**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 7**

Тема: Построение плоских полиномиальных кривых.

Студент: Айрапетова Евгения Ашотовна

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

Написать программу, строящую полиномиальную кривую по заданным точкам. Обеспечить возможность изменения позиции точек и, при необходимости, значений касательных векторов и натяжения.

Вариант задания:

NURB-кривая. n = 5, k = 4. Узловой вектор равномерный. Веса точек различны и модифицируются.

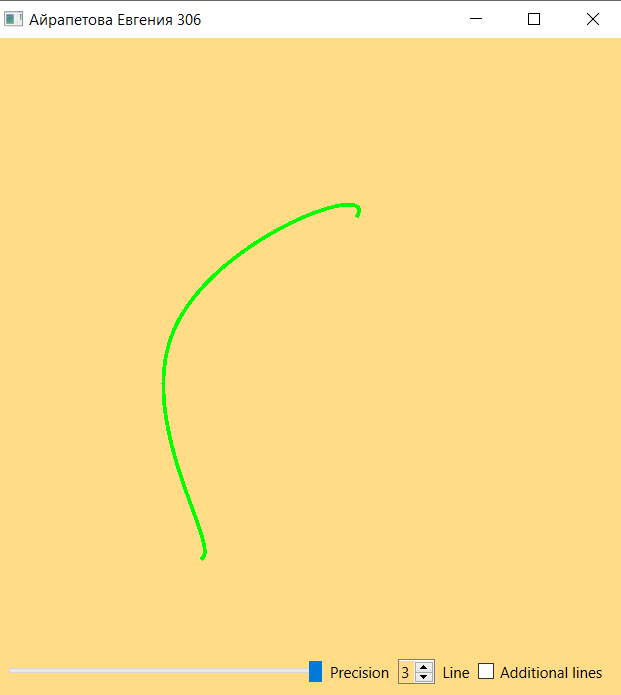
1. **Описание программы**

В программе я использовала фреймворк Qt, язык С++ и библиотеку OpenGL.

Для решения задачи были реализованы функции полинома, по которому отрисовывается кривая. Также были реализованы класс для отображения и обработки точек, класс для построения кривой и ее отображения. Само построение сплайна сводится к решению системы уравнений.

В интерфейс программы для более удобного использования были добавлены возможность регулировать аппроксимацию кривой, толщину линии кривой, а также возможность предпросмотра итоговой кривой без вспомогательных линий.

1. **Тесты**



1. Листинг программы

main.cpp:

// Айрапетова 306

#include"splineview.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

SplineView spl;

spl.setWindowTitle("Айрапетова Евгения 306");

spl.show();

return a.exec();

}

splineclass.h:

#ifndef SPLINECLASS\_H

#define SPLINECLASS\_H

#include<QList>

#include<QPointF>

#include<QObject>

#include<QPainter>

const int POINT\_RADIUS = 10;

class Polynom{

public:

float calculatePolynom(float t){

return a \* t \* t \* t + b \* t \* t + c \* t + d;

}

float a;

float b;

float c;

float d;

};

bool isCatch(int x, int y, int px, int py);

class SplinePoint : public QPointF{

QPointF rail = {0, 0};

bool israilnull = true;

public:

QPointF getRail(){

return rail;

}

QPointF toPoint() {

return (QPointF)(\*this);

}

void setRail(const QPointF& p){

israilnull = false;

rail = p - toPoint();

}

void delRail(){

israilnull = true;

}

bool isRailNull(){

return israilnull;

}

};

class SplineClass : public QObject

{

Q\_OBJECT

private:

QList<SplinePoint> points;

class SplineSegment{

public:

Polynom xPolynom;

Polynom yPolynom;

QPointF claculatePoint(float t){

return QPointF(xPolynom.calculatePolynom(t), yPolynom.calculatePolynom(t));

}

};

int presicion = 50;

public:

void calculatePolynoms(Polynom& fP, Polynom& sP, int p0, int p1, int p2, int r0, int r2);

void calculate();

SplineSegment fSegment;

SplineSegment sSegment;

SplineClass();

QList<SplinePoint>& getPoints();

bool add(int x, int y);

bool add(const QPointF&);

void clear();

bool isValid();

QList<SplinePoint>::iterator findPoint(int x, int y);

QList<SplinePoint>::iterator findPoint(const QPointF& p);

QList<SplinePoint>::iterator findPointOrRail(const QPointF& p);

QList<SplinePoint>::iterator findPointOrRail(int x, int y);

int getPresicion();

void setPresicion(int presiscion);

public slots:

signals:

void lineChanged();

};

#endif // SPLINECLASS\_H

splineclass.cpp:

#include "splineclass.h"

#include <QMessageBox>

#include <QtMath>

#include <QDebug>

SplineClass::SplineClass()

{

}

bool isCatch(int x, int y, int px, int py){

return (abs(x - px) < POINT\_RADIUS && abs(y -py) < POINT\_RADIUS);

}

QList<SplinePoint>::iterator SplineClass::findPoint(const QPointF& p){

return findPoint(p.x(), p.y());

}

QList<SplinePoint>::iterator SplineClass::findPoint(int x, int y){

QList<SplinePoint>::iterator i = points.begin();

for(; i != points.end(); ++i){

if(isCatch(x, y, i->x(), i->y())){

return i;

}

}

return i;

}

QList<SplinePoint>::iterator SplineClass::findPointOrRail(int x, int y){

QList<SplinePoint>::iterator i = findPoint(x, y);

if(i == points.end()){

i = points.begin();

for(; i != points.end(); ++i){

if(!i->isRailNull()){

if(isCatch(x, y, i->x() + i->getRail().x(),i->y() + i->getRail().y())){

return i;

}

}

}

}

return i;

}

QList<SplinePoint>::iterator SplineClass::findPointOrRail(const QPointF& p){

return findPointOrRail(p.x(), p.y());

}

QList<SplinePoint>& SplineClass::getPoints() {

return points;

}

bool SplineClass::add(const QPointF& p){

if(points.size() < 3){

if(findPoint(p.x(), p.y()) == points.end()){

SplinePoint t;

t.setX(p.x());

t.setY(p.y());

points.push\_back(t);

emit lineChanged();

return true;

}

}

return false;

}

void SplineClass::clear(){

points.clear();

emit lineChanged();

}

bool SplineClass::isValid(){

return points.size() == 3 && !points[0].isRailNull() && !points[2].isRailNull();

}

void SplineClass::calculatePolynoms(Polynom& fP, Polynom& sP, int p0, int p1, int p2, int r0, int r2){

float& a1 = fP.a;

float& a2 = sP.a;

float& b1 = fP.b;

float& b2 = sP.b;

float& c1 = fP.c;

float& c2 = sP.c;

float& d1 = fP.d;

float& d2 = sP.d;

//qDebug() << p0 << ' ' << p1 << ' '<< p2 << ' '<<r0 << ' '<< r2 << '\n';

d1 = p0; //+

d2 = p1; //+

c1 = r0;

float n = -c1;

float h = d2-d1-c1;

float q = p2 - d2;

float z = r2;

float x = 3\*q + 2\*n - z;

a1 = (x-5\*h)/4;

b1 = (9\*h - x)/4;

b2 = 3\*a1 + b1;

c2 = 3\*a1 + 2\*b1 - n;

a2 = q - b2 - c2;

}

void SplineClass::calculate(){

calculatePolynoms(fSegment.xPolynom,

sSegment.xPolynom,

points[0].x(), points[1].x(),

points[2].x(), points[0].getRail().x(), points[2].getRail().x());

calculatePolynoms(fSegment.yPolynom,

sSegment.yPolynom,

points[0].y(), points[1].y(),

points[2].y(), points[0].getRail().y(), points[2].getRail().y());

}

int SplineClass::getPresicion(){

return presicion;

}

void SplineClass::setPresicion(int p){

presicion = p;

emit lineChanged();

}

splineview.h

#ifndef SPLINEVIEW\_H

#define SPLINEVIEW\_H

#include "splinewidget.h"

#include <QWidget>

#include <QVBoxLayout>

#include <QSlider>

#include <QCheckBox>

#include <QSpinBox>

class SplineView : public QWidget

{

Q\_OBJECT

public:

QHBoxLayout\* hLayout;

SplineWidget\* splw;

QVBoxLayout\* vLayout;

QSlider\* pSlider;

QCheckBox\* isSubLinesWathched;

QSpinBox\* lineWidth;

void keyPressEvent(QKeyEvent\* e) override;

SplineView();

};

#endif // SPLINEVIEW\_H

splineview.cpp:

#include "splineview.h"

#include <QLabel>

SplineView::SplineView() {

this->setStyleSheet("background-color:rgb(255,220 ,133);");

vLayout = new QVBoxLayout(this);

hLayout = new QHBoxLayout();

pSlider = new QSlider(Qt::Horizontal);

isSubLinesWathched = new QCheckBox();

lineWidth = new QSpinBox();

splw = new SplineWidget(this);

vLayout->addWidget(splw,1);

vLayout->addLayout(hLayout);

hLayout->addWidget(pSlider);

hLayout->addWidget(new QLabel("Precision"));

hLayout->addWidget(lineWidth);

hLayout->addWidget(new QLabel("Line"));

hLayout->addWidget(isSubLinesWathched);

pSlider->setSingleStep(1);

pSlider->setRange(1, 40);

pSlider->setValue(splw->spline.getPresicion());

isSubLinesWathched->setChecked(true);

isSubLinesWathched->setText("Additional lines");

lineWidth->setSingleStep(1);

lineWidth->setRange(1,9);

lineWidth->setValue(splw->splineWidth);

QObject::connect(pSlider, SIGNAL(valueChanged(int)), splw, SLOT(rePaint(int)));

QObject::connect(isSubLinesWathched, SIGNAL(stateChanged(int)), splw, SLOT(rePaintSubLineSettings(int)));

QObject::connect(lineWidth, SIGNAL(valueChanged(int)), splw, SLOT(rePaintAndSetSplineWidth(int)));

}

void SplineView::keyPressEvent(QKeyEvent\* e){

splw->keyPressEvent(e);

}

**Вывод**

Для выполнения данной лабораторной работы я изучила такую функцию, как сплайн, а также закрепила свои навыки в работе с библиотекой OpenGL.

**ЛИТЕРАТУРА**

Документация по Qt: https://doc.qt.io (дата обращения 25.12.2021).