**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 804 «Теория вероятностей и компьютерное моделирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 7**

Тема: Построение плоских полиномиальных

кривых.

Студент: Шевчук П.В.

Группа: 80-304

Преподаватель: Чернышов Л.Н.

Дата: 11.12.18

Оценка:

Москва, 2018

**1. Постановка задачи**

Задание: Написать программу, строящую полиномиальную кривую по заданным точкам.

Вариант 15: B-сплайн. n = 6, k = 4. Узловой вектор равномерный.

**2. Решения задачи**

ЯП: C++

ОС: Linux Mint 18.2

Среда разработки: Qt Creator 4.7.2

Библиотеки: QOpenGLFunctions, QCustomPlot.

Выбранный язык удобен для решения задачи, так как он обладает удобными библиотеками для построения фигур и вывода их на экран.

Основные функции:

* Функция customPlot позволяет отрисовать один график по

нескольким точкам и за вертикальной линией будет следовать трассировщик, который будет вставать в ближайшие к точки графика от вертикальной линии.

* Функции initializeGL, resizeGL и paintGL перекрывают методы

родительского класса QGLWidget. Обработчики событий от мыши перекрывают обработчики, унаследованные от QWidget.

* В конструкторе вызывается QGLWidget::setFormat, чтобы

задать контекст устройства отображения OpenGL, и инициализируются приватные переменные-члены класса.

* Функция initializeGL вызывается один раз, перед вызовом

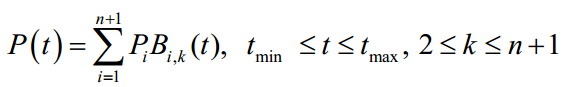
paintGL. Здесь выполняется настройка контекста отображения.

* Функция resizeGL вызывается один раз, перед paintGL, но после

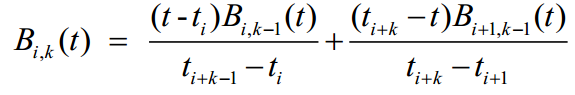
того, как будет вызвана функция initializeGL. Здесь настраивается область просмотра (viewport), проекция и прочие настройки, которые зависят от размера виджета.

Ход работы:

Реализован алгоритм Кокса-Де Бура для вычисления точек В-сплайна. B-сплайн задается уравнением:



- управляющие точки, - базисные функции B-сплайна, определяемые рекурсивными формулами Кокса-Де Бура:



где - элементы вектора узлов, которые образуют неубывающую последовательность, т.е. для всех . При вычислениях если и знаменатель оказываются одновременно равными нулю, то вся дробь полагается равной нулю.

**3. Руководство по использованию программы**

В окне вывода пользователь может менять положения точек, которые влияют на форму сплайна(рис.1).

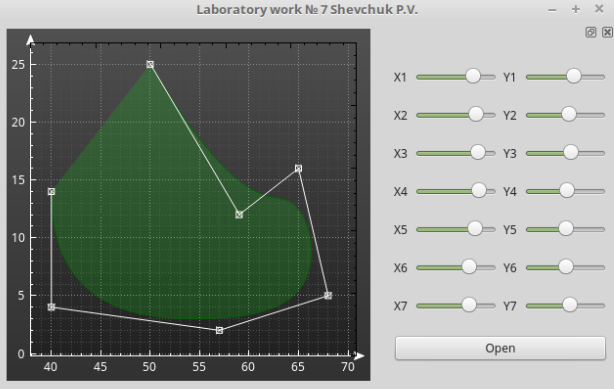


Рис 1. Значения точек по умолчанию

Пользователь может изменить фигуру на замкнутый сплайн(рис. 2). В таком случае первая и последняя точки не будут соединяться отрезком.

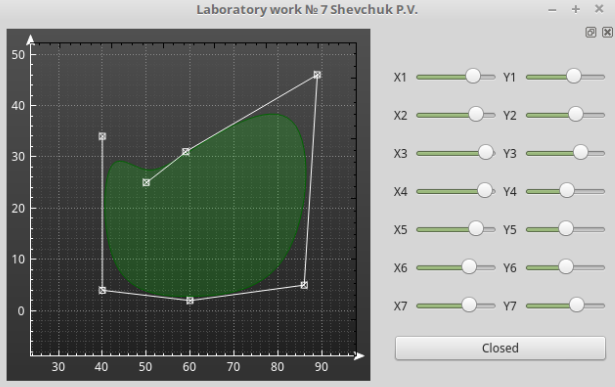


Рис 2. Замкнутый сплайн

**4. Листинг программы**

#include "view.h"

#include "panel.h"

#include <QOpenGLContext>

#include <QPainter>

#include <QtMath>

// сигнал

View::View(QWidget \*parent) : QWidget(parent), pan(nullptr) {

QVBoxLayout \*mainLayout = new QVBoxLayout;

customPlot = new QCustomPlot;

customPlot->rescaleAxes();

mainLayout->addWidget(customPlot);

this->setLayout(mainLayout);

n = 100;

degree = 3;

cv = {

qMakePair(50.f, 25.f), qMakePair(59.f, 12.f), qMakePair(65.f, 16.f),

qMakePair(68.f, 5.f), qMakePair(57.f, 2.f), qMakePair(40.f, 4.f),

qMakePair(40.f, 14.f)};

}

// инициализация панели

const Panel \*View::controlPanel() const { return pan; }

void View::setControlPanel(const Panel \*p) {

pan = p;

update();

}

// изменение точки вектора

void View::SetKnotVector() {

// создание диапазона значений

int count = 7;

u.resize(n);

knots.clear();

if (not pan->getPeriodic()) {

for (int i = 0; i < n; ++i) {

u[i] = 1.0f \* i / (n - 1) \* (count - degree);

}

for (int i = 0; i < degree; ++i) {

knots.push\_back(0);

}

for (int i = 0; i < count - degree + 1; ++i) {

knots.push\_back(i);

}

for (int i = 0; i < degree; ++i) {

knots.push\_back(count - degree);

}

} else {

for (int i = 0; i < n; ++i) {

u[i] = static\_cast<float>(

fmod(fmod((1.0f \* i / (n - 1) \* count) - (0.5f \* (degree - 1)),

count) +

count,

count));

}

for (int i = 0 - degree; i < count + degree + degree - 1; ++i) {

knots.push\_back(i);

}

}

}

// нарисовать сплайн

void View::DrawBSpline(QCustomPlot \*customPlot) {

customPlot->clearGraphs();

customPlot->clearItems();

customPlot->clearPlottables();

// создание empty curve объектов:

QCPCurve \*curve = new QCPCurve(customPlot->xAxis, customPlot->yAxis);

QVector<QPair<float, float>> samples(n, qMakePair(0.0f, 0.0f));

int count = 7;

for (int i = 0; i < n; ++i) {

if (not pan->getPeriodic()) {

if (static\_cast<int>(u[i]) == (count - degree)) {

samples[i] = cv.back();

} else {

for (int k = 0; k < count; ++k) {

float tmp = CoxDeBoor(u[i], k, degree);

samples[i].first += tmp \* cv[k].first;

samples[i].second += tmp \* cv[k].second;

}

}

} else {

for (int k = 0; k < count + degree; ++k) {

float tmp = CoxDeBoor(u[i], k, degree);

samples[i].first += tmp \* cv[k % count].first;

samples[i].second += tmp \* cv[k % count].second;

}

}

}

QCPCurve \*curve2 = new QCPCurve(customPlot->xAxis, customPlot->yAxis);

QVector<QCPCurveData> polygon(count);

for (int i = 0; i < count; ++i) {

polygon[i] = QCPCurveData(i, cv[i].first, cv[i].second);

}

// сгенирировать точки кривой:

QVector<QCPCurveData> plot(n);

for (int i = 0; i < n; ++i) {

plot[i] = QCPCurveData(i, samples[i].first, samples[i].second);

}

// передача данных кривым

curve->data()->set(plot, true);

// цвета кривых:

curve->setPen(QPen(QColor(0, 100, 0)));

curve->setBrush(QBrush(QColor(0, 255, 0, 40)));

curve2->data()->set(polygon, true);

curve2->setPen(QPen(Qt::white));

curve2->setScatterStyle(QCPScatterStyle::ssCrossSquare);

// установить ручки, кисти и фон:

customPlot->xAxis->setBasePen(QPen(Qt::white, 1));

customPlot->yAxis->setBasePen(QPen(Qt::white, 1));

customPlot->xAxis->setTickPen(QPen(Qt::white, 1));

customPlot->yAxis->setTickPen(QPen(Qt::white, 1));

customPlot->xAxis->setSubTickPen(QPen(Qt::white, 1));

customPlot->yAxis->setSubTickPen(QPen(Qt::white, 1));

customPlot->xAxis->setTickLabelColor(Qt::white);

customPlot->yAxis->setTickLabelColor(Qt::white);

customPlot->xAxis->grid()->setPen(QPen(QColor(140, 140, 140), 1, Qt::DotLine));

customPlot->yAxis->grid()->setPen(QPen(QColor(140, 140, 140), 1, Qt::DotLine));

customPlot->xAxis->grid()->setSubGridPen(QPen(QColor(80, 80, 80), 1, Qt::DotLine));

customPlot->yAxis->grid()->setSubGridPen(QPen(QColor(80, 80, 80), 1, Qt::DotLine));

customPlot->xAxis->grid()->setSubGridVisible(true);

customPlot->yAxis->grid()->setSubGridVisible(true);

customPlot->xAxis->grid()->setZeroLinePen(Qt::NoPen);

customPlot->yAxis->grid()->setZeroLinePen(Qt::NoPen);

customPlot->xAxis->setUpperEnding(QCPLineEnding::esSpikeArrow);

customPlot->yAxis->setUpperEnding(QCPLineEnding::esSpikeArrow);

QLinearGradient plotGradient;

plotGradient.setStart(0, 0);

plotGradient.setFinalStop(0, 350);

plotGradient.setColorAt(0, QColor(80, 80, 80));

plotGradient.setColorAt(1, QColor(50, 50, 50));

customPlot->setBackground(plotGradient);

QLinearGradient axisRectGradient;

axisRectGradient.setStart(0, 0);

axisRectGradient.setFinalStop(0, 350);

axisRectGradient.setColorAt(0, QColor(80, 80, 80));

axisRectGradient.setColorAt(1, QColor(30, 30, 30));

customPlot->axisRect()->setBackground(axisRectGradient);

// установка customPlot config:

customPlot->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom |

QCP::iSelectPlottables);

customPlot->axisRect()->setupFullAxesBox();

}

// Обновление значение через панель

void View::updateValues()

{

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

cv[i] = controlPanel()->getPoints(i);

}

}

void View::init()

{

SetKnotVector();

DrawBSpline(customPlot);

}

// алгоритм

float View::CoxDeBoor(float u, int k, int d) {

if (d == 0) {

if (knots[k] <= u and u < knots[k + 1]) {

return 1.f;

} else {

return 0.f;

}

}

float den1 = knots[k + d] - knots[k];

float den2 = knots[k + d + 1] - knots[k + 1];

float eq1 = 0;

float eq2 = 0;

if (den1 > 0) {

eq1 = ((u - knots[k]) / den1) \* CoxDeBoor(u, k, d - 1);

}

if (den2 > 0) {

eq2 = ((knots[k + d + 1] - u) / den2) \* CoxDeBoor(u, k + 1, d - 1);

}

return eq1 + eq2;

}

**5. Вывод**

Благодаря проделанной работе удалось разобраться не только в алгоритме построения B-сплайна, но и в функциях из встроенной библиотеки Qt, которые помогли разобраться с интерфейсом в режиме пользователя.

**Список литературы**

1.Использование QCustomPlot для создания анимированных графиков

[Электронный ресурс].

URL: <https://habr.com/post/266957/>

2. Кривые на плоскости [Лекция].