Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

**Курсовая работа**

**по курсу «Компьютерная графика»**

|  |  |
| --- | --- |
| Студент: | Марков А.Н. |
| Группа: | М80-308Б-18 |
| Преподаватель: | Филиппов Г.С. |
| Оценка: |  |
| Дата: |  |

Москва

2020

**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 №5: Линейчатая поверхность (направляющие – кривые Безье 3D 2-й степени).

**2. Решение задачи.**

Одним из способов построения поверхности является способ, при котором поверхность представляется как совокупность положений некоторой линии, перемещающейся в пространстве по определенному правилу.

Линия, которая перемещается в пространстве по определенному правилу, называется образующей.

Линии, вдоль которых двигается образующая, называются направляющими. В моем варианте направляющими линиями являются кривые Безье 2-й степени.

Кривая Безье 2-й степени строится в соответствии со следующей формулой:

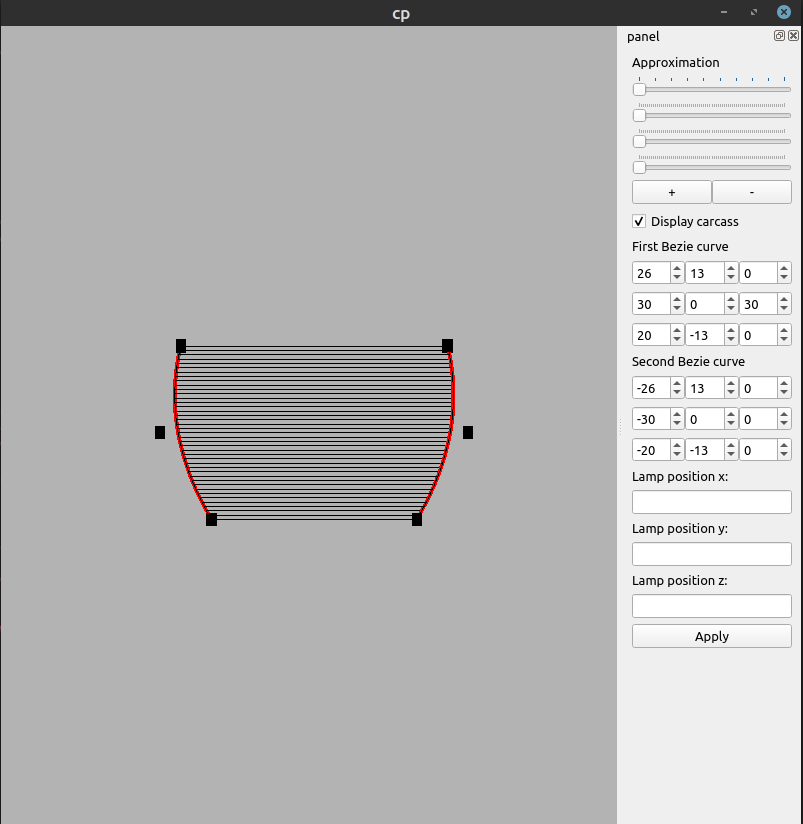


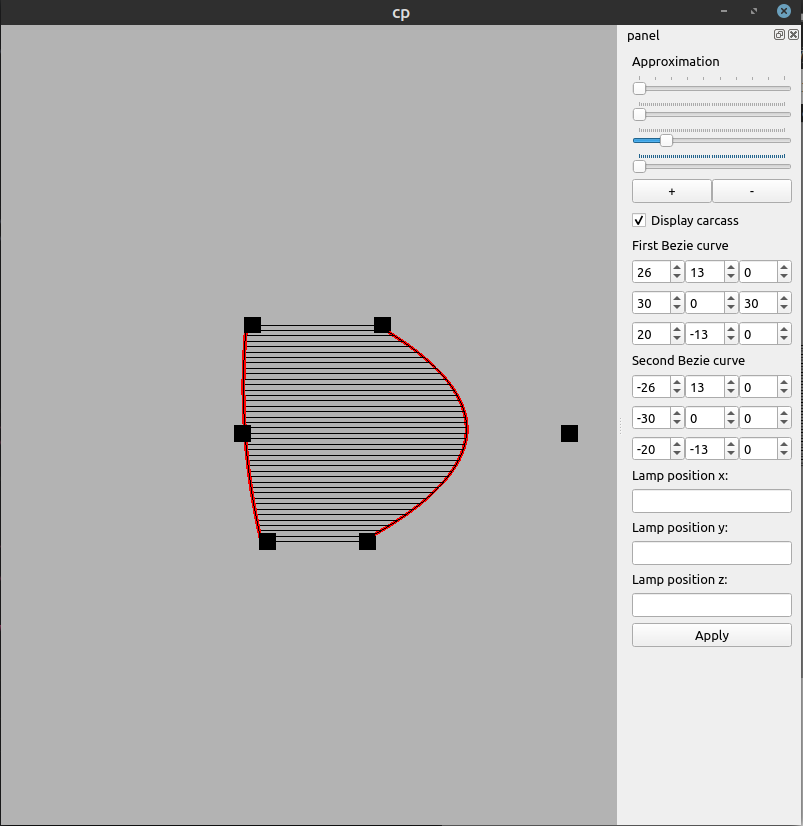
,

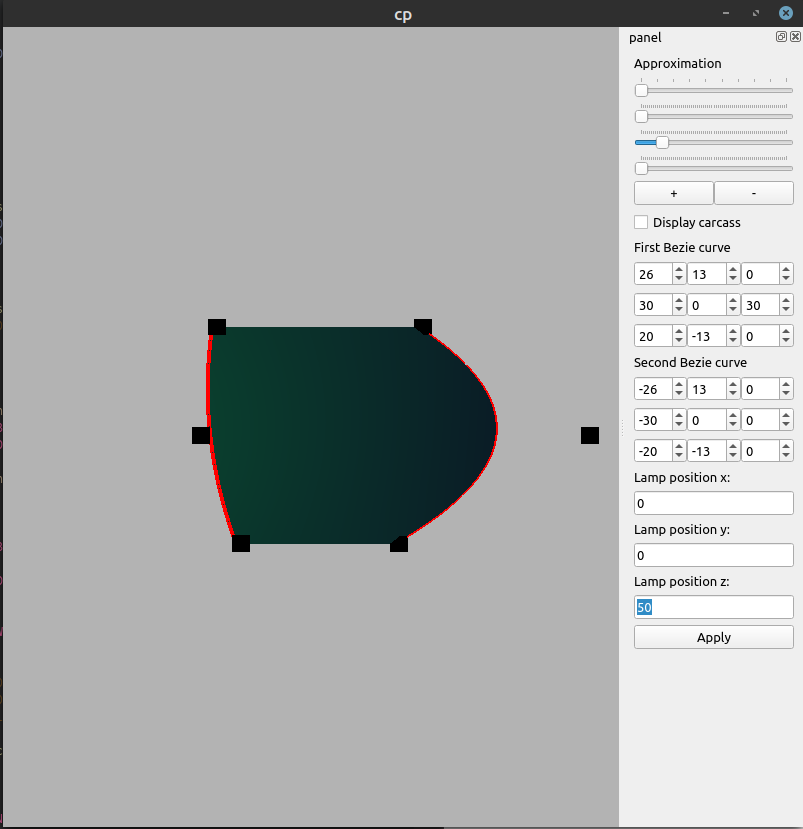
где P0, P1, P2 — опорные точки.

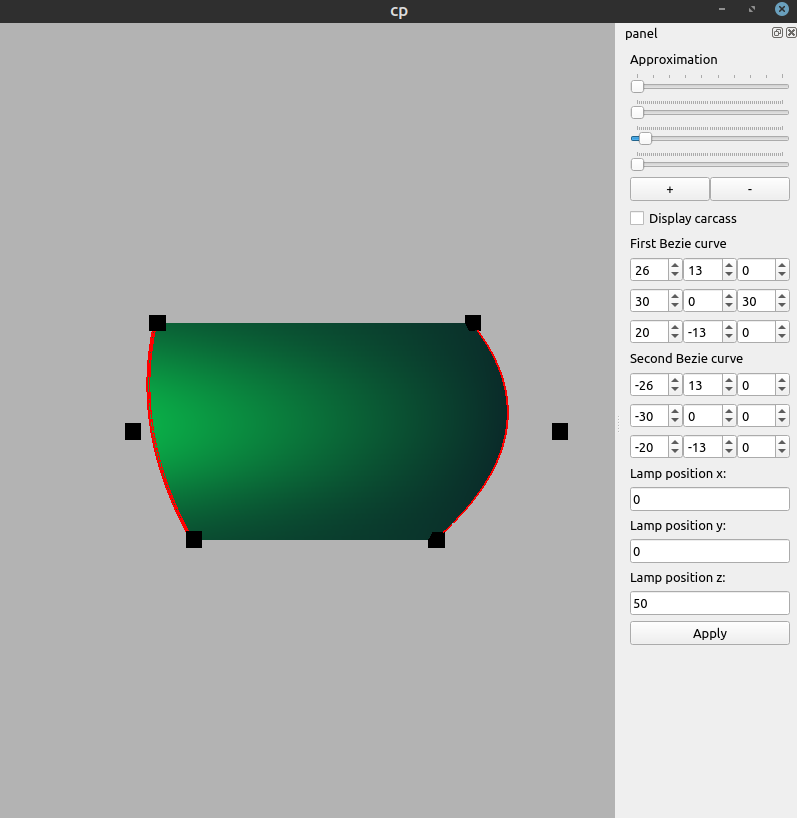
Для решения задачи я использовал C++, фреймворк Qt и OpenGL.

**3. Демонстрация работы программы.**









**4. Листинг программы.**

**view.h**

#ifndef VIEW\_H

#define VIEW\_H

#include <QGLWidget>

#include <QOpenGLFunctions>

#include <vector>

#include <QVector4D>

#include <QWidget>

class View : public QGLWidget, public QOpenGLFunctions

{

float step = 0.025f;

float scale = 1.;

float angleX;

float angleY;

float angleZ;

unsigned int cntPoints1;

unsigned int cntPoints2;

bool display\_carcass = true;

float lightPositionX;

float lightPositionY;

float lightPositionZ;

std::vector<QVector4D> points1;

std::vector<QVector4D> points2;

void drawBezierCurve(const std::vector<QVector4D> &points);

public:

View();

void initializeGL() override;

void resizeGL(int width, int height) override;

void paintGL() override;

void change\_scale(float s);

void change\_display\_carcass();

float get\_scale();

void set\_angle\_x(float angle);

void set\_angle\_y(float angle);

void set\_angle\_z(float angle);

void set\_step(float s);

void change\_points1(float val, unsigned int numPoint, unsigned int cord);

void change\_points2(float val, unsigned int numPoint, unsigned int cord);

void set\_light\_position\_x(float x\_pos);

void set\_light\_position\_y(float y\_pos);

void set\_light\_position\_z(float z\_pos);

float get\_light\_position\_x();

float get\_light\_position\_y();

float get\_light\_position\_z();

private:

float calcX(double t, double v);

float calcY(double t, double v);

float calcZ(double t, double v);

};

#endif // VIEW\_H

**view.cpp**

#include "view.h"

#include <QMouseEvent>

#include <cmath>

#include <QMatrix4x4>

const unsigned int MAX\_CNT\_POINTS = 3;

const int SQUARE\_SIZE = 2;

View::View() : QGLWidget() {

angleX = 0;

angleY = 0;

angleZ = 0;

cntPoints1 = 0;

cntPoints2 = 0;

lightPositionX = lightPositionY = lightPositionZ = 0.f;

points1.resize(MAX\_CNT\_POINTS);

points2.resize(MAX\_CNT\_POINTS);

}

void View::initializeGL() {

initializeOpenGLFunctions();

glClearColor(0.702f, 0.702f, 0.702f, 1.f);

glEnable(GL\_DEPTH\_TEST);

}

void View::resizeGL(int width, int height) {

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glViewport(0, 0, width, height);

}

void View::paintGL() {

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

glEnable(GL\_DEPTH\_TEST);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-60., 60., -60., 60., -100. \* (width() + height()), 100. \* (width() + height()));

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glRotatef(angleX, 1, 0, 0);

glRotatef(angleY, 0, 1, 0);

glRotatef(angleZ, 0, 0, 1);

glScalef(scale, scale, scale);

glLineWidth(1.f);

if (display\_carcass) {

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glColor3f(0.f, 0.f, 0.f);

glDisable(GL\_LIGHTING);

} else {

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

glEnable(GL\_LIGHTING);

}

glPushMatrix();

glLoadIdentity();

glEnable(GL\_NORMALIZE);

float ambient\_color[] = {0.2f, 0.2f, 0.2f, 1.f};

float diffuse\_color[] = {0.3f, 0.3f, 0.3f, 1.f};

float specular\_color[] = {1.f, 1.f, 1.f, 1.f};

unsigned int shininess = 90;

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, ambient\_color);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, diffuse\_color);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, specular\_color);

glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, shininess);

float light\_ambient[] = {0.f, 0.22f, 0.51f, 1.f};

float light\_diffuse[] = {0.f, 0.55f, 0.128f, 1.f};

float light\_specular[] = {0.f, 0.44f, 0.102f, 1.f};

float light\_position[] = {lightPositionX,

lightPositionY,

lightPositionZ, 1.f};

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

glLightf(GL\_LIGHT0, GL\_SPOT\_EXPONENT, 128);

glLightf(GL\_LIGHT0, GL\_CONSTANT\_ATTENUATION, 1.f);

glPopMatrix();

QVector4D firstPoint{calcX(0, 0), calcY(0, 0), calcZ(0, 0), 1};

QVector4D fourthPoint{calcX(0, 1), calcY(0, 1), calcZ(0, 1), 1};

for (double t = static\_cast<double>(step); t < 1.; t += static\_cast<double>(step)) {

QVector4D secondPoint{calcX(t, 0), calcY(t, 0), calcZ(t, 0), 1};

QVector4D thirdPoint{calcX(t, 1), calcY(t, 1), calcZ(t, 1), 1};

glBegin(GL\_POLYGON);

glVertex3f(firstPoint.x(), firstPoint.y(), firstPoint.z());

glVertex3f(secondPoint.x(), secondPoint.y(), secondPoint.z());

glVertex3f(thirdPoint.x(), thirdPoint.y(), thirdPoint.z());

glVertex3f(fourthPoint.x(), fourthPoint.y(), fourthPoint.z());

glEnd();

firstPoint = secondPoint;

fourthPoint = thirdPoint;

if (t + static\_cast<double>(step) >= 1.) {

secondPoint = QVector4D{calcX(1, 0), calcY(1, 0), calcZ(1., 0), 1};

thirdPoint = QVector4D{calcX(1, 1), calcY(1, 1), calcZ(1, 1), 1};

glBegin(GL\_POLYGON);

glVertex3f(firstPoint.x(), firstPoint.y(), firstPoint.z());

glVertex3f(secondPoint.x(), secondPoint.y(), secondPoint.z());

glVertex3f(thirdPoint.x(), thirdPoint.y(), thirdPoint.z());

glVertex3f(fourthPoint.x(), fourthPoint.y(), fourthPoint.z());

glEnd();

}

}

glDisable(GL\_LIGHT0);

glDisable(GL\_LIGHTING);

drawBezierCurve(points1);

drawBezierCurve(points2);

// draw points of Bezier curves

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

glColor3f(0, 0, 0);

for (unsigned int i = 0; i < cntPoints1; i++) {

glBegin(GL\_POLYGON);

glVertex3f(points1[i].x(), points1[i].y(), points1[i].z());

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y(), points1[i].z());

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y() - SQUARE\_SIZE, points1[i].z());

glVertex3f(points1[i].x(), points1[i].y() - SQUARE\_SIZE, points1[i].z());

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(points1[i].x(), points1[i].y(), points1[i].z());

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y(), points1[i].z());

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y(), points1[i].z() + SQUARE\_SIZE);

glVertex3f(points1[i].x(), points1[i].y(), points1[i].z() + SQUARE\_SIZE);

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(points1[i].x(), points1[i].y(), points1[i].z());

glVertex3f(points1[i].x(), points1[i].y() - SQUARE\_SIZE, points1[i].z());

glVertex3f(points1[i].x(), points1[i].y() - SQUARE\_SIZE, points1[i].z() + SQUARE\_SIZE);

glVertex3f(points1[i].x(), points1[i].y(), points1[i].z() + SQUARE\_SIZE);

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y(), points1[i].z());

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y() - SQUARE\_SIZE, points1[i].z());

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y() - SQUARE\_SIZE, points1[i].z() + SQUARE\_SIZE);

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y(), points1[i].z() + SQUARE\_SIZE);

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(points1[i].x(), points1[i].y(), points1[i].z() + SQUARE\_SIZE);

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y(), points1[i].z() + SQUARE\_SIZE);

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y() - SQUARE\_SIZE, points1[i].z() + SQUARE\_SIZE);

glVertex3f(points1[i].x(), points1[i].y() - SQUARE\_SIZE, points1[i].z() + SQUARE\_SIZE);

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(points1[i].x(), points1[i].y() - SQUARE\_SIZE, points1[i].z());

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y() - SQUARE\_SIZE, points1[i].z());

glVertex3f(points1[i].x() + SQUARE\_SIZE, points1[i].y() - SQUARE\_SIZE, points1[i].z() + SQUARE\_SIZE);

glVertex3f(points1[i].x(), points1[i].y() - SQUARE\_SIZE, points1[i].z() + SQUARE\_SIZE);

glEnd();

}

for (unsigned int i = 0; i < cntPoints2; i++) {

glBegin(GL\_POLYGON);

glVertex3f(points2[i].x(), points2[i].y(), points2[i].z());

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y(), points2[i].z());

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y() - SQUARE\_SIZE, points2[i].z());

glVertex3f(points2[i].x(), points2[i].y() - SQUARE\_SIZE, points2[i].z());

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(points2[i].x(), points2[i].y(), points2[i].z());

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y(), points2[i].z());

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y(), points2[i].z() + SQUARE\_SIZE);

glVertex3f(points2[i].x(), points2[i].y(), points2[i].z() + SQUARE\_SIZE);

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(points2[i].x(), points2[i].y(), points2[i].z());

glVertex3f(points2[i].x(), points2[i].y() - SQUARE\_SIZE, points2[i].z());

glVertex3f(points2[i].x(), points2[i].y() - SQUARE\_SIZE, points2[i].z() + SQUARE\_SIZE);

glVertex3f(points2[i].x(), points2[i].y(), points2[i].z() + SQUARE\_SIZE);

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y(), points2[i].z());

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y() - SQUARE\_SIZE, points2[i].z());

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y() - SQUARE\_SIZE, points2[i].z() + SQUARE\_SIZE);

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y(), points2[i].z() + SQUARE\_SIZE);

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(points2[i].x(), points2[i].y(), points2[i].z() + SQUARE\_SIZE);

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y(), points2[i].z() + SQUARE\_SIZE);

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y() - SQUARE\_SIZE, points2[i].z() + SQUARE\_SIZE);

glVertex3f(points2[i].x(), points2[i].y() - SQUARE\_SIZE, points2[i].z() + SQUARE\_SIZE);

glEnd();

glBegin(GL\_POLYGON);

glVertex3f(points2[i].x(), points2[i].y() - SQUARE\_SIZE, points2[i].z());

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y() - SQUARE\_SIZE, points2[i].z());

glVertex3f(points2[i].x() + SQUARE\_SIZE, points2[i].y() - SQUARE\_SIZE, points2[i].z() + SQUARE\_SIZE);

glVertex3f(points2[i].x(), points2[i].y() - SQUARE\_SIZE, points2[i].z() + SQUARE\_SIZE);

glEnd();

}

glDisable(GL\_DEPTH\_TEST);

}

void View::change\_scale(float s) {

scale += s;

}

void View::change\_display\_carcass() {

if (display\_carcass) {

display\_carcass = false;

} else {

display\_carcass = true;

}

}

float View::get\_scale() {

return scale;

}

void View::set\_angle\_x(float angle) {

angleX = angle;

}

void View::set\_angle\_y(float angle) {

angleY = angle;

}

void View::set\_angle\_z(float angle) {

angleZ = angle;

}

void View::set\_step(float s) {

step = s;

}

void View::change\_points1(float val, unsigned int numPoint, unsigned int cord) {

if (cord == 0) {

points1[numPoint].setX(val);

} else if (cord == 1) {

points1[numPoint].setY(val);

} else if (cord == 2) {

points1[numPoint].setZ(val);

}

if (cntPoints1 < 3) {

cntPoints1++;

}

}

void View::change\_points2(float val, unsigned int numPoint, unsigned int cord) {

if (cord == 0) {

points2[numPoint].setX(val);

} else if (cord == 1) {

points2[numPoint].setY(val);

} else if (cord == 2) {

points2[numPoint].setZ(val);

}

if (cntPoints2 < 3) {

cntPoints2++;

}

}

float View::calcX(double t, double v) {

double x = (1 - v) \* (std::pow(1 - t, 2) \* static\_cast<double>(points1[0].x() + SQUARE\_SIZE / 2) +

2 \* t \* (1 - t) \* static\_cast<double>(points1[1].x() + SQUARE\_SIZE / 2) +

std::pow(t, 2) \* static\_cast<double>(points1[2].x() + SQUARE\_SIZE / 2)) +

v \* (std::pow(1 - t, 2) \* static\_cast<double>(points2[0].x() + SQUARE\_SIZE / 2) +

2 \* t \* (1 - t) \* static\_cast<double>(points2[1].x() + SQUARE\_SIZE / 2) +

std::pow(t, 2) \* static\_cast<double>(points2[2].x() + SQUARE\_SIZE / 2));

return static\_cast<float>(x);

}

float View::calcY(double t, double v) {

double y = (1 - v) \* (std::pow(1 - t, 2) \* static\_cast<double>(points1[0].y() - SQUARE\_SIZE / 2) +

2 \* t \* (1 - t) \* static\_cast<double>(points1[1].y() - SQUARE\_SIZE / 2) +

std::pow(t, 2) \* static\_cast<double>(points1[2].y() - SQUARE\_SIZE / 2)) +

v \* (std::pow(1 - t, 2) \* static\_cast<double>(points2[0].y() - SQUARE\_SIZE / 2) +

2 \* t \* (1 - t) \* static\_cast<double>(points2[1].y() - SQUARE\_SIZE / 2) +

std::pow(t, 2) \* static\_cast<double>(points2[2].y() - SQUARE\_SIZE / 2));

return static\_cast<float>(y);

}

float View::calcZ(double t, double v) {

double z = (1 - v) \* (std::pow(1 - t, 2) \* static\_cast<double>(points1[0].z()) +

2 \* t \* (1 - t) \* static\_cast<double>(points1[1].z()) +

std::pow(t, 2) \* static\_cast<double>(points1[2].z())) +

v \* (std::pow(1 - t, 2) \* static\_cast<double>(points2[0].z()) +

2 \* t \* (1 - t) \* static\_cast<double>(points2[1].z()) +

std::pow(t, 2) \* static\_cast<double>(points2[2].z()));

return static\_cast<float>(z);

}

void View::set\_light\_position\_x(float x\_pos) {

lightPositionX = x\_pos;

}

void View::set\_light\_position\_y(float y\_pos) {

lightPositionY = y\_pos;

}

void View::set\_light\_position\_z(float z\_pos) {

lightPositionZ = z\_pos;

}

float View::get\_light\_position\_x() {

return lightPositionX;

}

float View::get\_light\_position\_y() {

return lightPositionY;

}

float View::get\_light\_position\_z() {

return lightPositionZ;

}

void View::drawBezierCurve(const std::vector<QVector4D> &points) {

glColor3f(1.f, 0.f, 0.f);

float prevX = points[0].x() + SQUARE\_SIZE / 2, prevY = points[0].y() - SQUARE\_SIZE / 2;

float prevZ = points[0].z();

glLineWidth(4.f);

glBegin(GL\_LINE\_STRIP);

glVertex3f(prevX, prevY, prevZ);

for (double t = static\_cast<double>(step); t < 1.; t += static\_cast<double>(step)) {

float x = static\_cast<float>(std::pow((1. - t), 2.)) \* (points[0].x() + SQUARE\_SIZE / 2) +

2.f \* static\_cast<float>(t) \* static\_cast<float>(1. - t) \* (points[1].x() + SQUARE\_SIZE / 2) +

static\_cast<float>(std::pow(t, 2.)) \* (points[2].x() + SQUARE\_SIZE / 2);

float y = static\_cast<float>(std::pow((1. - t), 2.)) \* (points[0].y() - SQUARE\_SIZE / 2) +

2.f \* static\_cast<float>(t) \* static\_cast<float>(1. - t) \* (points[1].y() - SQUARE\_SIZE / 2) +

static\_cast<float>(std::pow(t, 2.)) \* (points[2].y() - SQUARE\_SIZE / 2);

float z = static\_cast<float>(std::pow((1. - t), 2.)) \* (points[0].z()) +

2.f \* static\_cast<float>(t) \* static\_cast<float>(1. - t) \* (points[1].z()) +

static\_cast<float>(std::pow(t, 2.)) \* (points[2].z());

glVertex3f(x, y, z);

prevX = x;

prevY = y;

prevZ = z;

if (t + static\_cast<double>(step) >= 1.) {

x = points[2].x() + SQUARE\_SIZE / 2;

y = points[2].y() - SQUARE\_SIZE / 2;

z = points[2].z();

glVertex3f(x, y, z);

}

}

glEnd();

}

**5. Выводы**

В ходе выполнения данной курсовой работы была реализована программа, позволяющая моделировать линейчатую поверхность, направляющими которой являются кривые Безье 2-й степени. Выполнить данную лабораторную было довольно интересно и в меру сложно.