МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Компьютерная графика»

ТЕМА: «КУБИЧЕСКИЕ СПЛАЙН»

Студентки гр. 1384 Усачева Д.В.
Пчелинцева К.Р.

Преподаватель Герасимова Т.В.

Санкт-Петербург
2024

Задание.

Вариант 6: Кривая Безье 3-й степени, состоящая из 2-х сегментов.

Общие теоретические сведения.

Сплайны - это гладкие (имеющие несколько непрерывных производных) кусочно-полиномиальные функции, которые могут быть использованы для представления функций, заданных большим количеством значений и для которых неприменима аппроксимация одним полиномом. Так как сплайны гладки, экономичны и легки в работе, они используются при построении произвольных функций для:

- моделирования кривых;
- аппроксимации данных с помощью кривых;
- выполнения функциональных аппроксимаций;
- решения функциональных уравнений.

Здесь кратко излагаются некоторые основные положения и использования сплайнов в 3D графики.

Важным их свойством является простота вычислений. На практике часто используют сплайны вида полиномов третьей степени. С их помощью довольно удобно проводить кривые, которые интуитивно соответствуют человеческому субъективному понятию гладкости.

Определим искомую функцию $y = S(x)$, причем поставим два условия:

- Функция должна проходить через все точки: $S(x_i) = y_i$, $i = \overline{0,m}$;
- Функция должна быть дважды непрерывно дифференцируема, то есть иметь непрерывную вторую производную на всем отрезке $[x_0, x_m]$.

На каждом из отрезков $[x_i, x_{i+1}]$, $i = \overline{0,m-1}$, ищется функция в виде полинома третьей степени:

$$S_i(x) = \sum_{j=0}^{3} a_{ij}(x - x_i)^j .$$

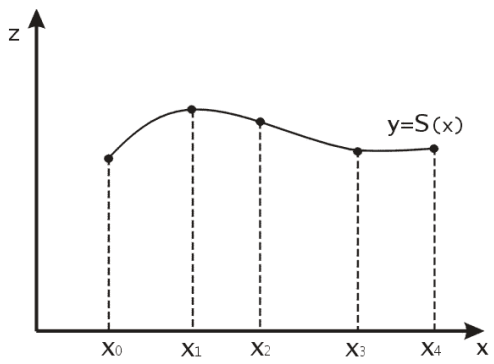


Рис. Сплайновая функция

Поскольку для каждого из отрезков $[x_i, x_{i+1}]$ необходимо найти S Задача построения полинома сводится к нахождению коэффициентов a_{ij}, при этом общее количество искомых коэффициентов будет $4m$.

Перейдем к более сложному случаю – заданию кривых в трехмерном пространстве. Для функционального задания кривой $\begin{cases} y = f(x) \\ z = f(x) \end{cases}$ возможны многозначности в случае самопересечений и неудобства при значениях производных равных ∞.

Ввиду этого ищется функция в параметрическом виде. Пусть t - независимый параметр, такой что $0 \le t \le 1$. Кубическим параметрическим сплайном назовем следующую систему уравнений:

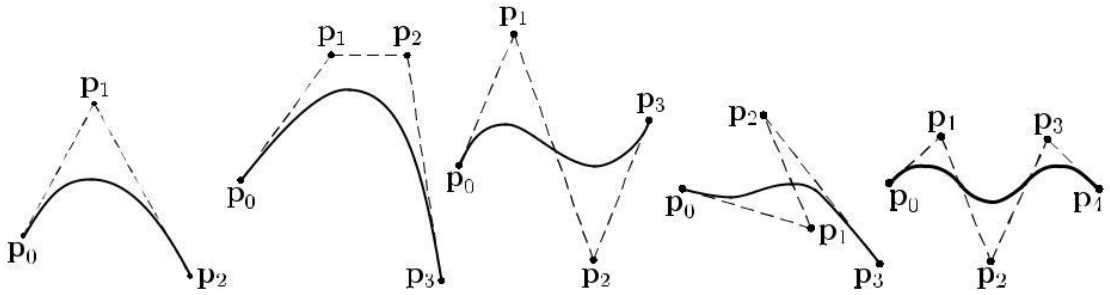
$$\begin{cases} x(t) = a_x t^3 + b_x t^2 + c_x t + d_x; \\ y(t) = a_y t^3 + b_y t^2 + c_y t + d_y; \\ z(t) = a_z t^3 + b_z t^2 + c_z t + d_z. \end{cases}$$

Координаты точек на кривой описываются вектором $(x(t), y(t), z(t))$, а три производные задают координаты соответствующего касательного вектора в точке. Например, для координаты x:

$$\frac{dx}{dt} = 3a_x t^2 + 2b_x t + c_x.$$

Рассмотрим форму Безье, которая отличается от формы Эрмита способом задания граничных условий, а именно вместо векторов R_1 и R_4 вводятся точки (и соответствующие им радиус- векторы) P_2 и P_3, как показано на рисунке 3, такие, что выполняются условия: $P'(0) = R_1 = 3(P_2 - P_1)$ и $P'(1) = R_4 = 3(P_4 - P_3)$.

Рис. Параметрический сплайн в форме Безье



Переход от формы Эрмита к форме Безье осуществляется преобразованием:

$$G_h = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = M_{hb} G_b \qquad (*)$$

где G_b - геометрический вектор Безье. Подставляя это в выражение для $x(t)$, получаем

$$x(t) = TM_h G_{hx} = TM_h M_{hb} G_{bx} = (1 - t^3) P_1 + 3t(t-1)^2 P_2 + 3t^2 (1-t) P_3 + t^3 P_4$$

Полезным свойством сплайнов в форме Безье является то, что кривая всегда лежит внутри выпуклой оболочки, образованной четырехугольником $(P_1 P_2 P_3 P_4)$. Это свойство можно доказать, пользуясь тем, что в выражении (*) коэффициенты принимают значения от 0 до 1 и их сумма равна единице.

Матрица вида

$$M_h M_{hb} = M_b = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$ - называется матрицей Безье.

В большинстве случаев кривая Безье — это полином, степень которого на единицу меньше заданного числа контрольных точек: три точки определяют параболу, четыре кубическую кривую и т.д. При определённом положении контрольных точек, однако, получаются вырожденные полиномы Безье. Например, кривая Безье, сгенерированная тремя контрольными точками, лежащими на одной прямой, является прямым отрезком. Наконец, ¾кривая¿ Безье для набора контрольных точек с совпадающими координатами представляет собой одну точку

Кривую Безье можно подобрать по любому числу контрольных точек, но это требует расчета полиномиальных функций большой степени. Если необходимо сгенерировать сложные кривые, их проще сформировать стыковкой нескольких участков Безье меньшей степени.

Выполнение работы.

Данная работа выполнена на операционной системе Windows 11. Приложение было создано на Python 3.11 с применением библиотеки OpenGL.

Для создания интерфейса использовалась библиотека PyQt6. Интерфейс был разработан интерактивно в программе QtDesigner.

Все расчеты, связанные с кривой Безье, были реализованы в файле bezier_curve.py. Для расчета кривой используется формула:

$$x(t) = (1 - t^3)P_1 + 3t(t-1)^2 P_2 + 3t^2(1-t)P_3 + t^3 P_4$$

Она используется в следующей функции представленной в листинге 1.

Листинг 1 – Расчет кривой Безье

```
def bezier_curve(p0, p1, p2, p3, p4, p5, p6):  # кривая Безье
    curve_points = np.zeros((200, 2))
    for i, t in enumerate(np.linspace(0, 1, 100)):
        x = (1 - t) ** 3 * p0[0] + 3 * (1 - t) ** 2 * t * p1[0] +
3 * (1 - t) * t ** 2 * p2[0] + t ** 3 * p3[0]
        y = (1 - t) ** 3 * p0[1] + 3 * (1 - t) ** 2 * t * p1[1] +
3 * (1 - t) * t ** 2 * p2[1] + t ** 3 * p3[1]
        curve_points[i] = [x, y]
    for i, t in enumerate(np.linspace(0, 1, 100)):
        z = (1 - t) ** 3 * p3[0] + 3 * (1 - t) ** 2 * t * p4[0] +
3 * (1 - t) * t ** 2 * p5[0] + t ** 3 * p6[0]
        w = (1 - t) ** 3 * p3[1] + 3 * (1 - t) ** 2 * t * p4[1] +
3 * (1 - t) * t ** 2 * p5[1] + t ** 3 * p6[1]
        curve_points[i + 100] = [z, w]
    return curve_points
```

В файле controller.py создан виджет OpenGL с обработчиком нажатий и удерживания левой кнопки мыши, который используется для управления контрольными точками. При помощи удерживания ЛКМ можно перемещать эти точки по экрану.

В модуле design.py реализовано главное окно приложения с заголовком "OpenGL Window", а также добавляется OpenGL виджет в качестве центрального элемента окна.

Тестирование.

При запуске программы можно увидеть 7 контрольных точек и кривую. Результат представлен на рисунке 1.

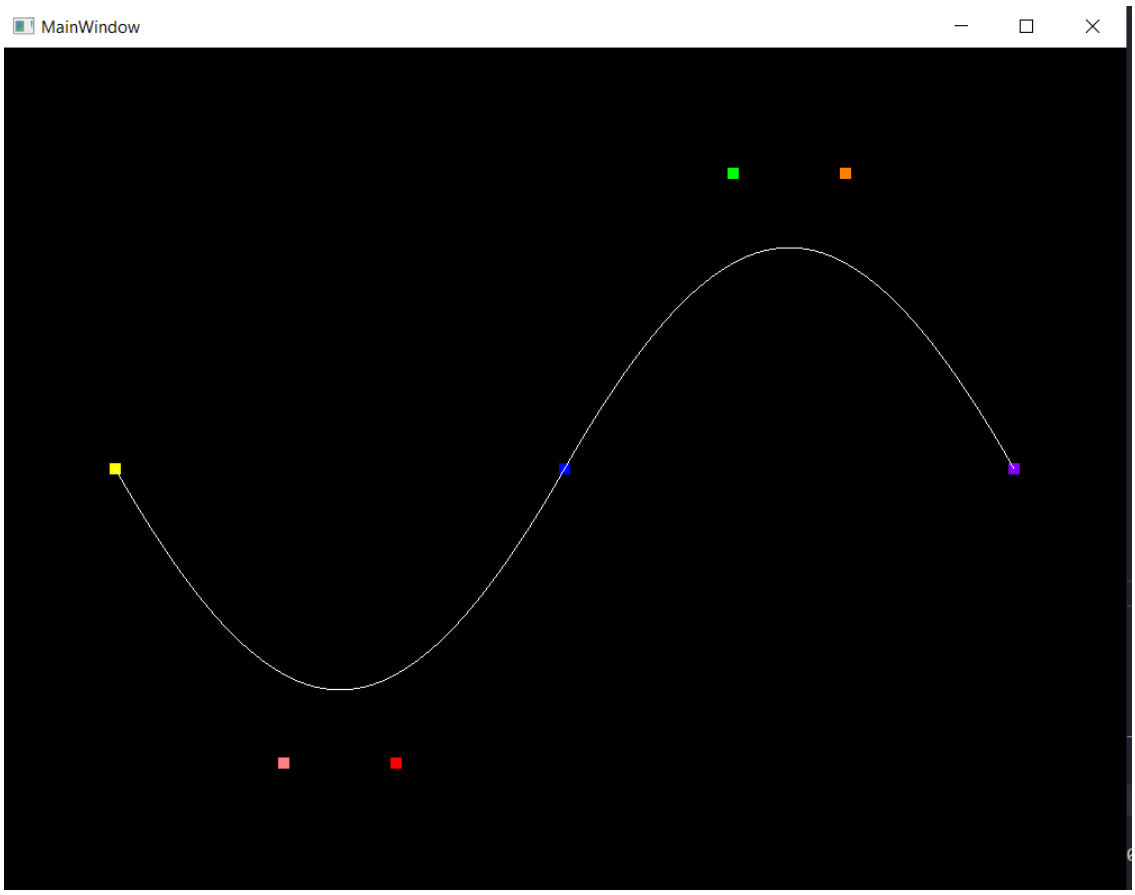


Рисунок 1 — Демонстрация работы программы

Можно перетащить контрольные точки с помощью левой кнопки мыши. Заметим, что в обоих случаях кривая (каждый из ее сегментов) лежит внутри выпуклой оболочки, образованной четырехугольником $(P_1P_2P_3P_4)$. Результат перемещения представлен ниже на рисунке 2.

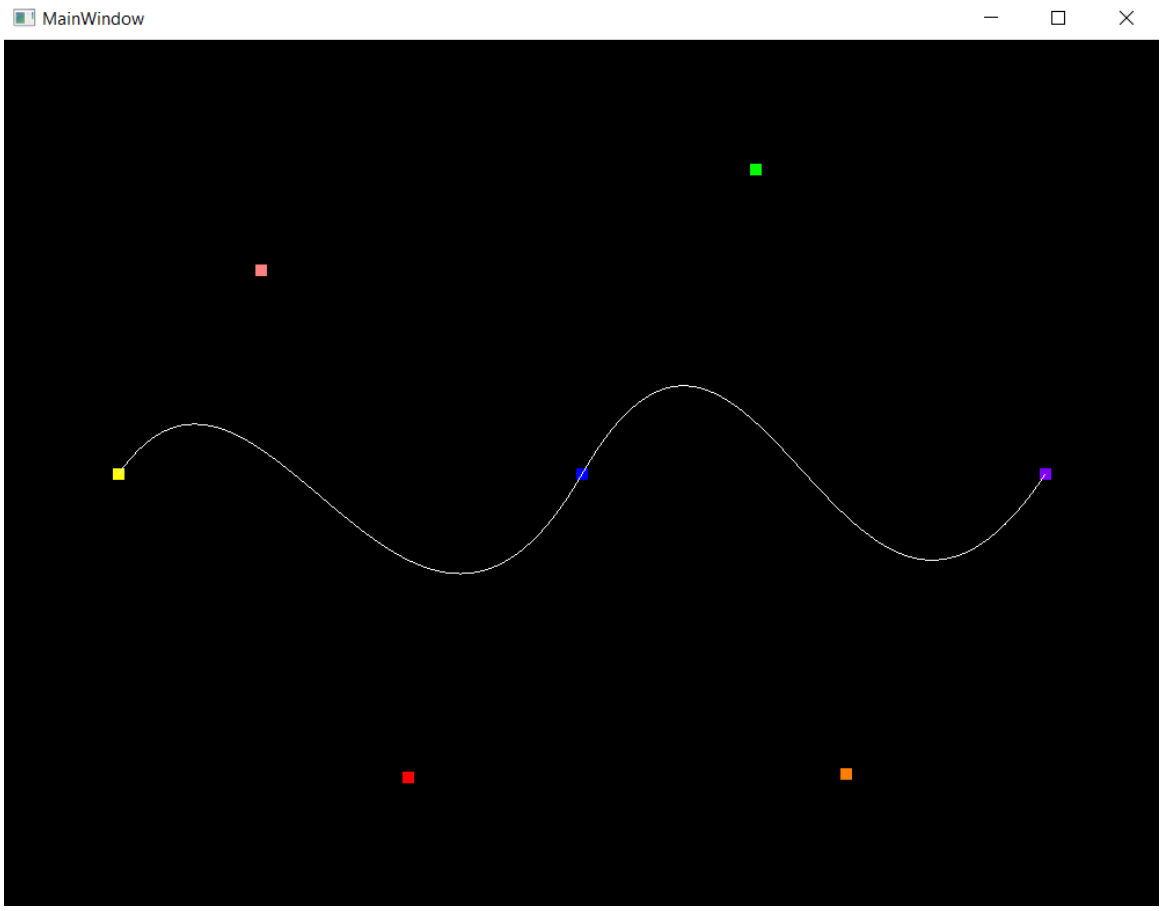


Рисунок 2 — Демонстрация перемещения точек

Вывод.

В ходе выполнения лабораторной работы была создана программа, которая генерирует кривую Безье третьего порядка, состоящую из двух отрезков. Программа обладает интерактивной функцией, позволяющей изменять расположение контрольных точек на кривой.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

design.py

```
from PyQt6 import QtCore, QtGui, QtWidgets
from controller import MyGLWidget,colors

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 600)
        self.centralwidget = QtWidgets.QWidget(parent=MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.openGLWidget = MyGLWidget(parent=self.centralwidget)
        self.openGLWidget.setGeometry(QtCore.QRect(0, 0, 800, 600))
        self.openGLWidget.setObjectName("openGLWidget")
        MainWindow.setCentralWidget(self.centralwidget)
        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow",
"MainWindow"))



if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec())
```

controller.py

```
from PyQt6 import QtCore, QtWidgets
import sys
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout,
QComboBox, QWidget
from PyQt6.QtOpenGLWidgets import QOpenGLWidget
from PyQt6.QtCore import Qt
import numpy as np
from bezier_curve import show_points, bezier_curve, show_curve

colors = {
    'Красная': [1.0, 0.0, 0.0],
    'Синяя': [0.0, 0.0, 1.0],
    'Желтая': [1.0, 1.0, 0.0],
```

```
    'Зеленая': [0.0, 1.0, 0.0],
    'Розовая': [1.0, 0.5, 0.5],
    'Фиолетовая': [0.5, 0.0, 1.0],
    'Оранжевая': [1.0, 0.5, 0.0]
}


class MyGLWidget(QOpenGLWidget):  # OpenGL виджет
    def __init__(self, parent):
        super(MyGLWidget, self).__init__(parent)
        self.current_mode = 'GL_POINTS'
        self.control_points = {
            'Желтая': [-0.8, 0.0, 0.9],
            'Розовая': [-0.5, -0.7, 0.9],
            'Красная': [-0.3, -0.7, 0.8],
            'Синяя': [0.0, 0.0, 0.1],
            'Зеленая': [0.3, 0.7, 0.3],
            'Оранжевая': [0.5, 0.7, 0.3],
            'Фиолетовая': [0.8, 0.0, 0.3]
        }
        self.selected_point = None
        self.bezier_points = []

    def initializeGL(self):  # инициализация
        glEnable(GL_DEPTH_TEST)
        glClearColor(0.0, 0.0, 0.0, 1.0)
        glPointSize(10)
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        width, height = self.width(), self.height()
        glViewport(0, 0, width, height)

        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()

    def paintGL(self):  # рисование
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        show_points(self.control_points, colors)
        points = np.array(list(self.control_points.values()))
        self.bezier_points = bezier_curve(*points)
        show_curve(self.bezier_points)
        self.update()

    def calculate_coordinate(self, pos, total, type: bool):
        if type:
            reflect = 1
        else:
            reflect = -1
        if pos <= total / 2:
            return reflect * (-1 + (pos / (total / 2.0)))
        else:
            return reflect * (pos - total / 2.0) / (total / 2.0)

    def mousePressEvent(self, event):
        x = self.calculate_coordinate(event.pos().x(), self.width(),
True)
        y = self.calculate_coordinate(event.pos().y(), self.height(),
False)
```

```
        for i, point in enumerate(self.control_points.values()):
            if abs(x - point[0]) < 0.1 and abs(y - point[1]) < 0.1:
                self.selected_point = i
                break

    def mouseMoveEvent(self, event):  # перемещение контрольной точки
на экране
        if self.selected_point is not None:
            x = self.calculate_coordinate(event.pos().x(),
self.width(), True)
            y = self.calculate_coordinate(event.pos().y(),
self.height(), False)
            key =
list(self.control_points.keys())[self.selected_point]
            self.control_points[key][0] = x
            self.control_points[key][1] = y
            self.update()

    def mouseReleaseEvent(self, event):  # отмена выбора контрольной
точки
        self.selected_point = None
```

bezier_curve.py

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import numpy as np


def show_points(points, colors):
    glPointSize(10)
    for p in points:
        glColor4f(colors[p][0],    colors[p][1],    colors[p][2],
colors[p][3])
        glBegin(GL_POINTS)
        glVertex2f(points[p][0], points[p][1])
        glEnd()


def show_curve(points):
    glPointSize(3)
    glColor4f(1.0, 1.0, 1.0, 1.0)
    for i in range(1, len(points)):
        glBegin(GL_LINES)
        glVertex2f(points[i - 1, 0], points[i - 1, 1])
        glVertex2f(points[i, 0], points[i, 1])
        glEnd()
    pass


def bezier_curve(p0, p1, p2, p3, p4, p5, p6):  # кривая Безье
    curve_points = np.zeros((200, 2))
    for i, t in enumerate(np.linspace(0, 1, 100)):
        x = (1 - t) ** 3 * p0[0] + 3 * (1 - t) ** 2 * t * p1[0] +
3 * (1 - t) * t ** 2 * p2[0] + t ** 3 * p3[0]
```

```
            y = (1 - t) ** 3 * p0[1] + 3 * (1 - t) ** 2 * t * p1[1] +
3 * (1 - t) * t ** 2 * p2[1] + t ** 3 * p3[1]
            curve_points[i] = [x, y]
        for i, t in enumerate(np.linspace(0, 1, 100)):
            z = (1 - t) ** 3 * p3[0] + 3 * (1 - t) ** 2 * t * p4[0] +
3 * (1 - t) * t ** 2 * p5[0] + t ** 3 * p6[0]
            w = (1 - t) ** 3 * p3[1] + 3 * (1 - t) ** 2 * t * p4[1] +
3 * (1 - t) * t ** 2 * p5[1] + t ** 3 * p6[1]
            curve_points[i + 100] = [z, w]
        return curve_points
```