**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Курсовой проект**

**по дисциплине «Компьютерная графика»**

Студент: Пищик Евгений Сергеевич

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

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. Для визуализации параметрических линий поверхности разрешается использовать только функции отрисовки отрезков в экранных координатах.

**Вариант:**

3. Линейчатая поверхность (направляющие – Cardinal Spline 3D)

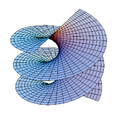
1. Описание программы

Программа отрисовывает линейчатую поверхность по двум направляющим, которые представляют собой Cardinal Spline 3D. Пользователь может задать координаты четырех точек для одной кривой и аналогично для другой, также пользователь может поворачивать фигуру вдоль осей X Y Z, менять точность аппроксимации, приближать/отдалять фигуру. Интерфейс программы реализован при помощи библиотеки Qt.

1. Линейчатая поверхность

Поверхность, образованная движением прямой линии. Прямые, принадлежащие этой поверхности, называются прямолинейными образующими, а каждая кривая, пересекающая все прямолинейные образующие, направляющей кривой.

Примеры линейчатых поверхностей:



линейчатый геликоид



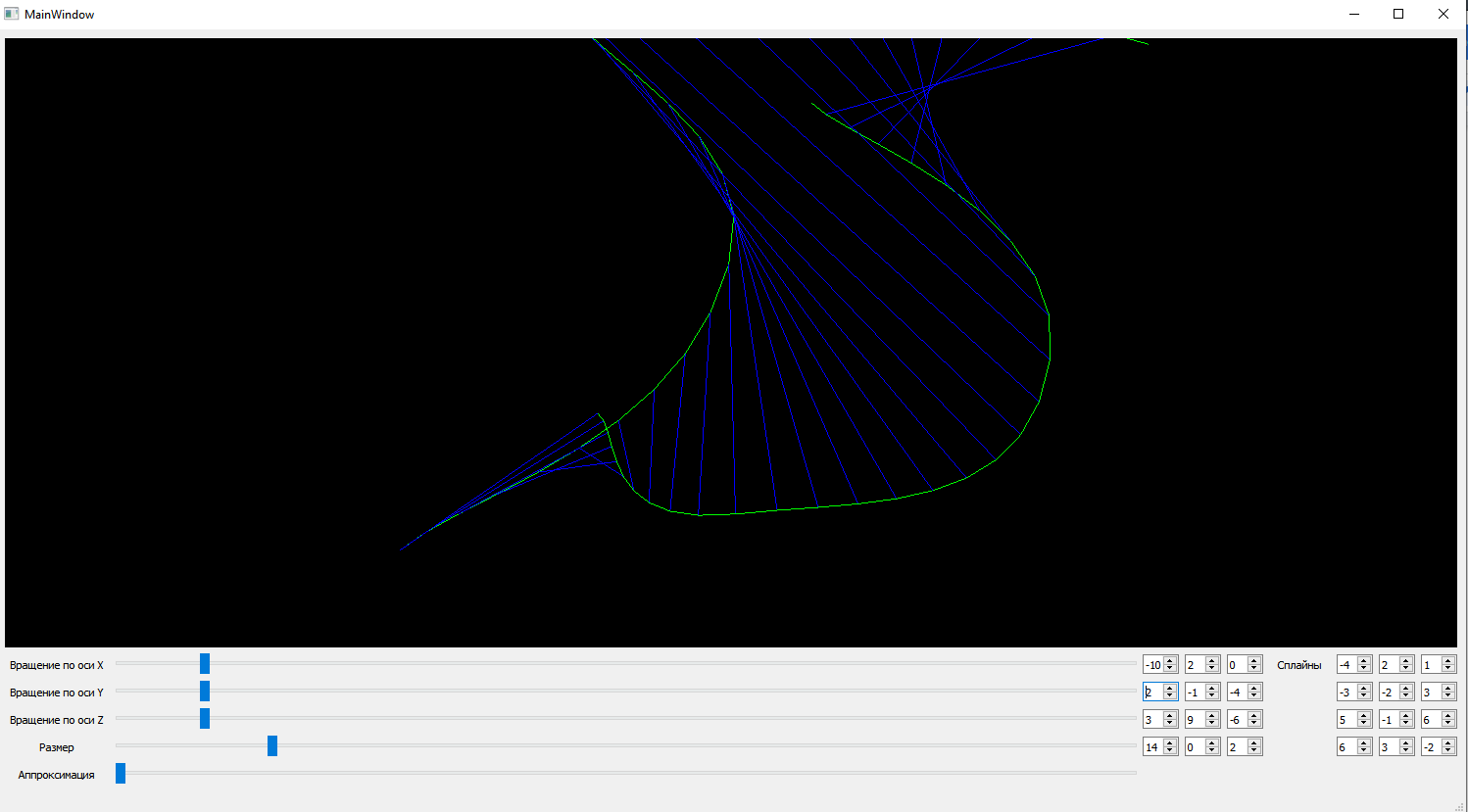
линейчатый гиперболоид

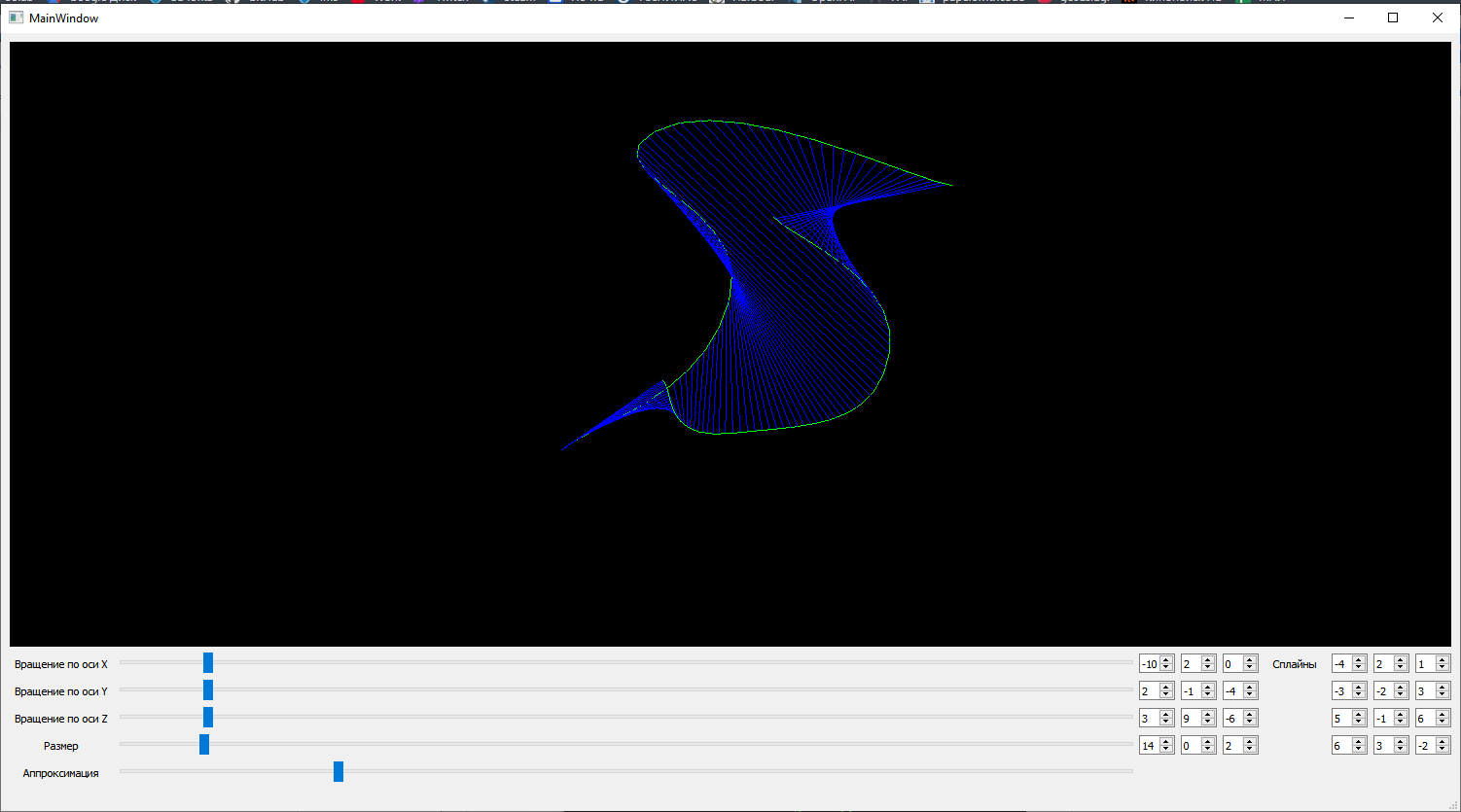
1. Кубический сплайн

Кубический сплайн — гладкая функция, область определения которой разбита на конечное число отрезков, на каждом из которых она совпадает с некоторым кубическим полиномом. Максимальная из степеней использованных полиномов называется степенью сплайна. Разность между степенью сплайна и получившейся гладкостью называется дефектом сплайна. Например, непрерывная ломаная есть сплайн степени 1 и дефекта 1. В современном понимании сплайны — это решения многоточечных краевых задач сеточными методами.

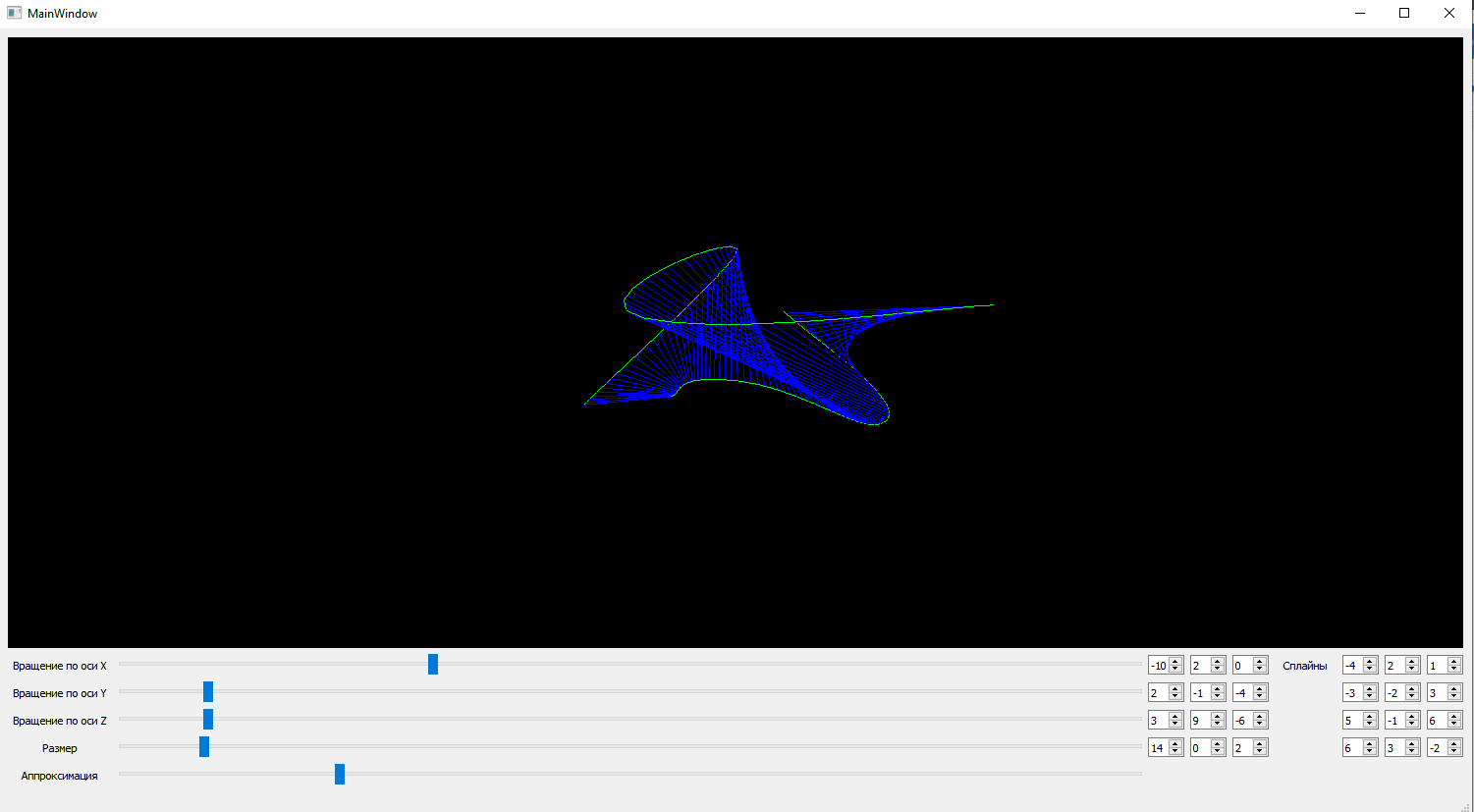
Другими словами сплайн — это кусочно заданная функция, то есть совокупность нескольких функций, каждая из которых задана на каком-то множестве значений аргумента, причем эти множества попарно непересекающиеся.

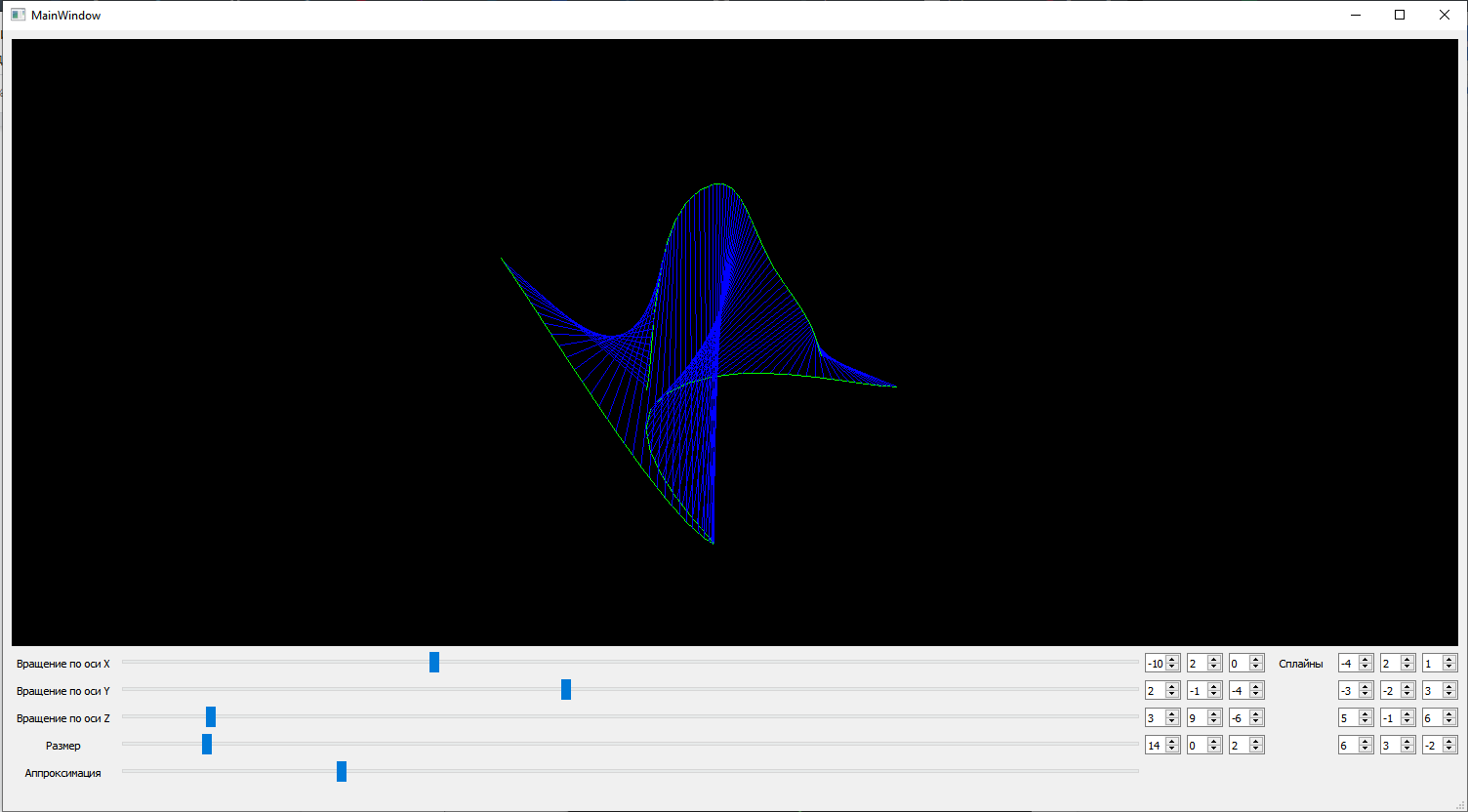
1. Набор тестов и результаты их выполнения.

Дефолтные значения.

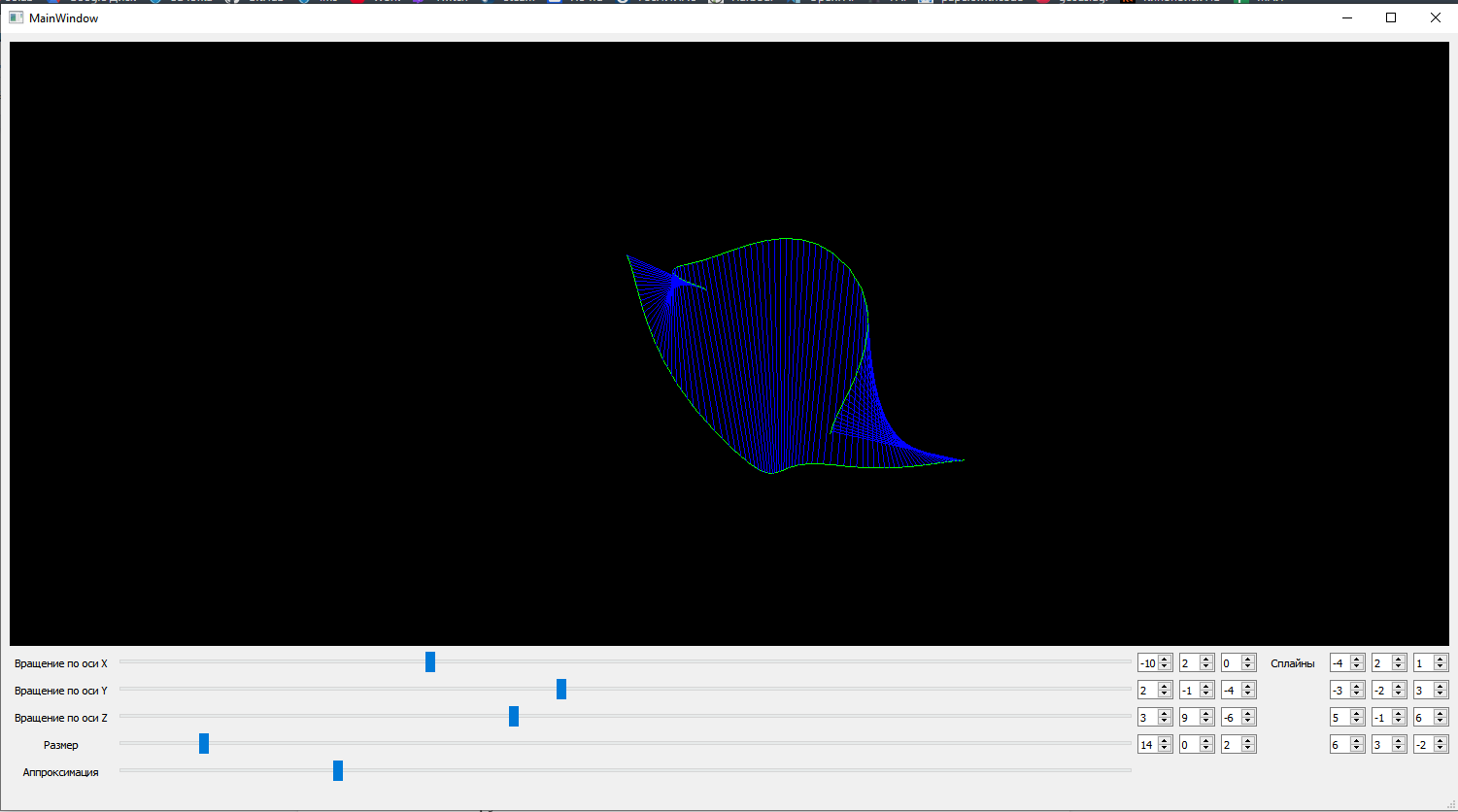
Увеличение точности аппроксимации и уменьшение размера фигуры.

Поворот по оси X.

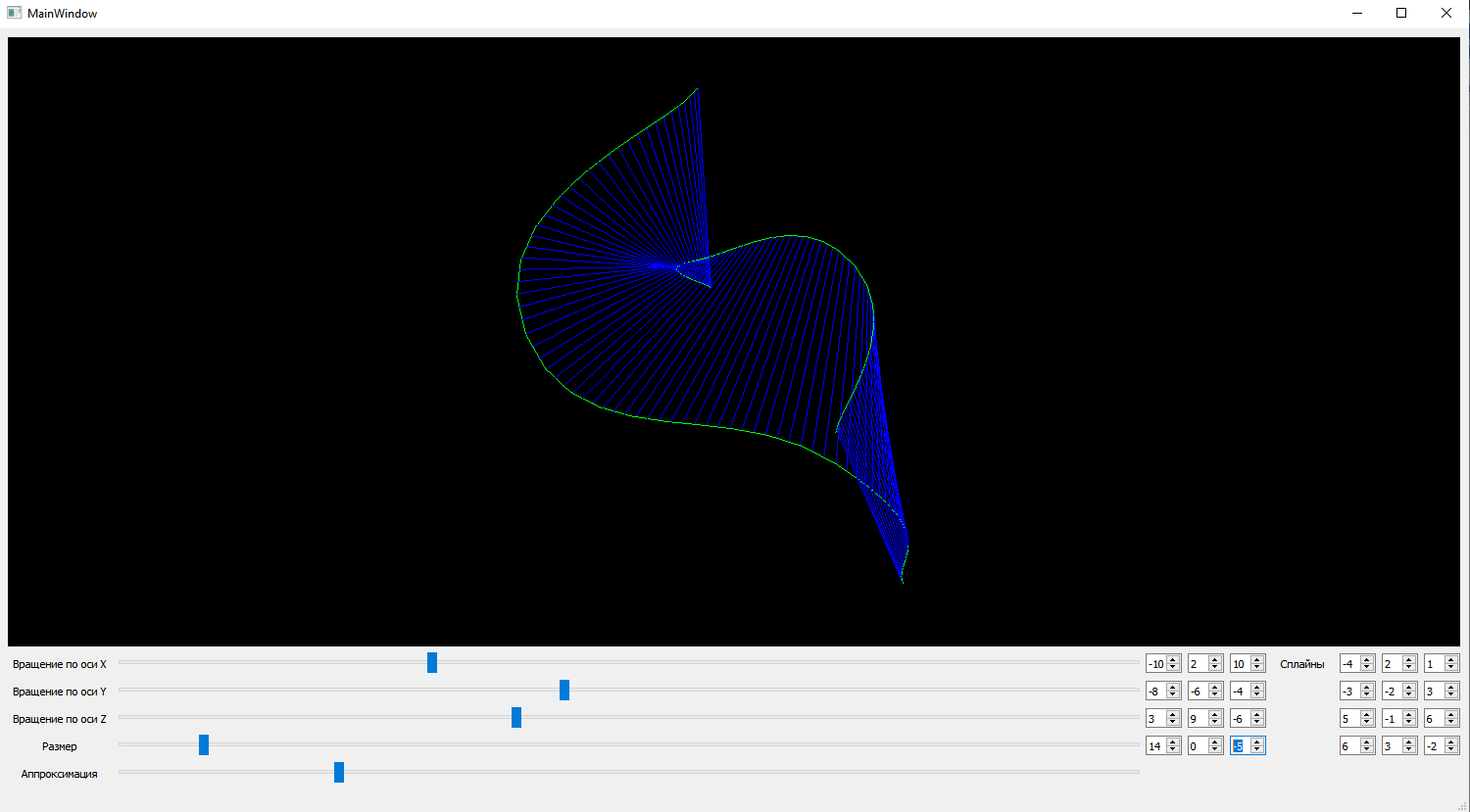
Поворот по оси Y.



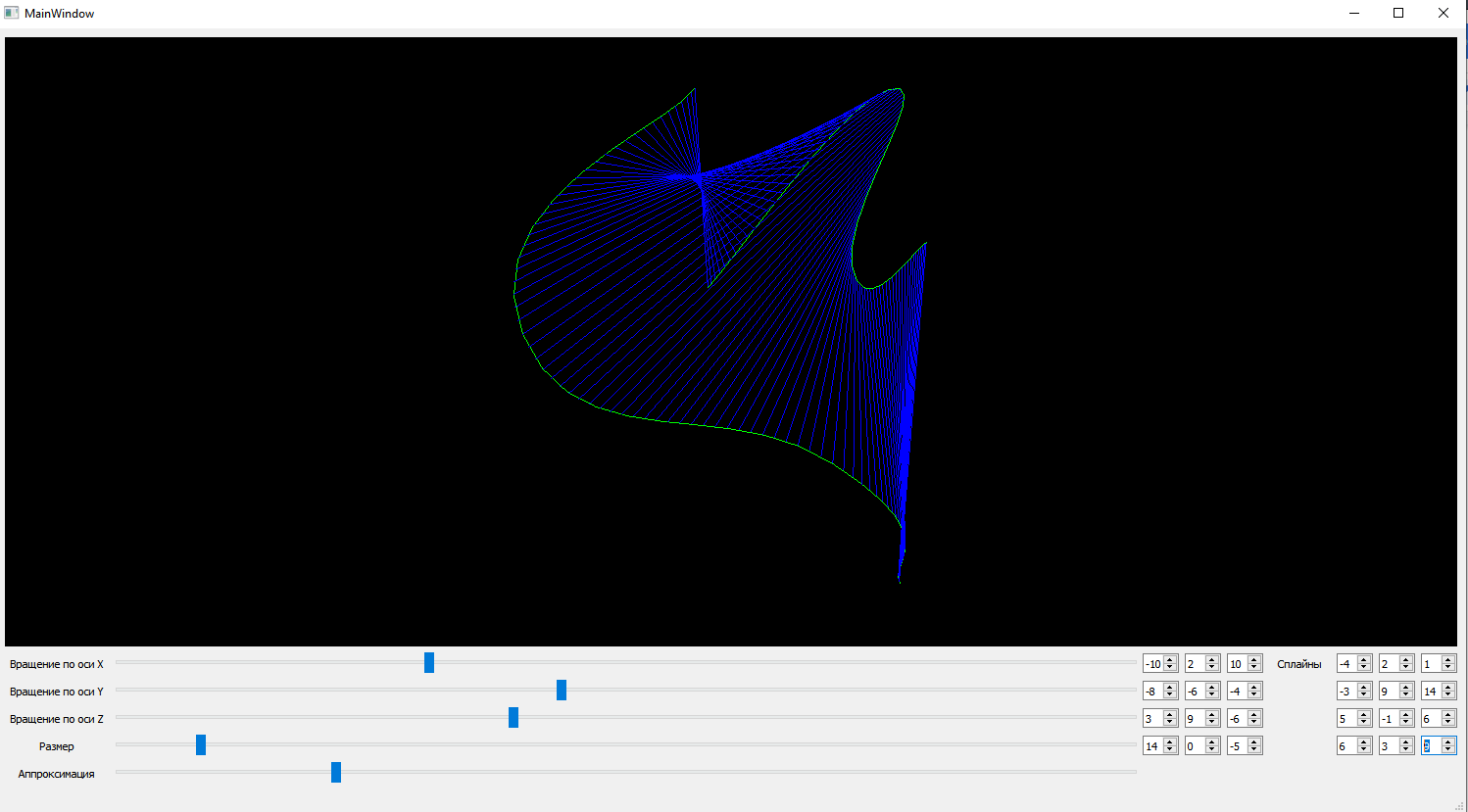
Поворот по оси Z.



Изменение координат точек первой кривой.



Изменение координат точек второй кривой.



1. Листинг программы

main.cpp

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication app(argc, argv);

MainWindow win;

win.show();

return app.exec();

}

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent): QMainWindow(parent), ui(new Ui::MainWindow)

{

ui->setupUi(this);

connect(ui->x\_rot, SIGNAL(valueChanged(int)), ui->widget, SLOT(setXRot(int)));

connect(ui->y\_rot, SIGNAL(valueChanged(int)), ui->widget, SLOT(setYRot(int)));

connect(ui->z\_rot, SIGNAL(valueChanged(int)), ui->widget, SLOT(setZRot(int)));

connect(ui->size, SIGNAL(valueChanged(int)), ui->widget, SLOT(setParam(int)));

connect(ui->apro, SIGNAL(valueChanged(int)), ui->widget, SLOT(setApro(int)));

}

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

mainwindow.h

#pragma once

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget\* parent = nullptr);

~MainWindow();

private:

Ui::MainWindow \*ui;

};

myopenglwidget.cpp

#include "myopenglwidget.h"

MyOpenGLWidget::MyOpenGLWidget(QWidget\* parent) : QOpenGLWidget{parent} {}

void MyOpenGLWidget::setParam(int param)

{

if((GLfloat)param != c \* 100)

{

c = (GLfloat)param / 100;

emit paramChanged(param);

update();

}

}

void MyOpenGLWidget::setApro(int apro)

{

if(p != apro)

{

p = apro;

update();

}

}

void MyOpenGLWidget::setxa1(int x)

{

if(x != xa1)

{

xa1 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setya1(int x)

{

if(x != ya1)

{

ya1 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setza1(int x)

{

if(x != za1)

{

za1 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setxa2(int x)

{

if(x != xa2)

{

xa2 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setya2(int x)

{

if(x != ya2)

{

ya2 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setza2(int x)

{

if(x != za2)

{

za2 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setxa3(int x)

{

if(x != xa3)

{

xa3 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setya3(int x)

{

if(x != ya3)

{

ya3 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setza3(int x)

{

if (x != za3) {

za3 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setxa4(int x)

{

if(x != xa4)

{

xa4 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setya4(int x)

{

if(x != ya4)

{

ya4 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setza4(int x)

{

if(x != za4)

{

za4 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setxb1(int x)

{

if(x != xb1)

{

xb1 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setyb1(int x)

{

if(x != yb1)

{

yb1 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setzb1(int x)

{

if(x != zb1)

{

zb1 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setxb2(int x)

{

if(x != xb2)

{

xb2 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setyb2(int x)

{

if(x != yb2)

{

yb2 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setzb2(int x)

{

if(x != zb2)

{

zb2 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setxb3(int x)

{

if(x != xb3)

{

xb3 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setyb3(int x)

{

if(x != yb3)

{

yb3 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setzb3(int x)

{

if(x != zb3)

{

zb3 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setxb4(int x)

{

if(x != xb4)

{

xb4 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setyb4(int x)

{

if(x != yb4)

{

yb4 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::setzb4(int x)

{

if(x != zb4)

{

zb4 = x;

calcInit();

update();

}

}

void MyOpenGLWidget::calcInit()

{

my\_point p1, p2, p3, p4;

p1.x = xa1; p1.y = ya1; p1.z = za1;

p2.x = xa2; p2.y = ya2; p2.z = za2;

p3.x = xa3; p3.y = ya3; p3.z = za3;

p4.x = xa4; p4.y = ya4; p4.z = za4;

my\_point po1, po2, po3, po4;

po1.x = xb1; po1.y = yb1; po1.z = zb1;

po2.x = xb2; po2.y = yb2; po2.z = zb2;

po3.x = xb3; po3.y = yb3; po3.z = zb3;

po4.x = xb4; po4.y = yb4; po4.z = zb4;

QVector<my\_point> pts1;

pts1.push\_back(p1);

pts1.push\_back(p2);

pts1.push\_back(p3);

pts1.push\_back(p4);

QVector<my\_point> pts2;

pts2.push\_back(po1);

pts2.push\_back(po2);

pts2.push\_back(po3);

pts2.push\_back(po4);

spline spln1(pts1);

spline spln2(pts2);

spln1.calc();

spln2.calc();

spl1 = spln1;

spl2 = spln2;

}

void MyOpenGLWidget::setXRot(int xRot)

{

if(xRot != xRotation)

{

xRotation = xRot;

emit xRotChanged(xRot);

update();

}

}

void MyOpenGLWidget::setYRot(int yRot)

{

if(yRot != yRotation)

{

yRotation = yRot;

emit yRotChanged(yRot);

update();

}

}

void MyOpenGLWidget::setZRot(int zRot)

{

if(zRot != zRotation)

{

zRotation = zRot;

emit zRotChanged(zRot);

update();

}

}

void MyOpenGLWidget::initializeGL()

{

glClearColor(0, 0, 0, 1);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_CULL\_FACE);

calcInit();

}

void MyOpenGLWidget::resizeGL(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

GLfloat x = (GLfloat)w / h;

if (x > 1) { glFrustum(-x, x, -1.0, 1.0, 4.0, 15.0); }

else { glFrustum(-1, 1, -1 / x, 1 / x, 4.0, 15.0); }

glMatrixMode(GL\_MODELVIEW);

}

void MyOpenGLWidget::drawSurface(spline &s1, spline &s2)

{

s1.draw();

s2.draw();

GLfloat x1,y1,z1,x2,y2,z2;

glColor3f(0.0f, 0.0f, 1.0f);

glBegin(GL\_LINES);

GLfloat step = (GLfloat)1 / p;

for(int i =0; i < s1.segments.size(); ++i)

{

for (GLfloat t = 0.0; t < 1.0; t += step) {

x1 = s1.get\_f(s1.segments[i].x\_pol[0], s1.segments[i].x\_pol[1], s1.segments[i].x\_pol[2], s1.segments[i].x\_pol[3], t);

y1 = s1.get\_f(s1.segments[i].y\_pol[0], s1.segments[i].y\_pol[1], s1.segments[i].y\_pol[2], s1.segments[i].y\_pol[3], t);

z1 = s1.get\_f(s1.segments[i].z\_pol[0], s1.segments[i].z\_pol[1], s1.segments[i].z\_pol[2], s1.segments[i].z\_pol[3], t);

glVertex3f(x1, y1, z1);

x2 = s1.get\_f(s2.segments[i].x\_pol[0], s2.segments[i].x\_pol[1], s2.segments[i].x\_pol[2], s2.segments[i].x\_pol[3], t);

y2 = s1.get\_f(s2.segments[i].y\_pol[0], s2.segments[i].y\_pol[1], s2.segments[i].y\_pol[2], s2.segments[i].y\_pol[3], t);

z2 = s1.get\_f(s2.segments[i].z\_pol[0], s2.segments[i].z\_pol[1], s2.segments[i].z\_pol[2], s2.segments[i].z\_pol[3], t);

glVertex3f(x2, y2, z2);

}

}

glEnd();

}

void MyOpenGLWidget::paintGL()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

glTranslatef(0.0, 0.0, -10.0);

glRotatef(xRotation, 1.0, 0.0, 0.0);

glRotatef(yRotation, 0.0, 1.0, 0.0);

glRotatef(zRotation, 0.0, 0.0, 1.0);

glScalef(c,c,c);

drawSurface(spl1, spl2);

}

myopenglwidget.h

#pragma once

#include <QOpenGLWidget>

#include <QGLWidget>

#include <QtWidgets>

#include <QtOpenGL>

#include <QOpenGLFunctions>

#include <QtMath>

#include "spline.h"

class MyOpenGLWidget : public QOpenGLWidget

{

Q\_OBJECT

public:

MyOpenGLWidget(QWidget \*parent = nullptr);

public slots:

void setXRot(int xRot);

void setYRot(int yRot);

void setZRot(int zRot);

void setParam(int param);

void setApro(int apro);

void setxa1(int x);

void setya1(int x);

void setza1(int x);

void setxa2(int x);

void setya2(int x);

void setza2(int x);

void setxa3(int x);

void setya3(int x);

void setza3(int x);

void setxa4(int x);

void setya4(int x);

void setza4(int x);

void setxb1(int x);

void setyb1(int x);

void setzb1(int x);

void setxb2(int x);

void setyb2(int x);

void setzb2(int x);

void setxb3(int x);

void setyb3(int x);

void setzb3(int x);

void setxb4(int x);

void setyb4(int x);

void setzb4(int x);

signals:

void xRotChanged(int xRot);

void yRotChanged(int yRot);

void zRotChanged(int zRot);

void paramChanged(int param);

protected:

void calcInit();

virtual void initializeGL();

virtual void resizeGL(int w, int h);

void drawSurface(spline& s1, spline& s2);

virtual void paintGL();

private:

GLfloat c = 0.3;

int xRotation = 30;

int yRotation = 30;

int zRotation = 30;

int p = 10;

spline spl1;

spline spl2;

GLfloat xa1=-10, xa2=2, xa3=4, xa4=14, xb1=-4, xb2=-3, xb3=5, xb4=6;

GLfloat ya1=2, ya2=-1, ya3=9, ya4=0, yb1=2, yb2=-2, yb3=-1, yb4=3;

GLfloat za1=0, za2=-4, za3=-6, za4=2, zb1=1, zb2=3, zb3=7, zb4=-2;

};

spline.cpp

#include "spline.h"

#include <QDebug>

spline::spline() {}

spline::spline(QVector<my\_point>& pnts)

{

points = pnts;

segments.resize(points.size() - 1);

for(int i = 0; i < segments.size(); ++i)

{

segments[i].x\_pol.resize(4);

segments[i].y\_pol.resize(4);

segments[i].z\_pol.resize(4);

}

}

GLfloat spline::get\_f(GLfloat a, GLfloat b, GLfloat c, GLfloat d, GLfloat t) { return qPow(t, 3) \* a + qPow(t, 2) \* b + t \* c + d; }

void spline::calc()

{

pre\_calc();

for(int i = 0; i < (int)segments.size() - 1; ++i)

{

calc\_pol(segments[i].x\_pol, segments[i + 1].x\_pol);

calc\_pol(segments[i].y\_pol, segments[i + 1].y\_pol);

calc\_pol(segments[i].z\_pol, segments[i + 1].z\_pol);

}

post\_calc();

}

void spline::calc\_pol(QVector<GLfloat>& f\_p, QVector<GLfloat>& s\_p)

{

GLfloat m = s\_p[3] - f\_p[3] - f\_p[2];

GLfloat q = s\_p[2] - f\_p[2];

f\_p[0] = q - 2 \* m;

f\_p[1] = 3 \* m - q;

}

void spline::pre\_calc()

{

segments[0].x\_pol[2] = k\*(points[1].x-points[0].x)/2;

segments[0].y\_pol[2] = k\*(points[1].y-points[0].y)/2;

segments[0].z\_pol[2] = k\*(points[1].z-points[0].z)/2;

for(int i = 1; i < (int)segments.size(); ++i)

{

segments[i].x\_pol[2] = k \* (points[i+1].x-points[i-1].x);

segments[i].y\_pol[2] = k \* (points[i+1].y-points[i-1].y);

segments[i].z\_pol[2] = k \* (points[i+1].z-points[i-1].z);

}

for(int i = 0; i < (int)segments.size();++i)

{

segments[i].x\_pol[3] = points[i].x;

segments[i].y\_pol[3] = points[i].y;

segments[i].z\_pol[3] = points[i].z;

}

}

void spline::post\_calc()

{

calc\_last\_pol(segments.back().x\_pol, points.back().x, points[points.size()-2].x);

calc\_last\_pol(segments.back().y\_pol, points.back().y, points[points.size()-2].y);

calc\_last\_pol(segments.back().z\_pol, points.back().z, points[points.size()-2].z);

}

void spline::calc\_last\_pol(QVector<GLfloat>& pol, float pl, float pr)

{

float t = k \* (pl - pr) / 2;

float m = t - pol[2];

float q = pl - pol[2] - pol[3];

pol[1] = 3 \* q - m;

pol[0] = m - 2 \* q;

}

void spline::draw()

{

glBegin(GL\_LINE\_STRIP);

glColor3f(0.0f, 1.0f, 0.0f);

GLfloat dx, dy, dz;

for(int i = 0; i < (int)segments.size(); ++i)

{

glVertex3f(points[i].x, points[i].y, points[i].z);

for(GLfloat t = 0.0; t < 1.0; t += 0.1)

{

dx = get\_f(segments[i].x\_pol[0], segments[i].x\_pol[1], segments[i].x\_pol[2], segments[i].x\_pol[3], t);

dy = get\_f(segments[i].y\_pol[0], segments[i].y\_pol[1], segments[i].y\_pol[2], segments[i].y\_pol[3], t);

dz = get\_f(segments[i].z\_pol[0], segments[i].z\_pol[1], segments[i].z\_pol[2], segments[i].z\_pol[3], t);

glColor3f(0.0f, 1.0f, 0.0f);

glVertex3f(dx, dy, dz);

}

glVertex3f(points[i + 1].x, points[i + 1].y, points[i + 1].z);

}

glEnd();

}

spline.h

#pragma once

#include <QOpenGLWidget>

#include <QtWidgets>

#include <QtOpenGL>

#include <QtMath>

struct segment

{

QVector<GLfloat> x\_pol;

QVector<GLfloat> y\_pol;

QVector<GLfloat> z\_pol;

};

struct my\_point

{

GLfloat x;

GLfloat y;

GLfloat z;

};

class spline

{

public:

spline();

spline(QVector<my\_point>& pnts);

GLfloat get\_f(GLfloat a, GLfloat b, GLfloat c, GLfloat d, GLfloat t);

void pre\_calc();

void calc();

void post\_calc();

void calc\_pol(QVector<GLfloat>& f\_p, QVector<GLfloat>& s\_p);

void calc\_last\_pol(QVector<GLfloat>& pol, float pl, float pr);

void draw();

QVector<segment> segments;

private:

QVector<my\_point> points;

int k = 1;

};

mainwindow.ui - форма с добавленным интерфейсом при помощи библиотеки Qt

1. Вывод

Выполнив данный курсовой проект я узнал, что такое линейчатые поверхности и сплайны, научился строить сглаженные кривые, проходящие через набор определенный точек, каркасное представление фигур, познакомился с библиотекой и IDE - Qt.

ЛИТЕРАТУРА

1. Документация OpenGL[Электронный ресурс]. URL: <https://www.khronos.org/registry/OpenGL-Refpages/es3/> (дата обращения: 12.12.2021).

2. Документация Qt[Электронный ресурс]. URL: <https://doc.qt.io> (дата обращения: 12.12.2021).

3. Сплайн[Электронный ресурс]. URL: [https://ru.wikipedia.org/wiki/Сплайн](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD) (дата обращения: 12.12.2021).

4. Линейчатая поверхность[Электронный ресурс]. URL: [https://ru.wikipedia.org/wiki/Линейчатая\_поверхность#:~:text=Линейчатая%20поверхность%20―%20поверхность%2C%20образованная%20движением,все%20прямолинейные%20образующие%2C%20направляющей%20кривой.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D1%87%D0%B0%D1%82%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B2%D0%B5%D1%80%D1%85%D0%BD%D0%BE%D1%81%D1%82%D1%8C#:~:text=%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D1%87%D0%B0%D1%82%D0%B0%D1%8F%20%D0%BF%D0%BE%D0%B2%D0%B5%D1%80%D1%85%D0%BD%D0%BE%D1%81%D1%82%D1%8C%20%E2%80%95%20%D0%BF%D0%BE%D0%B2%D0%B5%D1%80%D1%85%D0%BD%D0%BE%D1%81%D1%82%D1%8C%2C%20%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F%20%D0%B4%D0%B2%D0%B8%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5%D0%BC,%D0%B2%D1%81%D0%B5%20%D0%BF%D1%80%D1%8F%D0%BC%D0%BE%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D0%B5%20%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D1%83%D1%8E%D1%89%D0%B8%D0%B5%2C%20%D0%BD%D0%B0%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%8F%D1%8E%D1%89%D0%B5%D0%B9%20%D0%BA%D1%80%D0%B8%D0%B2%D0%BE%D0%B9.) (дата обращения: 12.12.2021).