Московский Авиационный Институт
(Национальный Исследовательский Университет)

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

Курсовая работа
на тему “Линейчатая поверхность с направляющими кубическими кривыми Безье”
по курсу “Компьютерная графика”

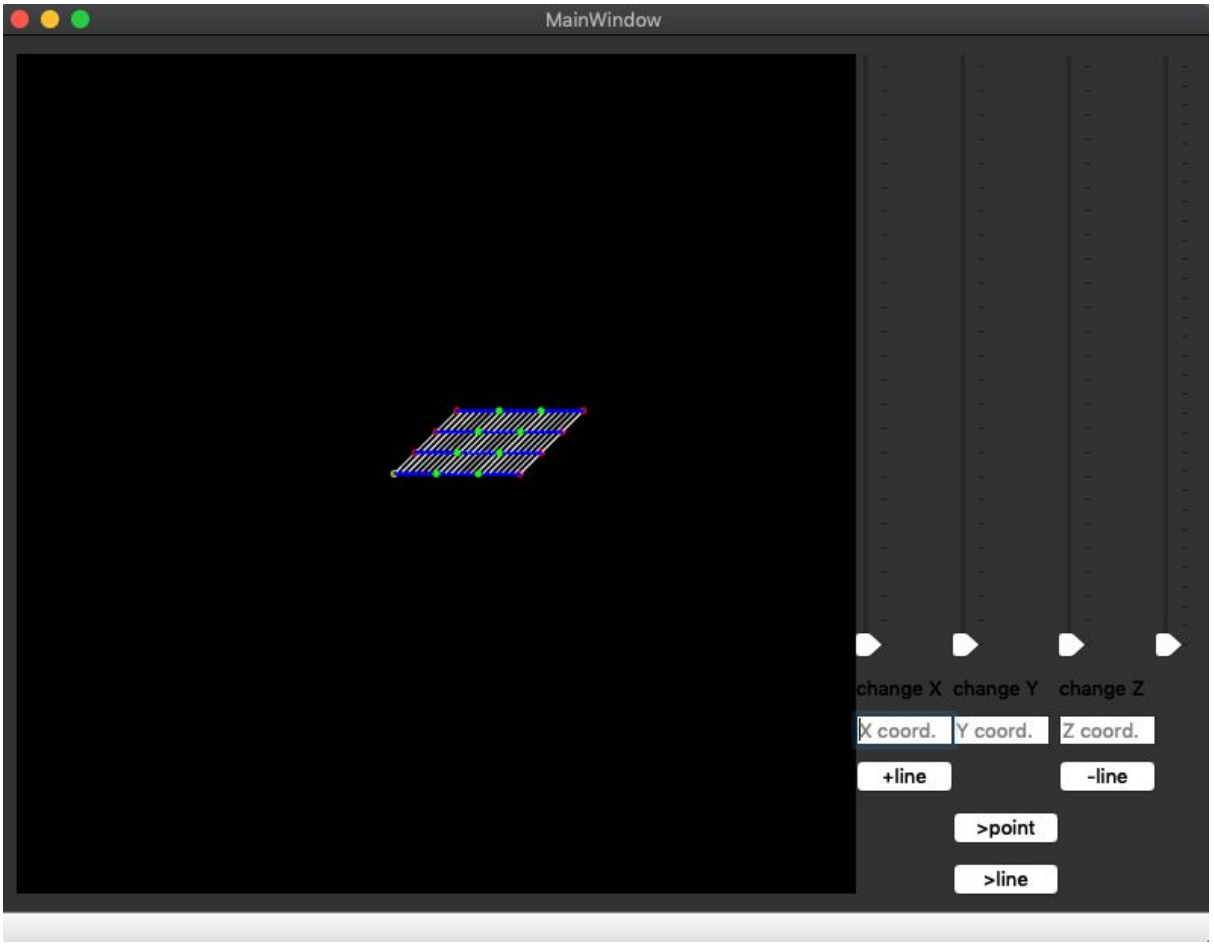
Студент	Полей-Добронравова А.В.
Группа	М8О-307Б-18
Преподаватель	Г.С.Филиппов
Вариант	4
Дата	
Оценка	

Москва, 2020

Задача

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и вводятся из файла или в панели ввода данных. Должна быть обеспечена возможность тестирования программы на различных наборах исходных данных. Программа должна обеспечивать выполнение аффинных преобразований для заданной порции поверхности, а также возможность управлять количеством изображаемых параметрических линий.

Интерфейс



При запуске появляется окно, которое содержит:

1) Изображение самой поверхности;
2) Четыре слайдера, слева направо - поворот по оси OX, поворот по оси OY, поворот по оси OZ, изменение масштаба. Повернуть поверхность можно также с помощью мышки.

3) Выбранная точка из опорных подсвечивается желтым цветом, на рисунке это нижняя слева. Для изменения координат выбранной точки три поля для ввода.
4) Линия считается выбранной, если ей принадлежит выбранная точка. Чтобы удалить выбранную линию, нужно нажать кнопку “-line”. Удалять можно до тех пор, пока линий не станет две.
5) Чтобы добавить еще одну линию Безье с краю поверхности, нужно нажать “+line”.
6) Для изменения выбранной точки на текущей линии нужно нажать кнопку “>point”
7) Для изменения выбранной линии нужно нажать кнопку “>line”.

Описание

Программа написана на языке C++ в QTCreator. Реализовано два класса:

1) mainwindow - окно для отрисовки.
2) beziercurve - класс, реализующий все слоты виджетов главного окна, создание и модификацию поверхности.

Кривая Безье N-1 степени задается уравнением

$$\mathbf{B}_{P_0 \dots P_n} = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k P_k$$

где P - опорная точка (x,y,z). N - количество опорных точек. t - параметр, изменяющийся от 0 до 1 с маленьким шагом (для иллюзии беспрерывной линии).
Для кубической кривой Безье её уравнение выглядит как

$$(1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$$

Начальное значение количества направляющих линий поверхности равно 4. Есть два вектора: вектор векторов опорных точек (4 штуки) направляющих линий, и вспомогательный вектор, хранящий все точки этих линий с промежутком параметра t равному 0.001. Кривые

соединяются с соседними кривыми с помощью прямых линий по краям, а также через каждые свои 50 точек. Всё это кроме задания опорных точек направляющих линий создано в функции void beziercurve::draw().

При удалении выбранной линии в функции void beziercurve::changelinecount2() из вектора точек удаляются все точки, принадлежащие ей, и идет обновление картинки. При добавлении новой линии в функции void beziercurve::changelinecount1() добавляется вектор точек, отличающийся от крайнего по осям OX и OY на одну координату.

void beziercurve::changepoint1() меняет выбор точки, передвигая на следующую. void beziercurve::changepoint2() меняет выбор линии, передвигая точку на ту же по индексу соседней линии.

Код

main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  MainWindow w;
  w.show();
  return a.exec();
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "beziercurve.h"

#include <QMainWindow>
#include <QPushButton>
#include <QSlider>
#include <QLineEdit>

namespace Ui {
class MainWindow;
}
```

```
class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  explicit MainWindow(QWidget *parent = 0);
  ~MainWindow();

private:
  Ui::MainWindow *ui;
  beziercurve* widget;
  QSlider *rotXSlider;
  QSlider *rotYSlider;
  QSlider *rotZSlider;
  QSlider *ScaleSlider;
  QLineEdit* changeXline;
  QLineEdit* changeYline;
  QLineEdit* changeZline;
  QPushButton* line1;
  QPushButton* line2;
  QPushButton* point1;
  QPushButton* point2;
};

#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <QGridLayout>
#include <QLabel>
#include <QComboBox>

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(10, 40);
  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 beziercurve();

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

  line1 = new QPushButton();
  line1->setText("+line");
  line2 = new QPushButton();
  line2->setText("-line");

  point1 = new QPushButton();
  point1->setText(">point");
  point2 = new QPushButton();
  point2->setText(">line");

  QLabel *echoLabel = new QLabel(tr("change X"));
  changeXline = new QLineEdit;
  changeXline->setPlaceholderText("X coord.");
  changeXline->setFocus();
  changeXline->setFixedSize(widget->width()/10,20);

  QLabel *echoLabel2 = new QLabel(tr("change Y"));
  changeYline = new QLineEdit;
  changeYline->setPlaceholderText("Y coord.");
```

```
  changeYline->setFocus();
  changeYline->setFixedSize(widget->width()/10,20);

  QLabel *echoLabel3 = new QLabel(tr("change Z"));
  changeZline = new QLineEdit;
  changeZline->setPlaceholderText("Z coord.");
  changeZline->setFocus();
  changeZline->setFixedSize(widget->width()/10,20);

  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)));

  connect(changeXline, SIGNAL(textEdited(const QString)), widget, SLOT(setX(const
QString)));
  connect(changeYline, SIGNAL(textEdited(const QString)), widget, SLOT(setY(const
QString)));
  connect(changeZline, SIGNAL(textEdited(const QString)), widget, SLOT(setZ(const
QString)));

  connect(line1, SIGNAL(pressed ()), widget, SLOT(changelinecount1()));
  connect(line2, SIGNAL(pressed ()), widget, SLOT(changelinecount2()));

  connect(point1, SIGNAL(pressed ()), widget, SLOT(changepoint1()));
  connect(point2, SIGNAL(pressed ()), widget, SLOT(changepoint2()));

  QGridLayout *mainLayout = new QGridLayout;

  QWidget *centralWidget = new QWidget(this);

     mainLayout->addWidget(widget,0,0,6,1);
     mainLayout->addWidget(rotXSlider, 0,1);
     mainLayout->addWidget(rotYSlider,0,2);
     mainLayout->addWidget(rotZSlider,0,3);
     mainLayout->addWidget(ScaleSlider,0,4);
     mainLayout->addWidget(echoLabel,1,1);
```

```
    mainLayout->addWidget(changeXline,2,1);
    mainLayout->addWidget(echoLabel2,1,2);
    mainLayout->addWidget(changeYline,2,2);
    mainLayout->addWidget(echoLabel3,1, 3);
    mainLayout->addWidget(changeZline,2,3);
    mainLayout->addWidget(line1,3,1);
    mainLayout->addWidget(line2,3,3);
    mainLayout->addWidget(point1,4,2);
    mainLayout->addWidget(point2,5,2);
    mainLayout->setHorizontalSpacing(0);
    centralWidget->setLayout(mainLayout);
    setCentralWidget(centralWidget);
}

MainWindow::~MainWindow()
{
  delete ui;
}
```

beziercurve.h

```
#ifndef BEZIERCURVE_H
#define BEZIERCURVE_H
#include <QPen>
#include <QBrush>
#include <QWidget>
#include <QtMath>
#include <QGLWidget>
#include <QVector>
#include <QVector3D>
#include <iostream>

class beziercurve : public QGLWidget
{
  Q_OBJECT
public:
  explicit beziercurve(QWidget *parent = 0);
  ~beziercurve();
  void draw();
  void resizeGL(int width, int height);
  void initializeGL();
  void paintGL();
  double scale;
  double toch; //точность аппроксимации
  int line_count_start; //количество линий безье в начале
```

```
    int selected_line;
    int selected_point;

protected:
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *);
    void mouseReleaseEvent(QMouseEvent *) {

    };

public slots:
    // slots for xyz-rotation slider
    void setXRotation(int angle);
    void setYRotation(int angle);
    void setZRotation(int angle);
    void setScale(int value);
    void setX(const QString value);
    void setY(const QString value);
    void setZ(const QString value);
    void changelinecount1();
    void changelinecount2();
    void changepoint1();
    void changepoint2();

signals:
    // signaling rotation from mouse movement
    void xRotationChanged(int angle);
    void yRotationChanged(int angle);
    void zRotationChanged(int angle);
    void ScaleChanged(int value);
    void clicked();
    void pressed ();
    void valueChanged(float);
    void textEdited(const QString &text);

private:
    const int NUM_POINTS = 4;
    const qreal POINT_RADIUS = 4.0;

    QVector<QVector<QVector3D>> m_points;
    QVector<QVector<QVector3D>> all_points; //буфер для отрисовки всех точек

    QPen       m_curvePen;
    QSize minimumSizeHint() const;
```

```
    QSize sizeHint() const;
    int xRot;
    int yRot;
    int zRot;
    QPoint lastPos;
};

#endif // BEZIERCURVE_H
```

beziercurve.cpp

```
#include "beziercurve.h"
#include <QPainter>
#include <QMouseEvent>
#include <array>
#include <QMatrix4x4>
#include <GLUT/glut.h>

beziercurve::beziercurve(QWidget *parent)
    : QGLWidget(QGLFormat(QGL::SampleBuffers), parent)
    , m_curvePen(Qt::black)
{
    m_curvePen.setWidth(2);
    xRot = yRot = zRot = 0;
    scale = 0.1;
    line_count_start = 4;
    selected_line = 0;
    selected_point = 0;
    qreal diff = 0;
    for(int i = 0; i < line_count_start; i++) {
        QVector3D v1(diff-2,diff,diff - 2);
        QVector3D v2(diff,diff,diff+ 2);
        QVector3D v3(diff + 2,diff,diff + 2);
        QVector3D v4(diff + 4,diff,diff -2);
        QVector<QVector3D> dop;
        dop.push_back(v1);
        dop.push_back(v2);
        dop.push_back(v3);
        dop.push_back(v4);
        m_points.push_back(dop);
        diff += 1;
    }
    update();
}
```

```
beziercurve::~beziercurve()
{

}


void beziercurve::mousePressEvent(QMouseEvent *event)
{
  lastPos = event->pos();
}

void beziercurve::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();
  update();
}


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

QSize beziercurve::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 beziercurve::setXRotation(int angle)
{
    qNormalizeAngle(angle);
    if (angle != xRot) {
        xRot = angle;
        emit xRotationChanged(angle);
        updateGL();
    }
}

// поворот меша на угол angle, относительно оси oY
void beziercurve::setYRotation(int angle)
{
    qNormalizeAngle(angle);
    if (angle != yRot) {
        yRot = angle;
        emit yRotationChanged(angle);
        updateGL();
    }
}
void beziercurve::changelinecount1() {
    line_count_start += 1;
    QVector<QVector3D> buf(m_points[m_points.size() - 1]);
    buf[0].setX(buf[0].x() + 1);
    buf[1].setX(buf[1].x() + 1);
    buf[2].setX(buf[2].x() + 1);
    buf[3].setX(buf[3].x() + 1);
    buf[0].setY(buf[0].y() + 1);
    buf[1].setY(buf[1].y() + 1);
    buf[2].setY(buf[2].y() + 1);
    buf[3].setY(buf[3].y() + 1);
    m_points.push_back(buf);
    updateGL();
}

void beziercurve::changelinecount2() {
    if (line_count_start >= 3) {
        QVector<QVector<QVector3D>> buf;
```

```
        int i = 0;
        for (; i < selected_line; i++) {
            QVector<QVector3D> buf1;
            for (int j = 0; j < 4; j++) {
                buf1.push_back(m_points[i][j]);
            }
            buf.push_back(buf1);
        }
        i++;
        for (; i < m_points.size(); i++) {
            QVector<QVector3D> buf1;
            for (int j = 0; j < 4; j++) {
                buf1.push_back(m_points[i][j]);
            }
            buf.push_back(buf1);
        }
        m_points.clear();
        m_points = buf;
        line_count_start -= 1;
        selected_line = 0;
    }
    updateGL();
}

void beziercurve::changepoint1() {
    selected_point = (selected_point + 1) % 4;
    updateGL();
}

void beziercurve::changepoint2() {
    selected_line = (selected_line + 1) % line_count_start;
    updateGL();
}

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

```
void beziercurve::setX(const QString value) {
  m_points[selected_line][selected_point].setX(value.toFloat());
  updateGL();
}
void beziercurve::setY(const QString value) {
  m_points[selected_line][selected_point].setY(value.toFloat());
  updateGL();
}
void beziercurve::setZ(const QString value) {
  m_points[selected_line][selected_point].setZ(value.toFloat());
  updateGL();
}

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

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

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

// функция отрисовки
void beziercurve::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 beziercurve::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);
}

void beziercurve::draw()
{
  glClear(GL_COLOR_BUFFER_BIT);
  all_points.clear();
  for (int i = 0; i < m_points.size(); i++) {
     QVector<QVector3D> buf;
     for (double t = 0; t < 1; t = t + 0.001) {
        double x = (1-t)*(1-t)*(1-t)*m_points[i][0].x() + 3*(1-t)*(1-t)*t*m_points[i][1].x() +
3*(1-t)*t*t*m_points[i][2].x() + t*t*t*m_points[i][3].x();
        double y = (1-t)*(1-t)*(1-t)*m_points[i][0].y() + 3*(1-t)*(1-t)*t*m_points[i][1].y() +
3*(1-t)*t*t*m_points[i][2].y() + t*t*t*m_points[i][3].y();
        double z = (1-t)*(1-t)*(1-t)*m_points[i][0].z() + 3*(1-t)*(1-t)*t*m_points[i][1].z() +
3*(1-t)*t*t*m_points[i][2].z() + t*t*t*m_points[i][3].z();
        glPointSize(2);
          glBegin(GL_POINTS);
           glColor3d(0,0,1);
            glVertex3d(x,y,z);
           glEnd();
        QVector3D xyz(x, y, z);
        buf.push_back(xyz);
     }
     all_points.push_back(buf);
  }
  for (int i = 0; i < m_points.size(); i++) { //опорные точки
     glPointSize(5);
       glBegin(GL_POINTS);
       if (i == selected_line && selected_point== 0) {
          glPointSize(50);
```

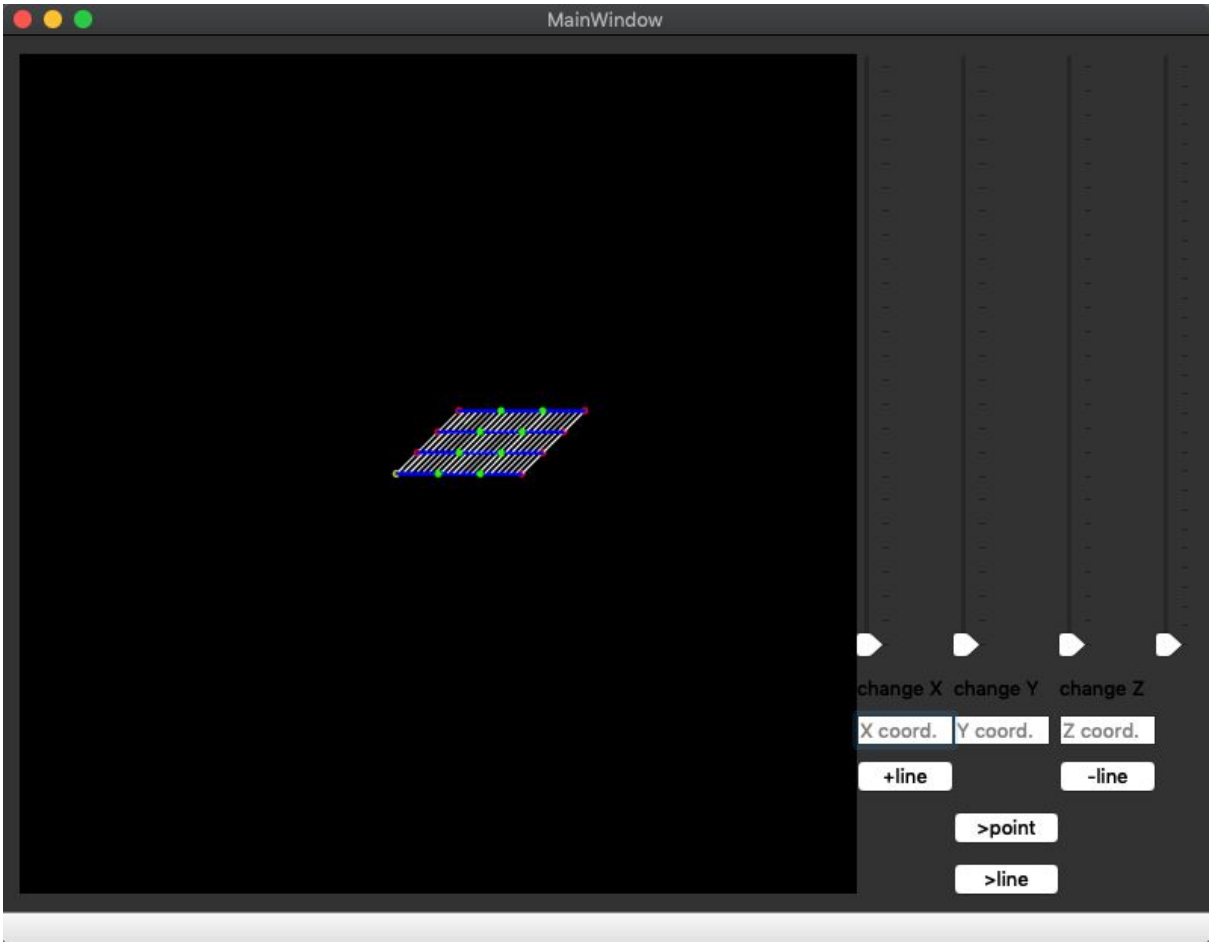
```
            glColor3d(1,1,0);
        }
        else {
         glColor3d(1,0,0);
        }
         glVertex3d(m_points[i][0].x(),m_points[i][0].y(),m_points[i][0].z());
         if (i == selected_line && selected_point == 1) {
            glPointSize(50);
            glColor3d(1,1,0);
         }
         else {
          glColor3d(0,1,0);
         }
        glVertex3d(m_points[i][1].x(),m_points[i][1].y(),m_points[i][1].z());
        if (i == selected_line && selected_point== 2) {
           glPointSize(50);
           glColor3d(1,1,0);
        }
        else {
         glColor3d(0,1,0);
        }
        glVertex3d(m_points[i][2].x(),m_points[i][2].y(),m_points[i][2].z());
        if (i == selected_line && selected_point== 3) {
           glPointSize(50);
           glColor3d(1,1,0);
        }
        else {
         glColor3d(1,0,0);
        }
        glVertex3d(m_points[i][3].x(),m_points[i][3].y(),m_points[i][3].z());
       glEnd();
}

for (int i = 0; i < all_points.size() - 1; i++) { //рисуем продольные линии с шагом des1
   for (int j = 0; j < all_points[i].size(); j += 50) {
   glBegin(GL_LINES);
    glColor3d(1,1,1);
    glVertex3d(all_points[i][j].x(),all_points[i][j].y(),all_points[i][j].z());
    glVertex3d(all_points[i+1][j].x(),all_points[i+1][j].y(),all_points[i+1][j].z());
   glEnd();
   }
}
for (int i = 0; i < m_points.size() - 1; i++) {
   glPointSize(5);
```

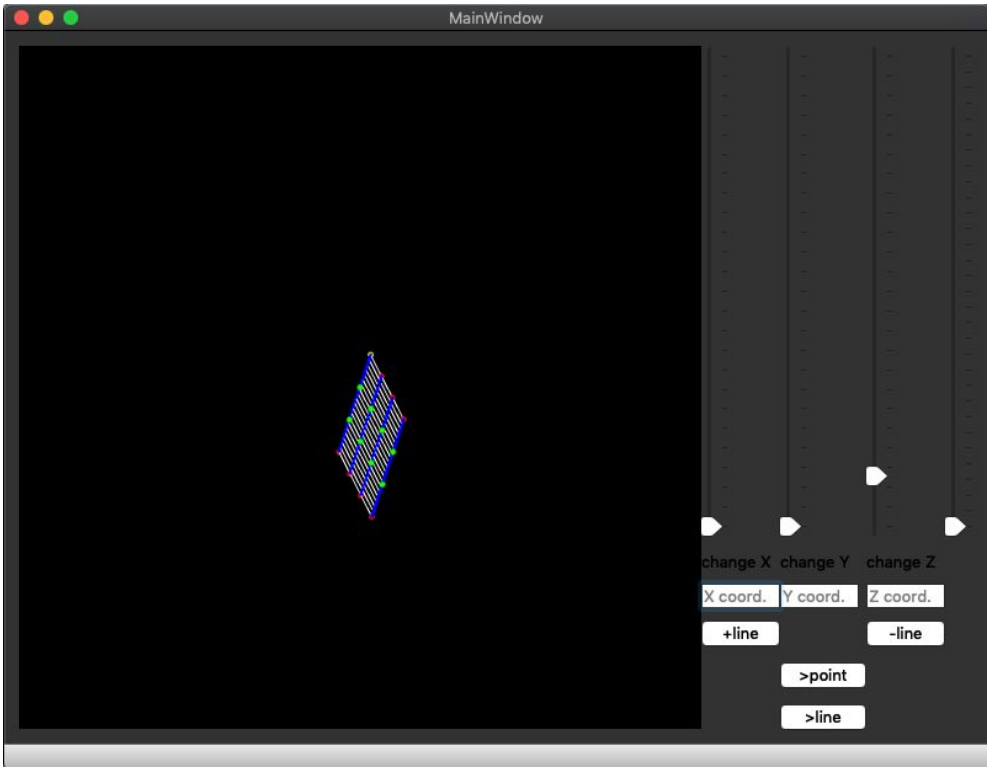
```
        glBegin(GL_LINES);
         glColor3d(1,1,1);
         glVertex3d(m_points[i][0].x(),all_points[i][0].y(),all_points[i][0].z());
         glVertex3d(m_points[i+1][0].x(),all_points[i+1][0].y(),all_points[i+1][0].z());
        glEnd();
        glBegin(GL_LINES);
         glColor3d(1,1,1);
         glVertex3d(m_points[i][3].x(),all_points[i][3].y(),all_points[i][3].z());
         glVertex3d(m_points[i+1][3].x(),all_points[i+1][3].y(),all_points[i+1][3].z());
        glEnd();
    }
}
```

Демонстрация работы по шагам

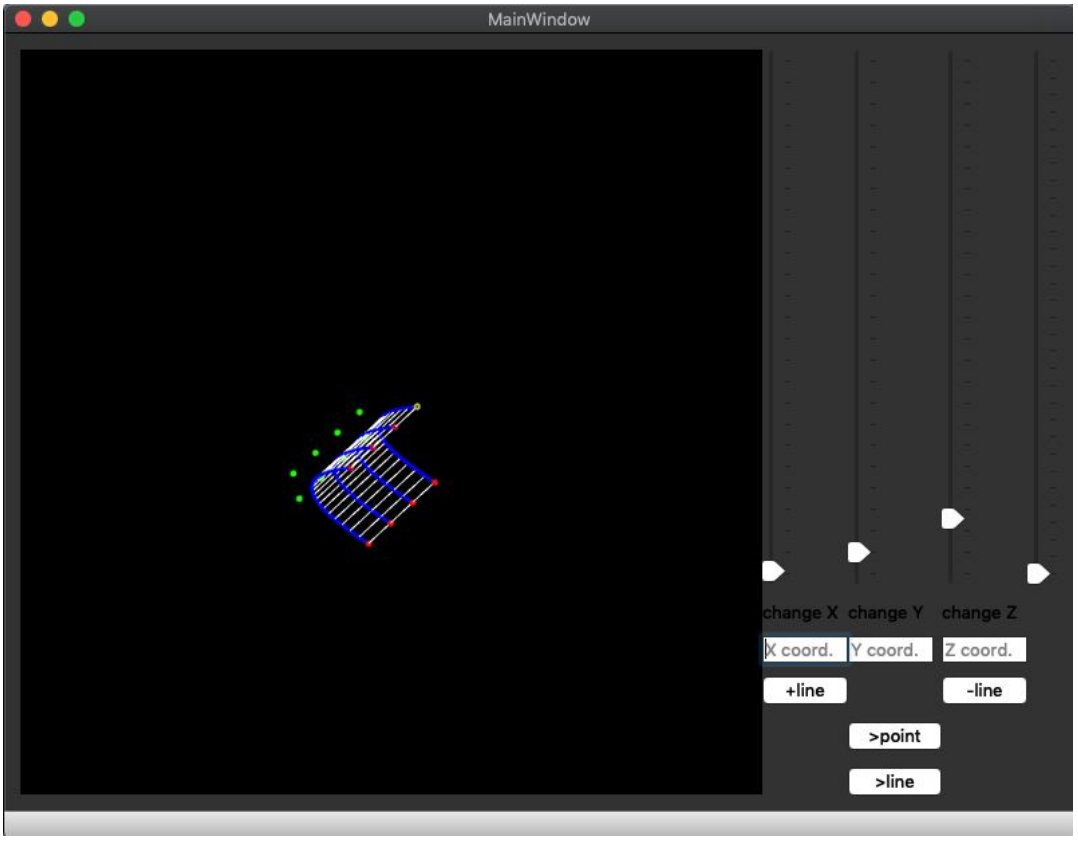
1) Запускаем программу



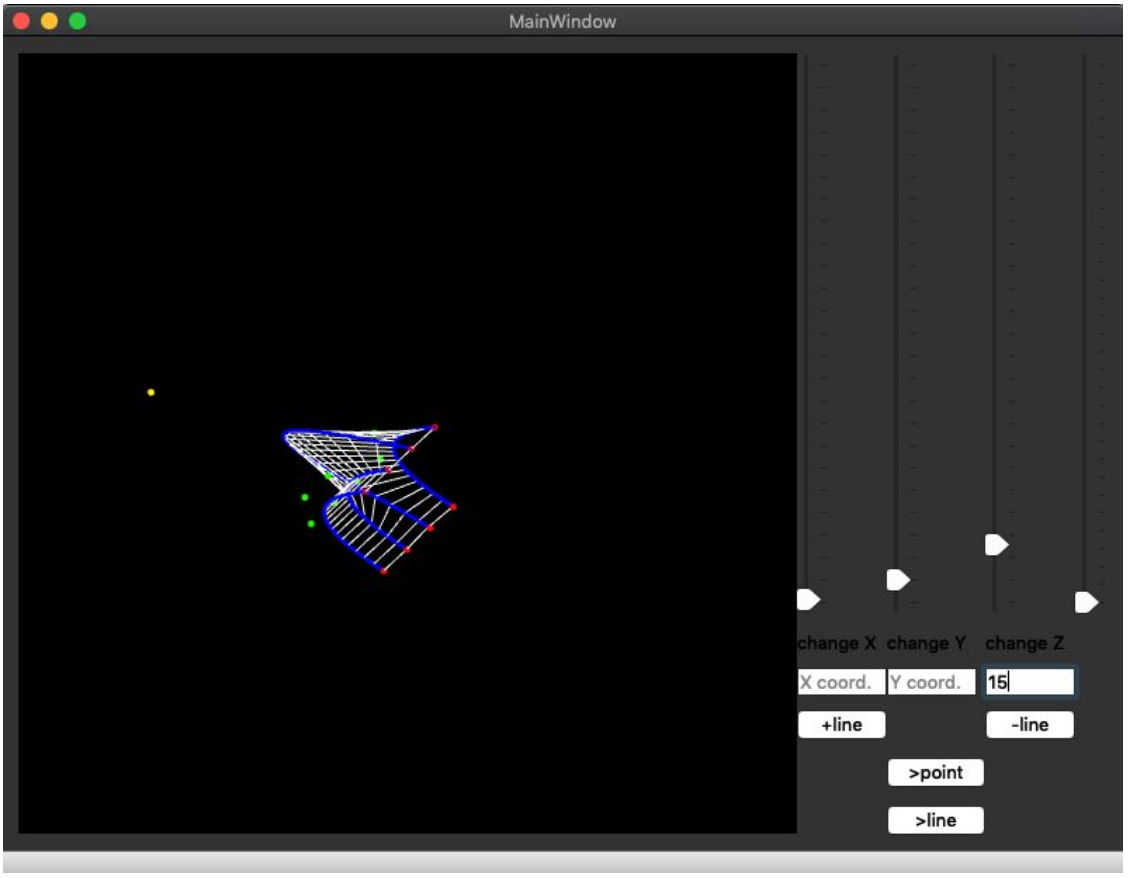
2) Повернем по OZ



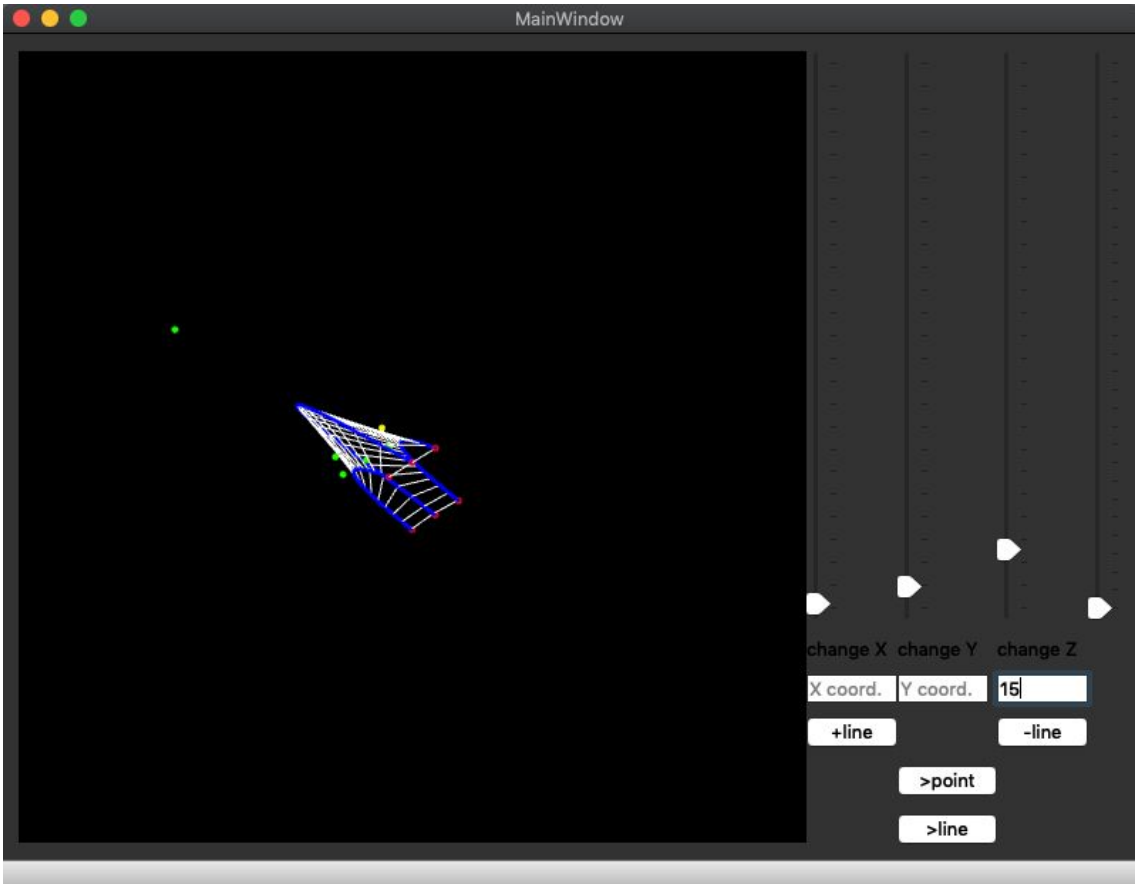
3) Повернем по OY



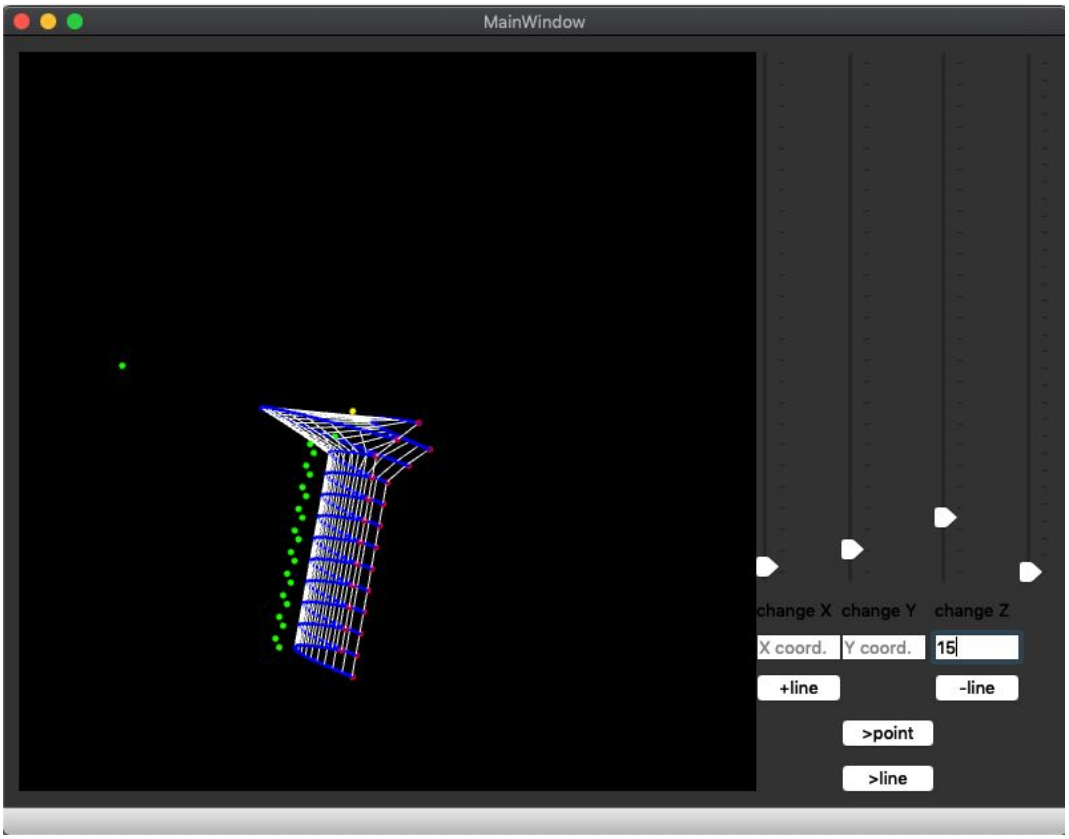
4) Поменяем выбор точки на середину второй линии и увеличим ее координату по OZ на 15



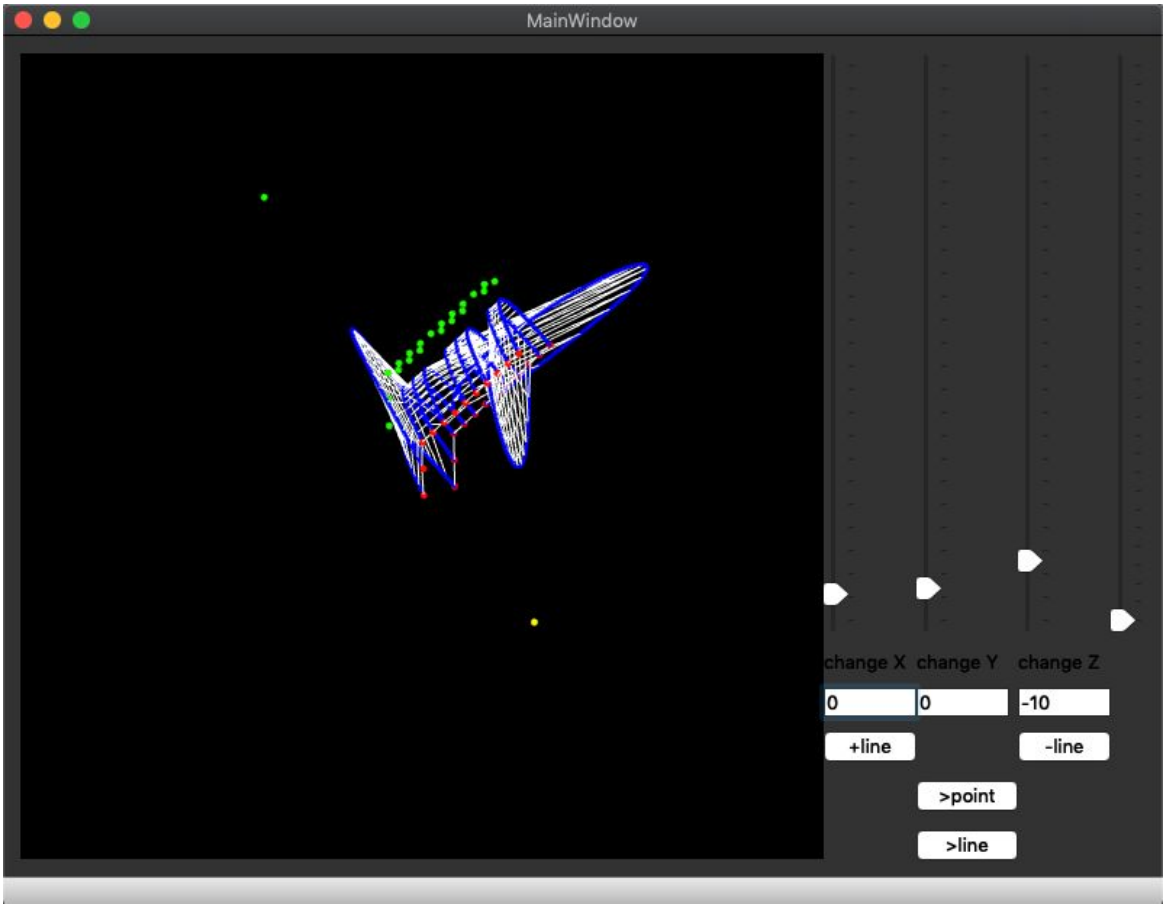
5) Немного развернем и удалим крайнюю четвертую линию



6) Добавим много линий



7) Поменяем координаты у двух случайных точек



Вывод

Исходя из всего курса “Компьютерная графика” этого семестра, хочется сделать вывод о том, что невозможно построить программно графический объект, будь то 2Д или 3Д, без знаний как базовой математики, так и линейной алгебры, математического анализа. Библиотека QT, которую я использовала для реализации данной работы и всех лабораторных, мне кажется не самой простой и удобной. Самые интересные и полезные функции можно найти, например, в OpenGL.

C++ выступает, как и всегда, языком быстрым. И используют его для графики в основном для ускорения работы, так как постоянные вычисления изменения координат и сама отрисовка - очень медленный процесс в принципе.