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.

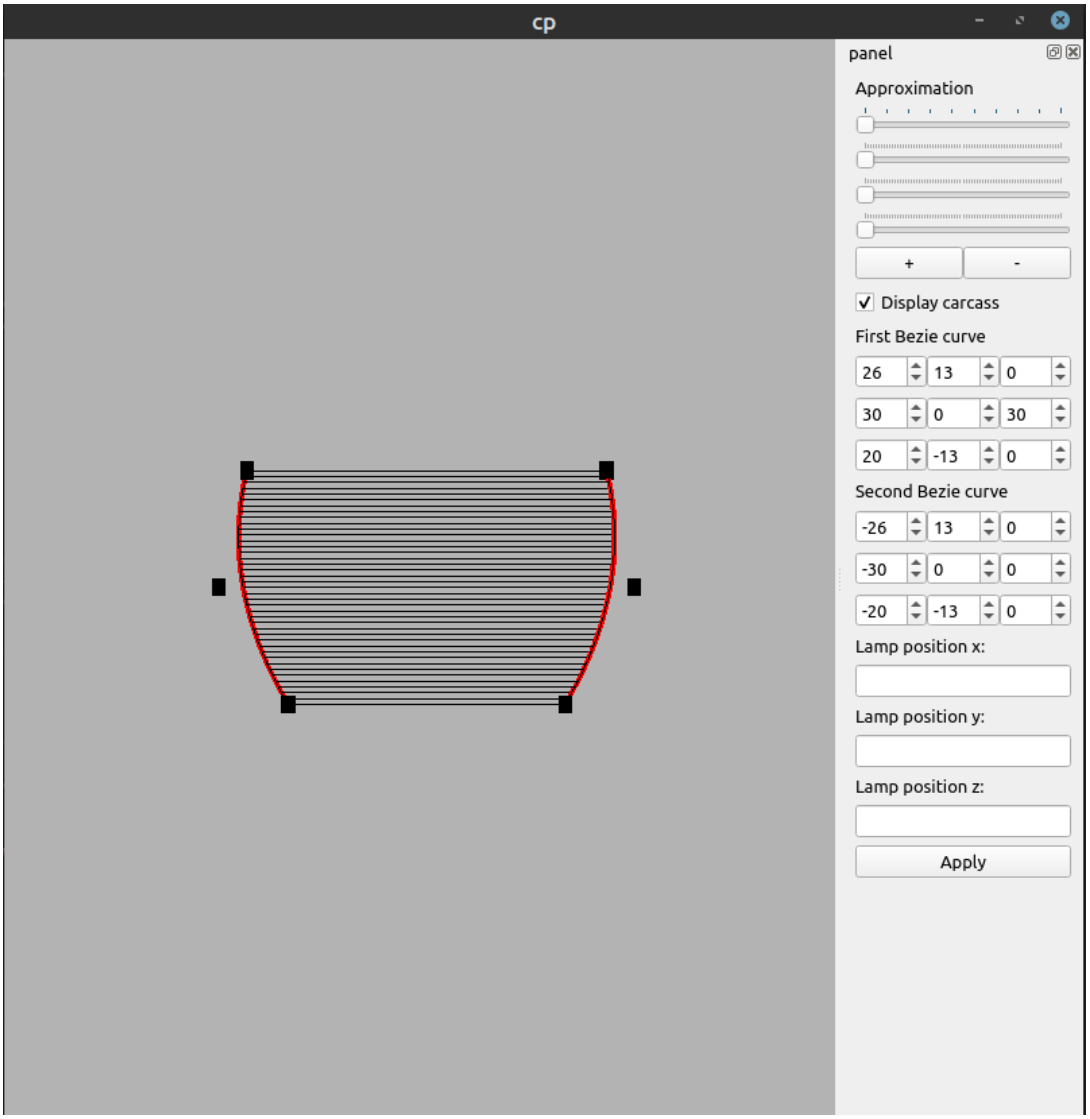
Кривая Безье 2-й степени строится в соответствии со следующей формулой:

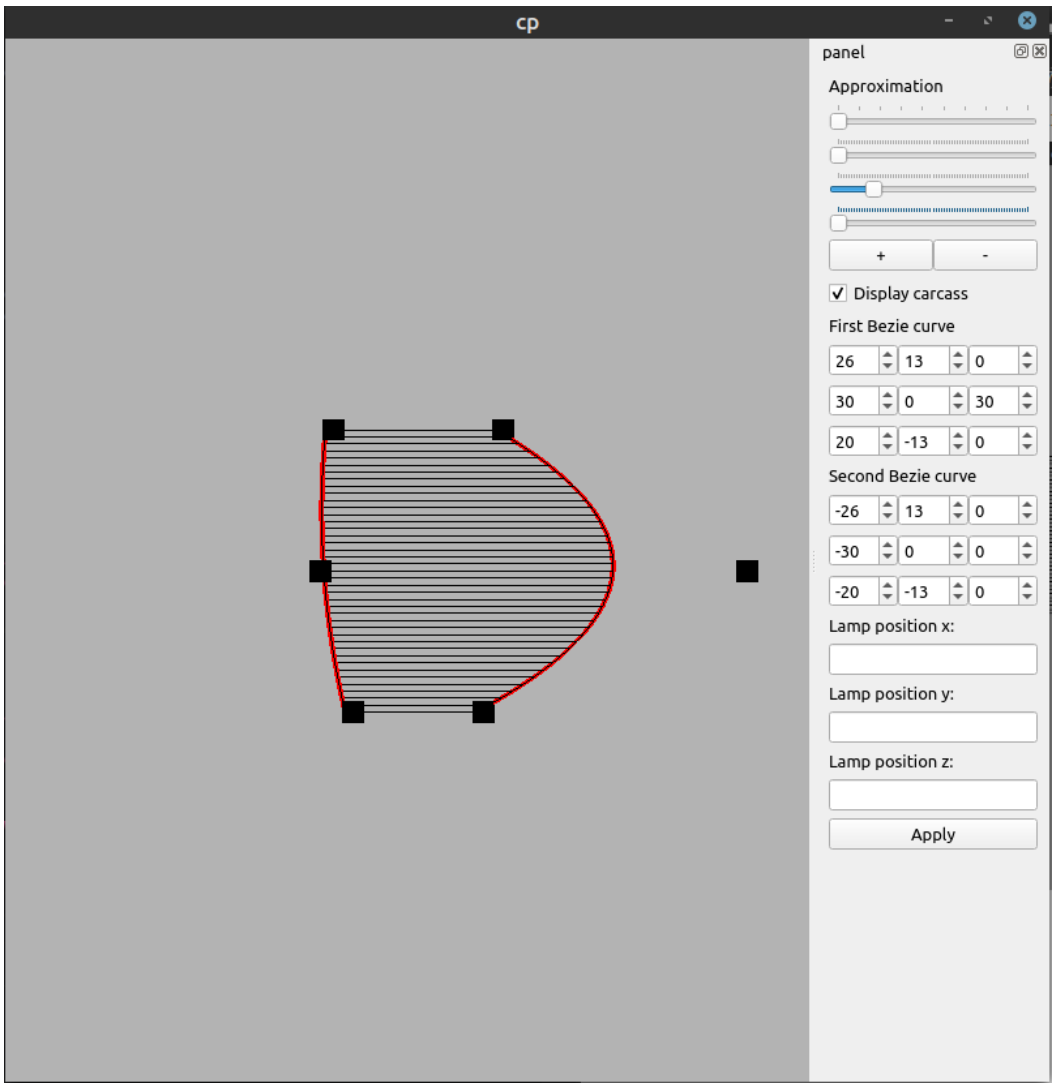
$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2, \quad t \in [0,1],$$

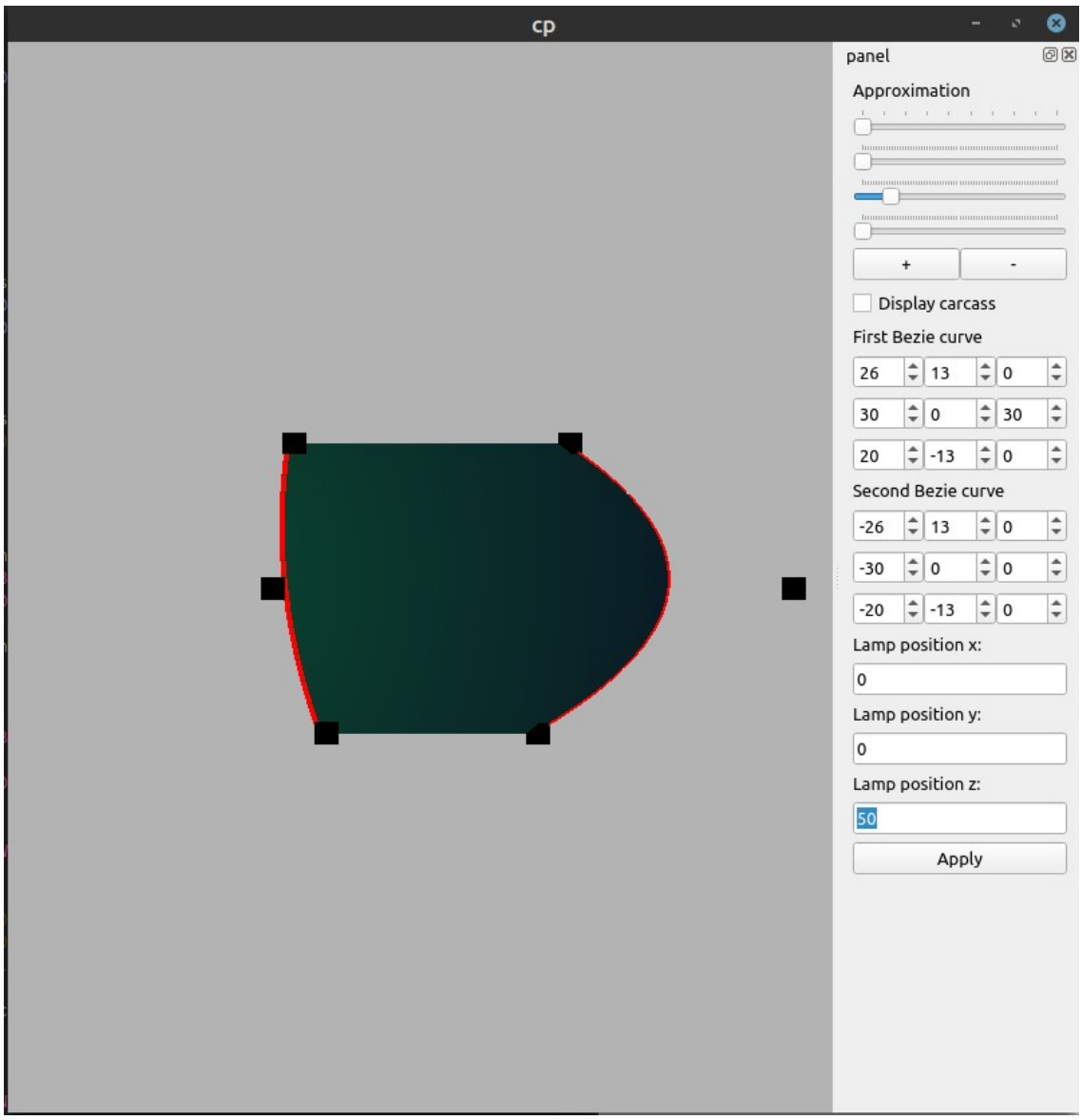
где P0, P1, P2 — опорные точки.

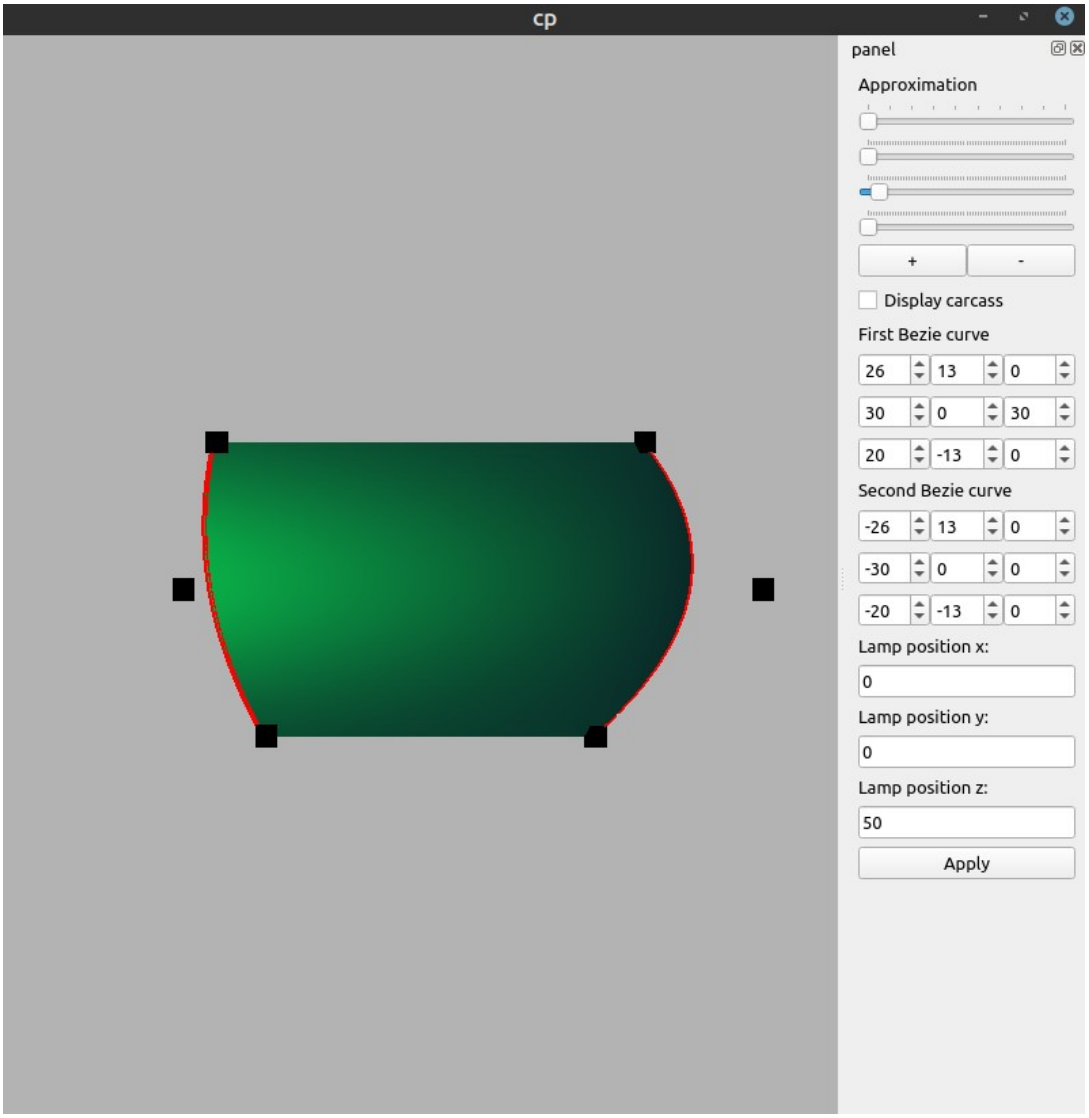
Для решения задачи я использовал C++, фреймворк Qt и OpenGL.

3. Демонстрация работы программы.




cp
panel
Approximation
+
-
Display carcass
First Bezie curve
26
13
0
30
0
30
20
-13
0
Second Bezie curve
-26
13
0
-30
0
0
-20
-13
0
Lamp position x:
Lamp position y:
Lamp position z:
Apply


cp
panel
Approximation
+
-
Display carcass
First Bezie curve
26
13
0
30
0
30
20
-13
0
Second Bezie curve
-26
13
0
-30
0
0
-20
-13
0
Lamp position x:
0
Lamp position y:
0
Lamp position z:
50
Apply



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.