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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Компьютерная графика»
Тема: Кубические сплайны

Студент гр. 0382 Корсунов А.А.

Преподаватель Герасимова Т.В.

Санкт-Петербург
2023

Цель работы.

Реализовать интерактивное приложение, отображающее заданные полиномиальные кривые.

Теоретические положения.

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.

Неоднородный рациональный B-сплайн, NURBS (Non-uniform rational B-spline) - математическая форма, применяемая в компьютерной графике для генерации и представления кривых и поверхностей. В общем случае B-сплайн состоит из нескольких сплайновых сегментов, каждый из которых определен как набор управляющих точек. Поэтому коэффициенты многочлена будут зависеть только от управляющих точек на рассматриваемом сегменте кривой. Этот эффект называется локальным управлением, поскольку перемещение управляющей точки будет влиять не на все сегменты кривой.

Уравнение NURBS-кривой в общем виде:

$$P(t)=\frac{\sum_{i=0}^{p} B_{i,n}(t) P_i w_i}{\sum_{i=0}^{p} B_{i,n}(t) w_i}$$

где, базовая функция $B_{i,n}$ определена рекурсивно формулами Кокса-де Бура:

$$B_{i,n}(t):\quad B_{i,0}(t)=\begin{cases}1, & t_i \le t < t_{i+1}\\ 0, & \text{иначе}\end{cases}\qquad \forall k>0,\ B_{i,k}(t)=\frac{t-t_i}{t_{i+k}-t_i}B_{i,k-1}(t)+\frac{t_{i+k+1}-t}{t_{i+k+1}-t_{i+1}}B_{i+1,k-1}(t)$$

$w_{i,j}$ – вес, ассоциированный с управляющей точкой $P_{i,j}$ (веса отображают «влияние» контрольной точки на построение кривой, чем больше вес, тем сильнее влияние, если вес равен 0, то контрольную точку можно выкинуть из построения), $1 \le k \le n$, $n = p-1$ – степень полиномов, p – порядок B-сплайна и число сегментов поддержки, $p+1$ – число узлов поддержки.

Сумма базисных функций $\sum_{i=0}^{p} B_{i,n}(t)=1$

Некоторые свойства:

1. NURBS-сплайны являются обобщением с использованием однородных координат B-сплайнов и наследуют большинство их свойств.

2. NURBS кривая в трехмерном пространстве является проекций B-сплайновой кривой в четырехмерном пространстве.

3. Когда степень сплайна становится максимально возможной, сплайн вырождается в рациональную кривую Безье.

4. Если все веса $\omega_i = 1$, тогда сплайн вырождается в B-сплайн.

Задание.

Разработать программу, реализующую NURBS-кривую. n = 6, k = 3. Узловой вектор неравномерный. Веса точек различны и модифицируются.

Ход работы.

Лабораторная работа выполнялась на языке программирования «Python» с использованием модулей «tkinter» и «OpenGL».

1. Итерационная формула для неоднородного рационального B-сплайна (NURBS) 3 степени:

$$P(t)=\frac{w_0 P_0(1-t)^3+3 w_1 P_1 t(1-t)^2+3 w_2 P_2 t^2(1-t)+w_3 P_3 t^3}{w_0(1-t)^3+3 w_1 t(1-t)^2+3 w_2 t^2(1-t)+w_3 t^3}$$

Данная формула является частным случаем формулы, описанной в теоретических положениях.

P_i – контрольные точки, $i = \overline{0,3}$ (в данном случае точки с координатами)

w_i – веса, ставящееся в соответствие контрольным точкам (веса могут быть положительными действительными числами, сумма весов равна единице)

t – параметр, принимающий значения от 0 до 1

- В условии задания дано 6 контрольных точек для кубического NURBS. Чтобы построить такой сплайн по 6 точкам, необходимо строить кривую для каждых четырех контрольных точек (т. е. для трех кривых: с нулевой по третью, с первой по четвертую, со второй по пятую)

2. Алгоритм построения.

а) Чтобы построить NURBS с помощью средств OpenGL можно реализовать функцию, которая бы принимала на вход четыре контрольные точки, соответствующие им веса и строила кривую согласно формуле, описанной в пункте 1.

Функция *curve:*

```
def curve(p0, p1, p2, p3, w0, w1, w2, w3, t_arr):
    len_weights = w0 + w1 + w2 + w3
    w0 = w0 / len_weights
    w1 = w1 / len_weights
    w2 = w2 / len_weights
    w3 = w3 / len_weights

    GL.glBegin(GL.GL_LINE_STRIP)

    for i in range(len(t_arr)):
        t = t_arr[i]
        up_x = w0 * p0[0] * (1 - t) ** 3 + 3 * w1 * p1[0] * t * (1 - t) ** 2 + 3 * w2 * p2[0] * t ** 2 *
(1 - t) + w3 * \
               p3[0] * t ** 3
        down = w0 * (1 - t) ** 3 + 3 * w1 * t * (1 - t) ** 2 + 3 * w2 * t ** 2 * (1 - t) + w3 * t ** 3
        x = up_x / down

        up_y = w0 * p0[1] * (1 - t) ** 3 + 3 * w1 * p1[1] * t * (1 - t) ** 2 + 3 * w2 * p2[1] * t ** 2 *
(1 - t) + w3 * \
               p3[1] * t ** 3
        y = up_y / down

        GL.glVertex3d(x, y, 0)
    GL.glEnd()
```

Функция *curve* принимает 4 контрольных точки (*p0, p1, p2, p3),* веса (*w0, w1, w2, w3),* а также массив узлов, сгенерированный с разным шагом (неравномерно).

Для удобства ввода весов, пользователь может вводить произвольные положительные действительные числа, не беспокоясь, дадут ли они в сумме 1 — в этой функции веса автоматически нормируются (за это отвечает переменная len_weights).

Далее в цикле вычисляются (согласно полученной ранее формуле) x_{n+1} и y_{n+1} координаты, после чего выводятся в окно с помощью примитива LINE_STRIP.

б) Имея функцию, описанную ранее, можно описать алгоритм построения непосредственно:

1) Задаем контрольные точки и веса

2) Строим по контрольным точкам прямые

3) Строим NURBS-кривые, используя функцию *curve*

Части функции draw:

```
GL.glBegin(GL.GL_LINE_STRIP)
for i in range(len(control_points)):
   GL.glColor4f(255, 0, 0, 0)
   GL.glVertex3f(control_points[i][0], control_points[i][1], 0.0)
GL.glEnd()

…

GL.glColor4f(0, 0, 255, 0)

curve(control_points[0], control_points[1], control_points[2], control_points[3], weights[0],
weights[1], weights[2], weights[3], t1_arr)

GL.glColor4f(0, 255, 0, 0)

curve(control_points[1], control_points[2], control_points[3], control_points[4], weights[1],
weights[2],
    weights[3], weights[4], t2_arr)

GL.glColor4f(0, 255, 255, 0)

curve(control_points[2], control_points[3], control_points[4], control_points[5], weights[2],
weights[3],
    weights[4], weights[5], t3_arr)
…
```

В коде выше изначально рисуются прямые с помощью примитива LINE_STRIP, после чего 3 раза вызывается функция отрисовки кривых NURBS.

Таким образом получается следующий результат:

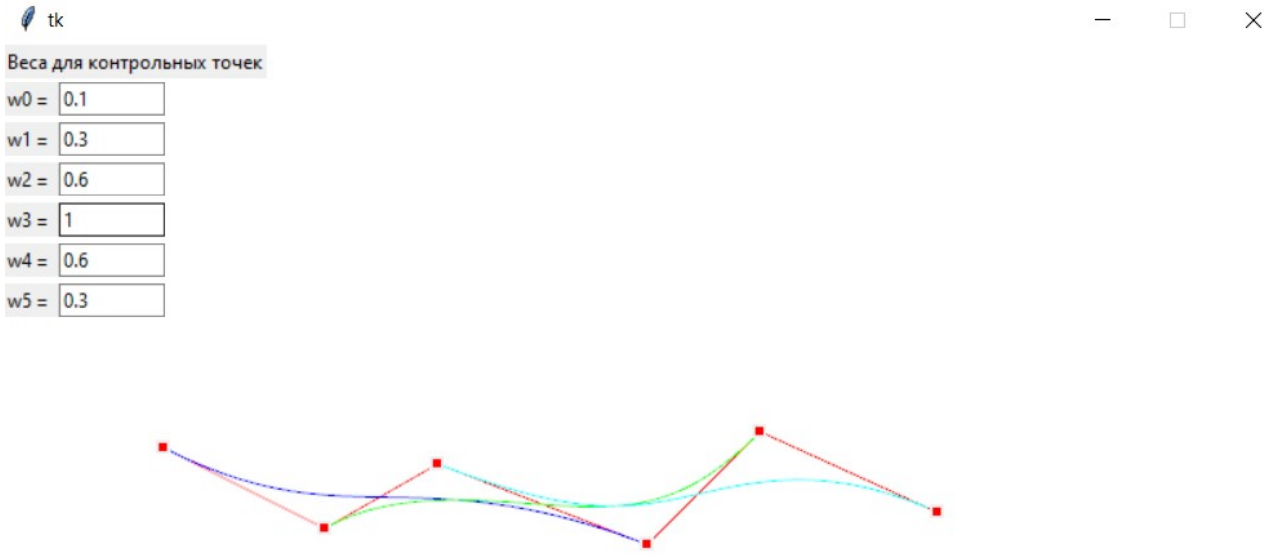

Рисунок 1 — NURBS-кривая по заданным контрольным точкам и их весам

3. Изменение положения контрольных точек в окне, также их весов.

а) Для того, чтобы пользователь мог с помощью мыши изменять координаты контрольных точек, к ним были привязаны canvas-объекты, при изменении положения которых изменяются и координаты контрольных точек.

Функция *drag*:

```
def drag(event):
    canvas_item_id = event.widget.gettags("current")[0]

    mouse_x = app.winfo_pointerx() - app.winfo_rootx()
    mouse_y = app.winfo_pointery() - app.winfo_rooty()
    event.widget.place(x=mouse_x - 5, y=mouse_y - 5)

    control_points[int(canvas_item_id)] = [mouse_x, mouse_y]
```

Функция выше привязывается к каждому canvas на событие нажатия левой кнопки мыши.

Таким образом можно менять контрольные точки следующим образом:

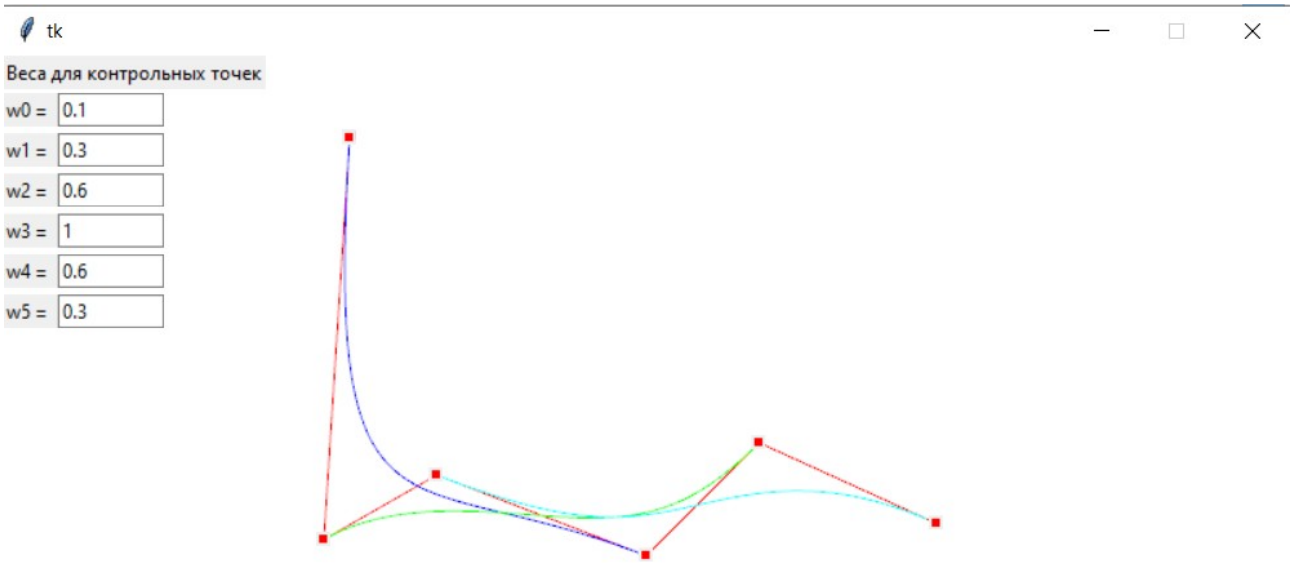

Рисунок 2 — Было изменено положение первой контрольной точки NURBS-кривой из рис. 1

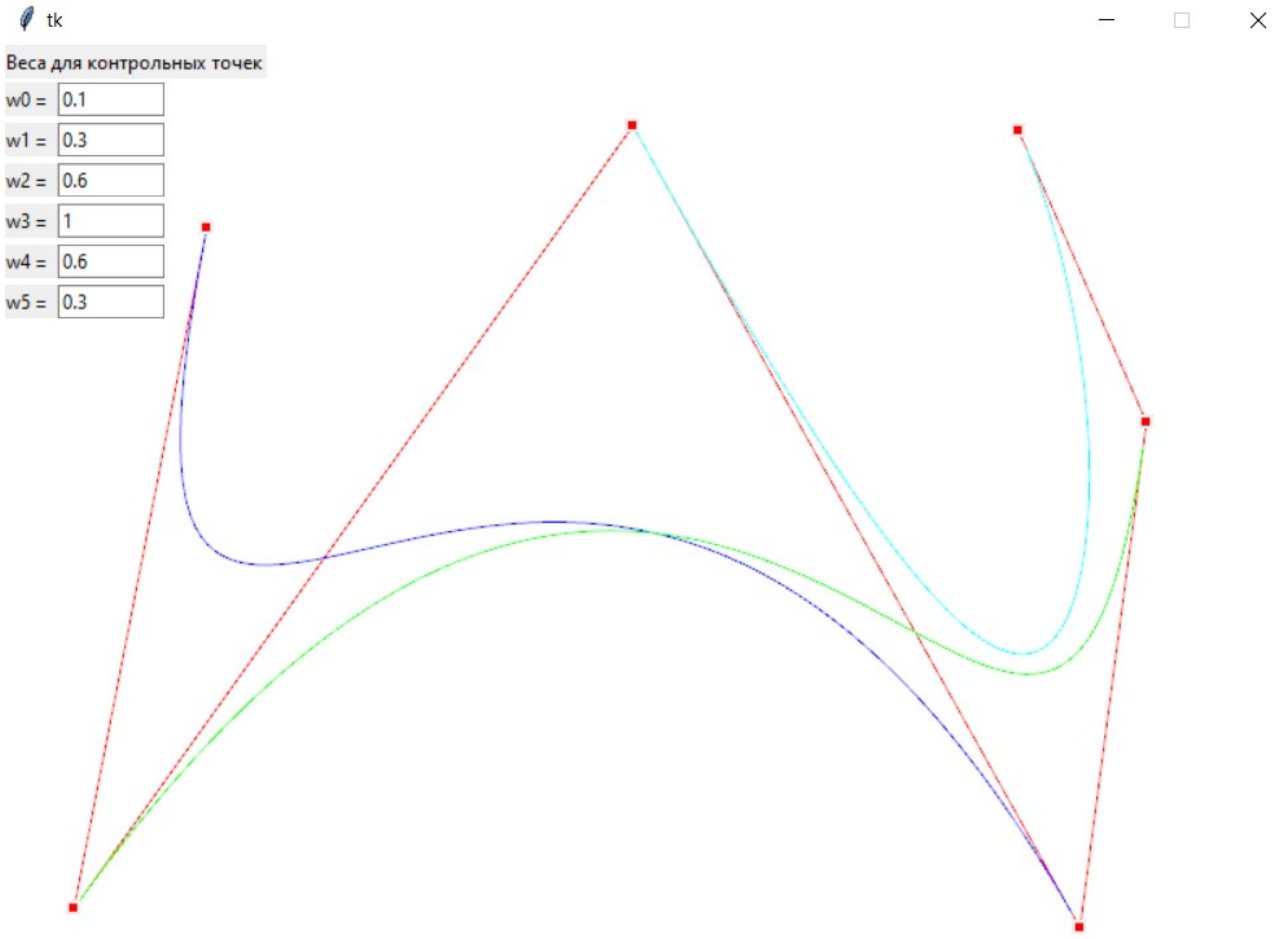

Рисунок 3 — Были изменены положения всех контрольных точек NURBS-кривой из рис.1

б) Для того, чтобы пользователь смог изменять веса, ему предлагается вводить их в специальные виджеты в левой верхней части окна (рис. 3). Эти виджеты реализованы с помощью Entry (из tkinter).

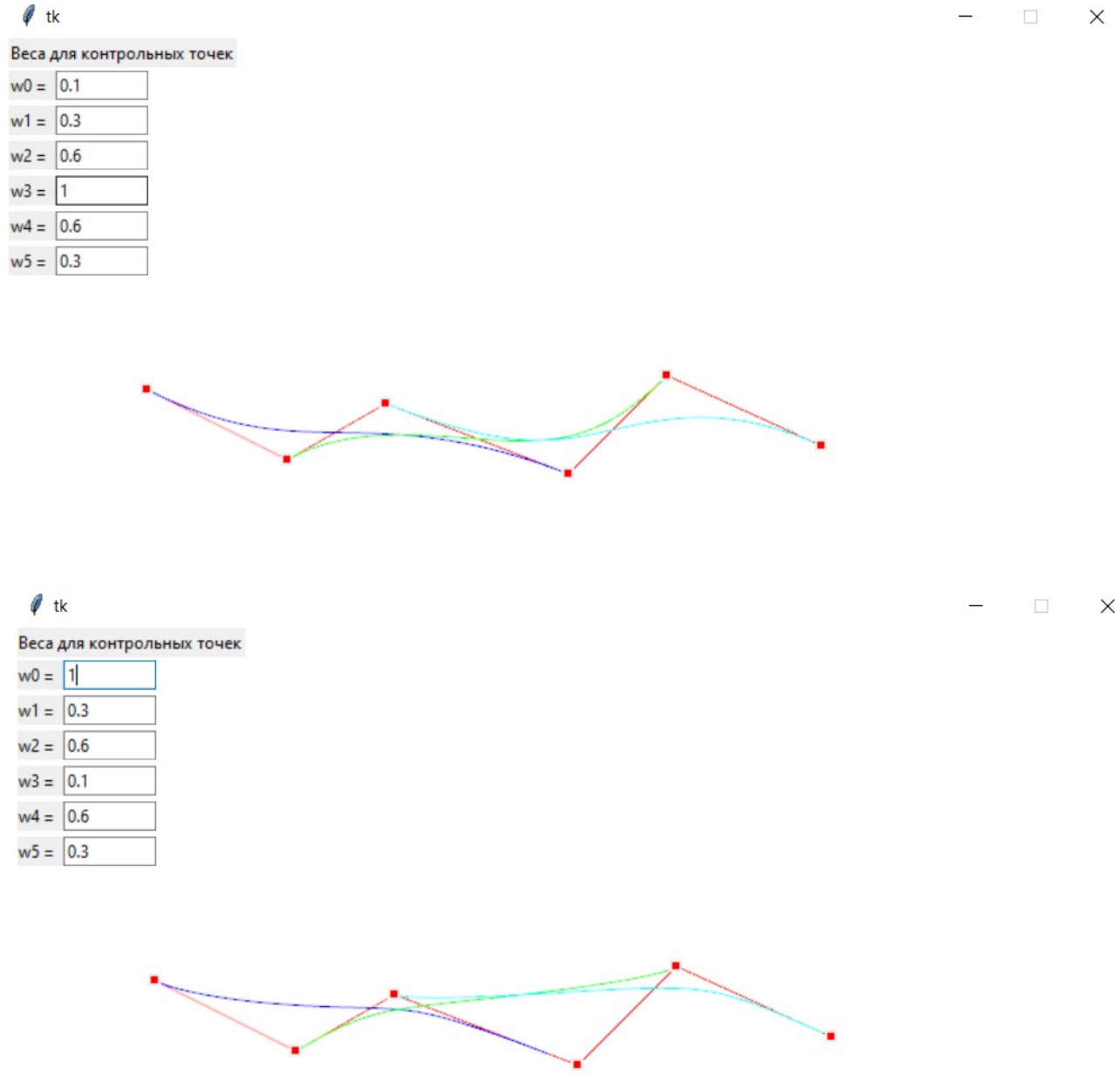

Рисунок 4 — Изменения весов контрольных точек

На рисунке 4 было продемонстрировано изменения весов для нулевой (вес увеличили) и третьей (вес уменьшили) контрольных точек. После изменения весов влияние нулевой контрольной точки на построение было увеличино, а третьей — уменьшено. Этот же вывод можно сделать, взглянув на формулу, выведенную в пункте 1.

Вывод.

Была разработана программа, реализующая интерактивное приложение, отображающее NURBS-кривую.

Приложение А. Исходный код.

Файл nurbs.py

```
from tkinter import *
from pyopengltk import OpenGLFrame
from OpenGL import GL, GLU
from tkinter.ttk import Entry
from random import uniform
import re


def is_valid(newval):
    return re.match("(^\d{0,3})|(^[+]?([.]\d+|\d+([.]\d+)?))$", newval) is not None


def curve(p0, p1, p2, p3, w0, w1, w2, w3, t_arr):
    len_weights = w0 + w1 + w2 + w3
    w0 = w0 / len_weights
    w1 = w1 / len_weights
    w2 = w2 / len_weights
    w3 = w3 / len_weights

    GL.glBegin(GL.GL_LINE_STRIP)

    for i in range(len(t_arr)):
        t = t_arr[i]
        up_x = w0 * p0[0] * (1 - t) ** 3 + 3 * w1 * p1[0] * t * (1 - t) ** 2 + 3 * w2 * p2[0] * t ** 2 *
(1 - t) + w3 * \
               p3[0] * t ** 3
        down = w0 * (1 - t) ** 3 + 3 * w1 * t * (1 - t) ** 2 + 3 * w2 * t ** 2 * (1 - t) + w3 * t ** 3
        x = up_x / down

        up_y = w0 * p0[1] * (1 - t) ** 3 + 3 * w1 * p1[1] * t * (1 - t) ** 2 + 3 * w2 * p2[1] * t ** 2 *
(1 - t) + w3 * \
               p3[1] * t ** 3
        y = up_y / down

        # t += 0.01

        GL.glVertex3d(x, y, 0)
    GL.glEnd()


def draw():  # отрисовка

    GL.glBegin(GL.GL_LINE_STRIP)  # непосредстенно отрисова виджета
    for i in range(len(control_points)):  # в цикле отрисовка вершин по координатам из списка
        GL.glColor4f(255, 0, 0, 0)  # установка цвета
        GL.glVertex3f(control_points[i][0], control_points[i][1], 0.0)
    GL.glEnd()
```

```
    w0 = entry_w0.get()
    w1 = entry_w1.get()
    w2 = entry_w2.get()
    w3 = entry_w3.get()
    w4 = entry_w4.get()
    w5 = entry_w5.get()

    if w0 != "" and w0 != "0" and w0 != "0.":
        w0 = float(w0)
        if w0 <= 100:
            weights[0] = w0
    if w1 != "" and w1 != "0" and w1 != "0.":
        w1 = float(w1)
        if w1 <= 100:
            weights[1] = w1
    if w2 != "" and w2 != "0" and w2 != "0.":
        w2 = float(w2)
        if w2 <= 100:
            weights[2] = w2
    if w3 != "" and w3 != "0" and w3 != "0.":
        w3 = float(w3)
        if w3 <= 100:
            weights[3] = w3
    if w4 != "" and w4 != "0" and w4 != "0.":
        w4 = float(w4)
        if w4 <= 100:
            weights[4] = w4
    if w5 != "" and w5 != "0" and w5 != "0.":
        w5 = float(w5)
        if w5 <= 100:
            weights[5] = w5

    GL.glColor4f(0, 0, 255, 0)

    curve(control_points[0], control_points[1], control_points[2], control_points[3], weights[0],
weights[1],
          weights[2], weights[3], t1_arr)

    GL.glColor4f(0, 255, 0, 0)

    curve(control_points[1], control_points[2], control_points[3], control_points[4], weights[1],
weights[2],
          weights[3], weights[4], t2_arr)

    GL.glColor4f(0, 255, 255, 0)

    curve(control_points[2], control_points[3], control_points[4], control_points[5], weights[2],
weights[3],
          weights[4], weights[5], t3_arr)

    GL.glFlush()
```

```
def drag(event):
    canvas_item_id = event.widget.gettags("current")[0]

    mouse_x = app.winfo_pointerx() - app.winfo_rootx()
    mouse_y = app.winfo_pointery() - app.winfo_rooty()
    event.widget.place(x=mouse_x - 5, y=mouse_y - 5)

    control_points[int(canvas_item_id)] = [mouse_x, mouse_y]


class DrawingWindow(OpenGLFrame):  # создание класса на основе пакета pyopengltk

    def initgl(self):  # инициализация
        GL.glClear(GL.GL_COLOR_BUFFER_BIT)  # очистка буфферов от цветов ~ очищение экарана
        GL.glClearColor(1, 1, 1, 0)  # задает цвет, в который окно будет окрашиваться при его очистке ~ очищение
        # цветопередачи
        GL.glMatrixMode(GL.GL_PROJECTION)  # матрица проекции (для проецирования 3D прсостранства в 2D)
        GL.glLoadIdentity()  # единичная матрица ~ очистика
        GLU.gluOrtho2D(0, window_width, window_height, 0)  # смещецение оси координат (чтобы не возникало искажения при
        # отрисовке

    def redraw(self):  # перерисовка
        GL.glClear(GL.GL_COLOR_BUFFER_BIT)
        draw()  # функция для отрисовки


t1 = 0
t1_arr = [0]
while t1 < 1:
    temp = uniform(0, 0.001)
    t1 += temp
    t1_arr.append(t1)

t2 = 0
t2_arr = [0]
while t2 < 1:
    temp = uniform(0, 0.001)
    t2 += temp
    t2_arr.append(t2)

t3 = 0
t3_arr = [0]
while t3 < 1:
    temp = uniform(0, 0.001)
    t3 += temp
    t3_arr.append(t3)
```

```
root = Tk()  # главное окно
root.resizable(False, False)

window_width = 800  # размеры окна (ширина и высота)
window_height = 600
coordinates = []  # (координаты вершин)

control_points = [[100, 250], [200, 300], [270, 260], [400, 310], [470, 240], [580, 290]]
weights = [0.1, 0.3, 0.6, 1, 0.6, 0.3]

app = DrawingWindow(root, width=window_width, height=window_height)  # создание окна для отрисови
app.pack(fill=BOTH, expand=YES)  # отобразить
app.animate = 1  # для отрисовки в реальном времени

Point_0 = Canvas(app, width=5, height=5)
Point_0.place(x=control_points[0][0] - 5, y=control_points[0][1] - 5)
Point_0.create_rectangle(0, 0, 10, 10, fill="red", tags="0")

Point_1 = Canvas(app, width=5, height=5)
Point_1.place(x=control_points[1][0] - 5, y=control_points[1][1] - 5)
Point_1.create_rectangle(0, 0, 10, 10, fill="red", tags="1")

Point_2 = Canvas(app, width=5, height=5)
Point_2.place(x=control_points[2][0] - 5, y=control_points[2][1] - 5)
Point_2.create_rectangle(0, 0, 10, 10, fill="red", tags="2")

Point_3 = Canvas(app, width=5, height=5)
Point_3.place(x=control_points[3][0] - 5, y=control_points[3][1] - 5)
Point_3.create_rectangle(0, 0, 10, 10, fill="red", tags="3")

Point_4 = Canvas(app, width=5, height=5)
Point_4.place(x=control_points[4][0] - 5, y=control_points[4][1] - 5)
Point_4.create_rectangle(0, 0, 10, 10, fill="red", tags="4")

Point_5 = Canvas(app, width=5, height=5)
Point_5.place(x=control_points[5][0] - 5, y=control_points[5][1] - 5)
Point_5.create_rectangle(0, 0, 10, 10, fill="red", tags="5")

Point_0.bind("<B1-Motion>", drag)
Point_1.bind("<B1-Motion>", drag)
Point_2.bind("<B1-Motion>", drag)
Point_3.bind("<B1-Motion>", drag)
Point_4.bind("<B1-Motion>", drag)
Point_5.bind("<B1-Motion>", drag)

check_valid = (app.register(is_valid), "%P")
label_Entry = Label(text="Веса для контрольных точек")
label_Entry.place(x=0, y=0)

label_w0 = Label(text="w0 = ")
```

```
label_w0.place(x=0, y=23)
entry_w0 = Entry(width=10, validate="key", validatecommand=check_valid)
entry_w0.place(x=35, y=23)
entry_w0.insert(0, str(weights[0]))

label_w1 = Label(text="w1 = ")
label_w1.place(x=0, y=48)
entry_w1 = Entry(width=10, validate="key", validatecommand=check_valid)
entry_w1.place(x=35, y=48)
entry_w1.insert(0, str(weights[1]))

label_w2 = Label(text="w2 = ")
label_w2.place(x=0, y=73)
entry_w2 = Entry(width=10, validate="key", validatecommand=check_valid)
entry_w2.place(x=35, y=73)
entry_w2.insert(0, str(weights[2]))

label_w3 = Label(text="w3 = ")
label_w3.place(x=0, y=98)
entry_w3 = Entry(width=10, validate="key", validatecommand=check_valid)
entry_w3.place(x=35, y=98)
entry_w3.insert(0, str(weights[3]))

label_w4 = Label(text="w4 = ")
label_w4.place(x=0, y=123)
entry_w4 = Entry(width=10, validate="key", validatecommand=check_valid)
entry_w4.place(x=35, y=123)
entry_w4.insert(0, str(weights[4]))

label_w5 = Label(text="w5 = ")
label_w5.place(x=0, y=148)
entry_w5 = Entry(width=10, validate="key", validatecommand=check_valid)
entry_w5.place(x=35, y=148)
entry_w5.insert(0, str(weights[5]))

app.mainloop()  # запуск главного цикла
```