Московский Авиационный Институт
(Национальный Исследовательский Университет)

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

Лабораторная работа №7

по курсу “Компьютерная графика”

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

Москва, 2020

Построение плоских полиномиальных кривых

Задача:

Написать программу, строящую полиномиальную кривую по заданным точкам. Обеспечить возможность изменения позиции точек.

Вариант 4

Кривая Безье 3-й степени (с использованием стандартной функцией рисования кривой и без)

Описание

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

1) mainwindow - окно для отрисовки.
2) beziercurve - класс, реализующий создание опорных точек, перемещение их по экрану и отрисовку линии Безье по ним.

Кривая Безье N-1 степени задается уравнением

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

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

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

Без использования встроенной функции отрисовки кривых в классе beziercurve.cpp в функции рисования *paintEvent*(QPaintEvent *) это реализовано через обычный цикл for, рисующий элипсы

```
for (qreal t = 0; t < 1; t = t + 0.001) {
    qreal x = (1-t)*(1-t)*(1-t)*m_points[0].x() + 3*(1-t)*(1-t)*t*m_points[1].x() +
3*(1-t)*t*t*m_points[2].x() + t*t*t*m_points[3].x();
    qreal y = (1-t)*(1-t)*(1-t)*m_points[0].y() + 3*(1-t)*(1-t)*t*m_points[1].y() +
3*(1-t)*t*t*m_points[2].y() + t*t*t*m_points[3].y();
    QPointF xy(x, y);
    QBrush* brush = new QBrush(Qt::black);
    painter.setBrush(*brush);
    painter.drawEllipse(xy, POINT_RADIUS, POINT_RADIUS);
  }
```

С использованием библиотечного класса QPainterPath кривая рисуется очень просто с помощью задания пути через опорные точки, хранящиеся в массиве m_points:

```
QPainterPath path;
  path.moveTo(m_points[StartPoint]);
  path.cubicTo(m_points[ControlPoint1], m_points[ControlPoint2], m_points[EndPoint]);
  painter.drawPath(path);
```

Опорные точки можно перемещать с помощью мыши, наблюдая, как изменяется кривая.

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>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
  Q_OBJECT

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

```
private:
  Ui::MainWindow *ui;
  beziercurve* m_editor;
};

#endif // MAINWINDOW_H
```

mainwindow.cpp

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

#include <QGridLayout>

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

  QGridLayout *mainLayout = new QGridLayout;
  QWidget *centralWidget = new QWidget(this);
  m_editor = new beziercurve();
     mainLayout->addWidget(m_editor,0,0);
     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>

class beziercurve : public QWidget
{
  Q_OBJECT
public:
  explicit beziercurve(QWidget *parent = 0);
  ~beziercurve();
```

```
protected:
    void paintEvent(QPaintEvent *);
    void resizeEvent(QResizeEvent *);
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *);
    void mouseReleaseEvent(QMouseEvent *);

signals:

public slots:

private:
    enum PointIndices {
        StartPoint = 0,
        ControlPoint1 = 1,
        ControlPoint2 = 2,
        EndPoint = 3
    };

   //Functions
private:
    qreal distance(QPointF a, QPointF b) {
        QPointF diff = a - b;
        return sqrt(diff.x()*diff.x() + diff.y()*diff.y());
    }

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

    QPointF         m_points[4];
    QPen            m_pens[4];
    QBrush          m_brushes[4];
    Qt::GlobalColor m_colors[4];

    QPen        m_curvePen;

    bool        m_dragging;
    int         m_selectedPoint;
};

#endif // BEZIERCURVE_H
```

beziercurve.cpp

```
#include "beziercurve.h"
#include <QPainter>
#include <QMouseEvent>

beziercurve::beziercurve(QWidget *parent)
```

```
  : QWidget(parent)
  , m_curvePen(Qt::black)
  , m_dragging(false)
{
  m_curvePen.setWidth(2);

  m_colors[0] = Qt::yellow;
  m_colors[1] = Qt::red;
  m_colors[2] = Qt::red;
  m_colors[3] = Qt::yellow;

  for (int i = 0; i < NUM_POINTS; i++) {
     m_pens[i]   = QPen(m_colors[i]);
     m_brushes[i] = QBrush(m_colors[i]);
  }
}

beziercurve::~beziercurve()
{

}
void beziercurve::mousePressEvent(QMouseEvent *event)
{
  for(int i = 0; i < NUM_POINTS; i++) {
     if(distance(m_points[i], event->pos()) <= POINT_RADIUS) {
        m_selectedPoint = i;
        m_dragging = true;
        break;
     }
  }
}

void beziercurve::mouseMoveEvent(QMouseEvent *event)
{
  if(m_dragging) {
     m_points[m_selectedPoint] = event->pos();
  }
  update();
}

void beziercurve::mouseReleaseEvent(QMouseEvent *)
{
  m_dragging = false;
}

void beziercurve::resizeEvent(QResizeEvent *)
{
  m_points[StartPoint]    = QPointF(20, height() - 20);
  m_points[ControlPoint1] = QPointF(width() - 20, height() - 20);
  m_points[ControlPoint2] = QPointF(20, 20);
```

```
  m_points[EndPoint]      = QPointF(width() - 20, 20);
}

void beziercurve::paintEvent(QPaintEvent *)
{
  QPainter painter(this);

  painter.setRenderHint(QPainter::Antialiasing, true);

  painter.setPen(m_curvePen);
  for (qreal t = 0; t < 1; t = t + 0.001) {
     qreal x = (1-t)*(1-t)*(1-t)*m_points[0].x() + 3*(1-t)*(1-t)*t*m_points[1].x() +
3*(1-t)*t*t*m_points[2].x() + t*t*t*m_points[3].x();
     qreal y = (1-t)*(1-t)*(1-t)*m_points[0].y() + 3*(1-t)*(1-t)*t*m_points[1].y() +
3*(1-t)*t*t*m_points[2].y() + t*t*t*m_points[3].y();
     QPointF xy(x, y);
     //painter.setPen(m_pens[0]);
     QBrush* brush = new QBrush(Qt::black);
     painter.setBrush(*brush);
     painter.drawEllipse(xy, POINT_RADIUS, POINT_RADIUS);
  }
  for (int i = 0; i < NUM_POINTS; i++) {
     painter.setPen(m_pens[i]);
     painter.setBrush(m_brushes[i]);
     painter.drawEllipse(m_points[i], POINT_RADIUS, POINT_RADIUS);
  }
}
```

а с библиотечными функциями класс **beziercurveeditor**
beziercurveeditor.h

```
#ifndef BEZIERCURVEEDITOR_H
#define BEZIERCURVEEDITOR_H

#include <QPen>
#include <QBrush>
#include <QWidget>
#include <QtMath>

class BezierCurveEditor : public QWidget
{
  Q_OBJECT
public:
  explicit BezierCurveEditor(QWidget *parent = 0);
  ~BezierCurveEditor();

protected:
  void paintEvent(QPaintEvent *);
  void resizeEvent(QResizeEvent *);
  void mousePressEvent(QMouseEvent *event);
```

```
    void mouseMoveEvent(QMouseEvent *);
    void mouseReleaseEvent(QMouseEvent *);

signals:

public slots:

private:
    enum PointIndices {
        StartPoint = 0,
        ControlPoint1 = 1,
        ControlPoint2 = 2,
        EndPoint = 3
    };

    //Functions
private:
    qreal distance(QPointF a, QPointF b) {
        QPointF diff = a - b;
        return sqrt(diff.x()*diff.x() + diff.y()*diff.y());
    }

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

    QPointF         m_points[4];
    QPen            m_pens[4];
    QBrush          m_brushes[4];
    Qt::GlobalColor m_colors[4];

    QPen        m_curvePen;

    bool        m_dragging;
    int         m_selectedPoint;
};

#endif // BEZIERCURVEEDITOR_H
```

beziercurveeditor.cpp

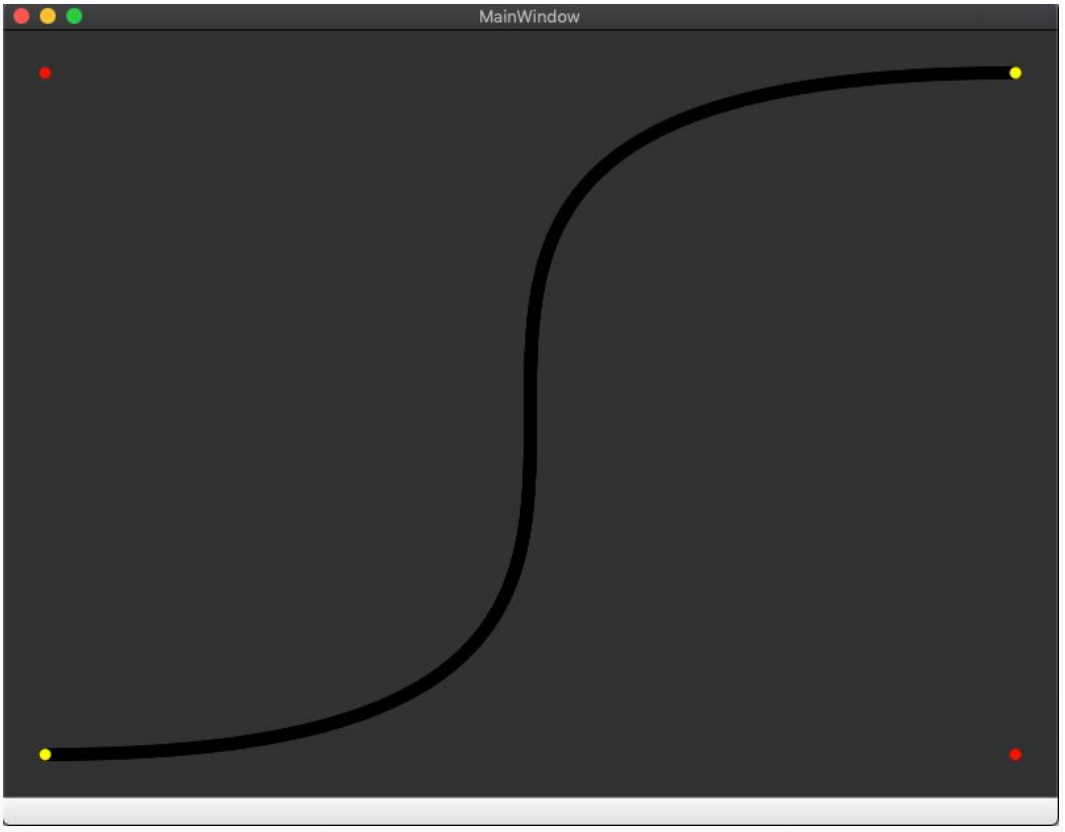
```
#include "beziercurveeditor.h"

#include <QPainter>
#include <QMouseEvent>

BezierCurveEditor::BezierCurveEditor(QWidget *parent)
    : QWidget(parent)
    , m_curvePen(Qt::black)
    , m_dragging(false)
```

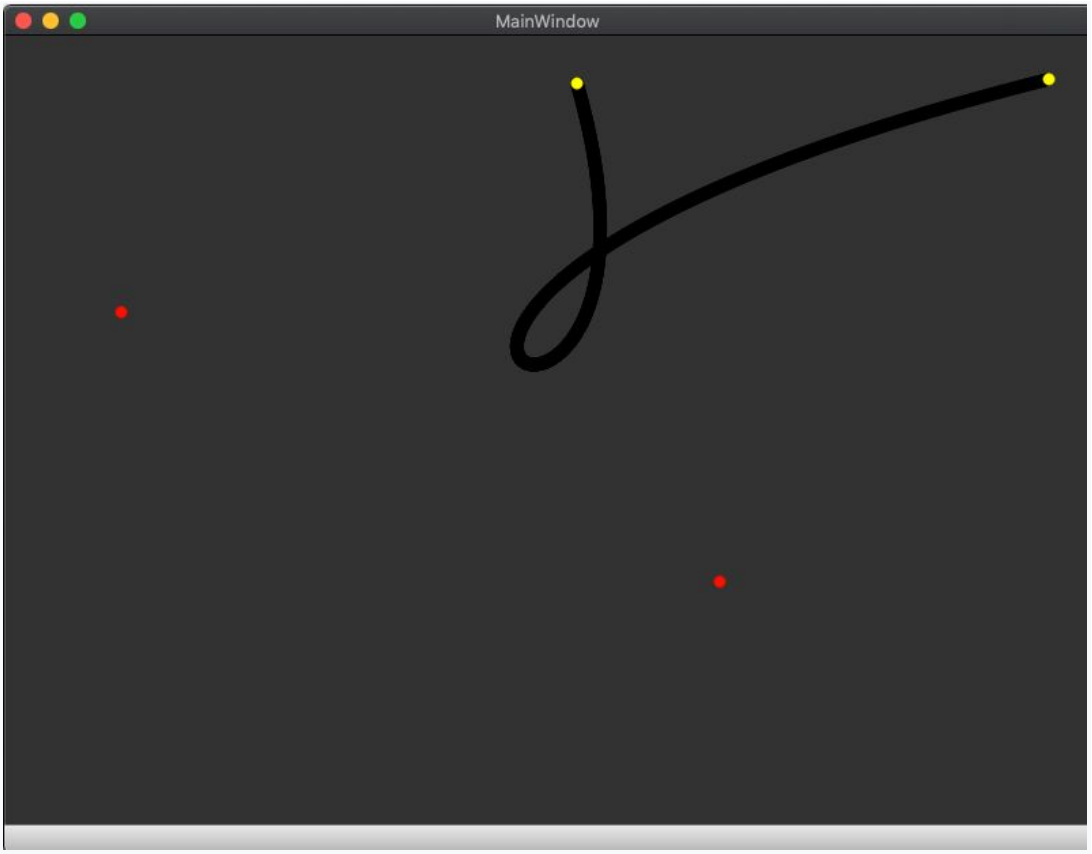
```
{
  m_curvePen.setWidth(2);

  m_colors[0] = Qt::yellow;
  m_colors[1] = Qt::red;
  m_colors[2] = Qt::red;
  m_colors[3] = Qt::yellow;

  for (int i = 0; i < NUM_POINTS; i++) {
     m_pens[i]    = QPen(m_colors[i]);
     m_brushes[i] = QBrush(m_colors[i]);
  }
}

BezierCurveEditor::~BezierCurveEditor()
{

}

void BezierCurveEditor::mousePressEvent(QMouseEvent *event)
{
  for(int i = 0; i < NUM_POINTS; i++) {
     if(distance(m_points[i], event->pos()) <= POINT_RADIUS) {
        m_selectedPoint = i;
        m_dragging = true;
        break;
     }
  }
}

void BezierCurveEditor::mouseMoveEvent(QMouseEvent *event)
{
  if(m_dragging) {
     m_points[m_selectedPoint] = event->pos();
  }
  update();
}

void BezierCurveEditor::mouseReleaseEvent(QMouseEvent *)
{
  m_dragging = false;
}

void BezierCurveEditor::resizeEvent(QResizeEvent *)
{
  m_points[StartPoint]    = QPointF(20, height() - 20);
  m_points[ControlPoint1] = QPointF(width() - 20, height() - 20);
  m_points[ControlPoint2] = QPointF(20, 20);
  m_points[EndPoint]      = QPointF(width() - 20, 20);
}
```

```
void BezierCurveEditor::paintEvent(QPaintEvent *)
{
  QPainter painter(this);

  painter.setRenderHint(QPainter::Antialiasing, true);

  painter.setPen(m_curvePen);
  QPainterPath path;
  path.moveTo(m_points[StartPoint]);
  path.cubicTo(m_points[ControlPoint1], m_points[ControlPoint2], m_points[EndPoint]);
  painter.drawPath(path);

  for (int i = 0; i < NUM_POINTS; i++) {
    painter.setPen(m_pens[i]);
    painter.setBrush(m_brushes[i]);
    painter.drawEllipse(m_points[i], POINT_RADIUS, POINT_RADIUS);
  }
}
```

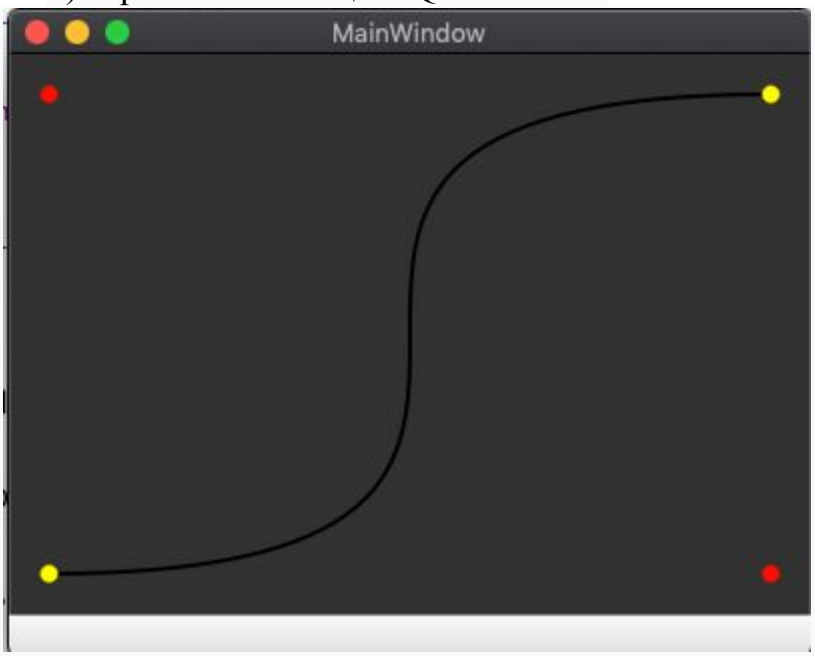
Пример работы обеих версий

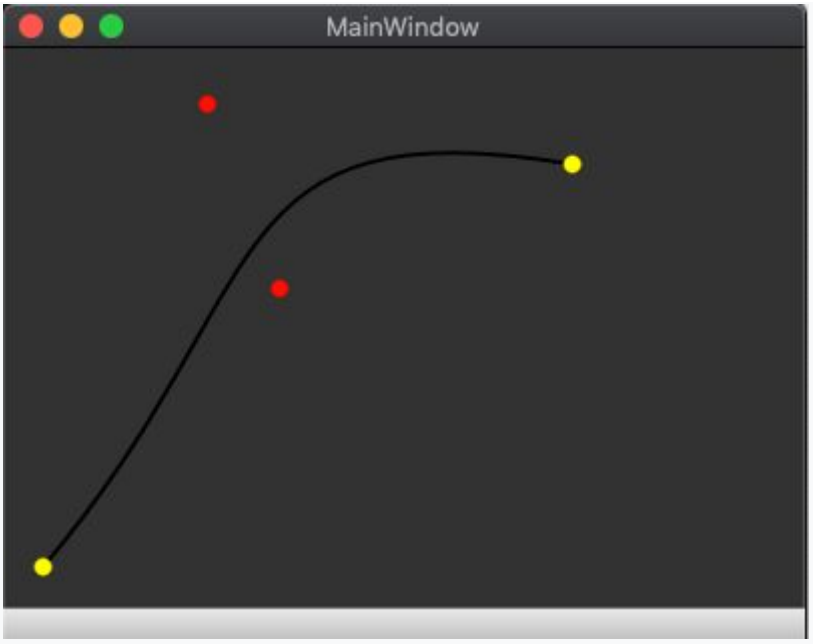
1) Точки эллипсы





2) Кривая с помощью QPainterPath





Вывод

Рисование кривых по точкам не сложнее, чем рисование прямых. Для отображения в 3д то же уравнение будет работать, но добавится еще координата по Z.