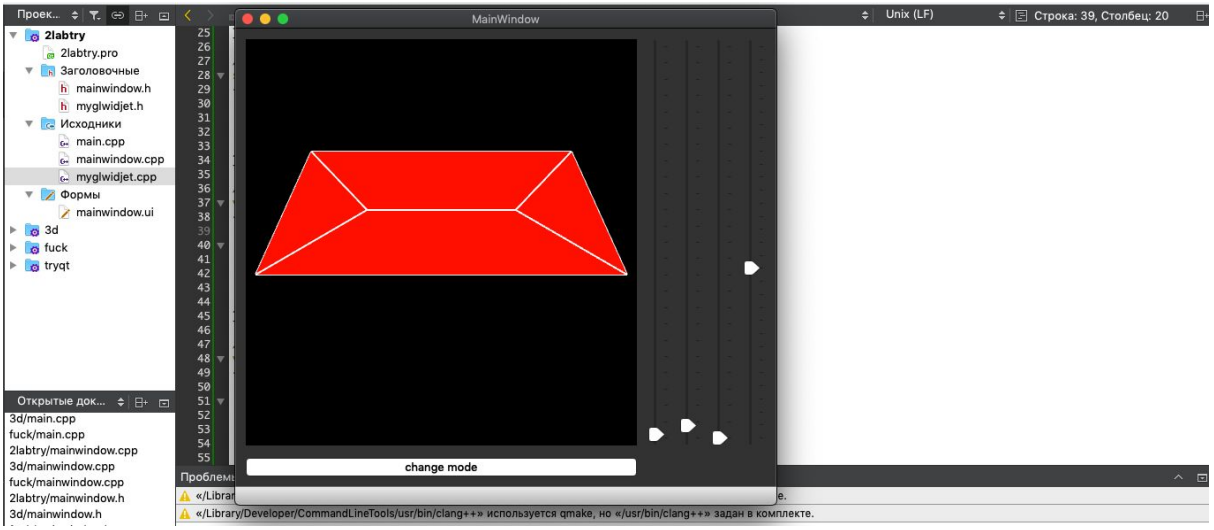
2) масштаб


MainWindow
change mode

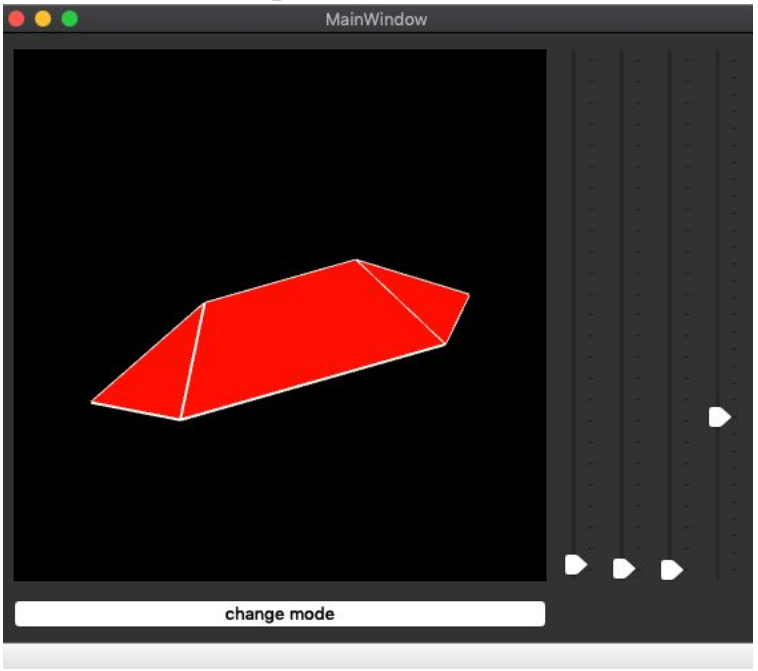

MainWindow
change mode

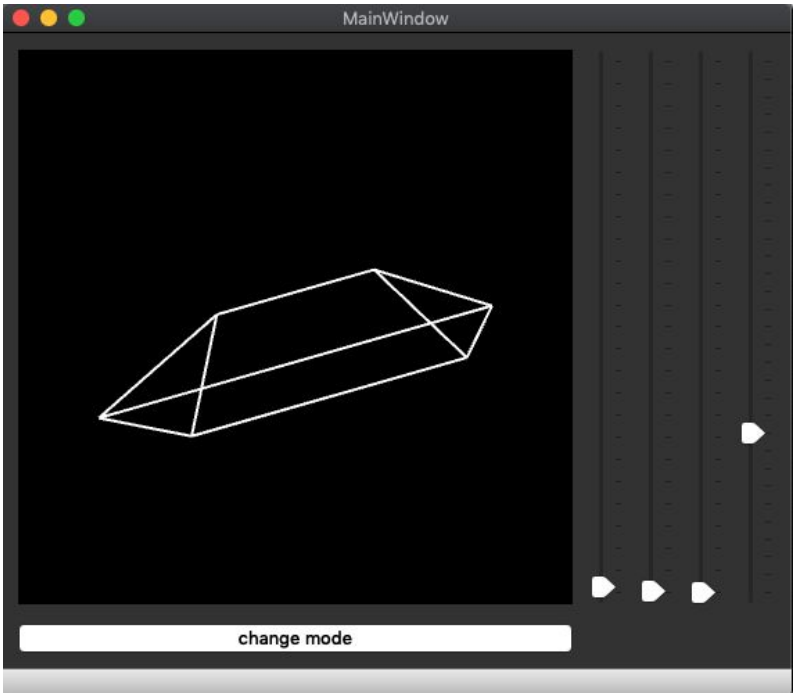
3) автоматическое центрирование и изменение размера изображения





4) изменение режима





Вывод

OpenGL это очень удобная для работы библиотека. Единственное неудобство, с которым я столкнулась в этой лабораторной: факт того, что для отрисовки GL_QUADS, GL_POLIGONS важен порядок передачи вершин, а также то, что определение FRONT и BACK плоскости определяется не камерой, направленной на объект, а опять же порядком задания вершин. FRONT и BACK нужно учитывать например при закрашивании плоских фигур.

Ключевое слово emit вырабатывает сигнал объекта с новым значением в качестве аргумента.