МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет
«ЛЭТИ» им. В.И. Ульянова (Ленина)
Кафедра систем автоматизированного проектирования (САПР)

отчет
по лабораторной работе № 2
по дисциплине «Компьютерная графика»
Тема: **«Формирования различных кривых с использованием ортогонального проектирования на плоскость визуализации (экране дисплея)»**

	Аршин А. Д
	Баймухамедов Р. Р.
Студенты гр. 3311	Пасечный Л. В.
Преподаватель	Колев Г. Ю.

Санкт-Петербург
2025

Цель работы

Исследование и формирование различных кривых с использованием ортогонального проектирования на плоскость визуализации (экране дисплея)

Задание (вариант 9):

Сформировать на плоскости B-сплайновую кривую различной степени (1, 2, 3, 4, 5, 6) на основе 7 не повторяющихся задающих точек. Обеспечить редактирование координат задающих точек с перерисовкой сплайна

Теоретическая основа

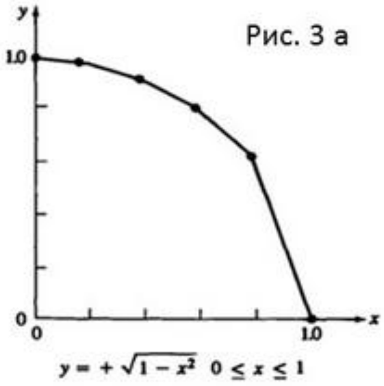
ПАРАМЕТРИЧЕСКИЕ КРИВЫЕ

- В параметрическом виде каждая координата точки кривой представлена как функция одного параметра. Значение параметра задает координатный вектор точки на кривой. Для двумерной кривой с параметром t координаты точки равны:
- x =x(t), y = y(t).
- Тогда векторное представление точки на кривой:
- P(t) = [x(t) y(t)]
- Параметрическая форма позволяет представить замкнутые и многозначные кривые. Производная, т. е. касательный вектор, есть
- P'(t) = [x'(t) y'(t)], где ' - дифференцирование по параметру.
- Наклон кривой, dy/dx, равен
- При x'(t) = 0 наклон бесконечен.

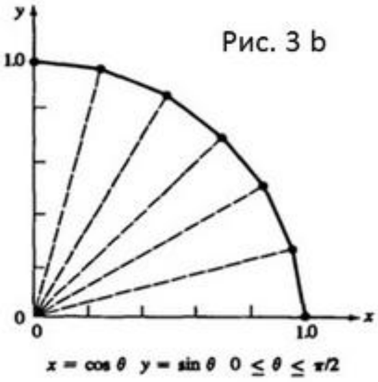
$$\frac{dy}{dx} = \frac{dy/dt}{dx/dt} = \frac{y'(t)}{x'(t)}$$

- Параметрическое представление не вызывает в этом случае вычислительных трудностей, достаточно приравнять нулю одну компоненту касательного вектора.
- Точка на параметрической кривой определяется только значением параметра, не зависит от выбора системы координат.
- Конечные точки и длина кривой определяются диапазоном изменения параметра.
- Удобно нормализовать параметр на интересующем отрезке кривой к 0 < t < 1.
- Осенезависимость параметрической кривой позволяет проводить с ней аффинные преобразования, рассмотренные ранее.

- Стандартная параметрическая форма единичной окружности:
- x = cosθ, 0≤θ≤2π,
- y = sinθ,
- или
- P(θ) = [x y] = [cosθ sinθ], 0≤θ≤2π,
- где параметр θ — геометрический угол, отмеряемый
- против часовой стрелки от положительной полуоси x.
- На рис. 3 сравниваются непараметрическое
- и параметрическое представления окружности
- в первом квадранте.

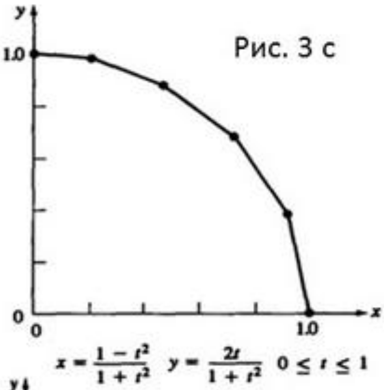


•Непараметрический вид (рис. 3 a). Точки на дуге соответствуют равным приращениям **x**. Дуга состоит из отрезков разной длины – получается приблизительное графическое представление окружности. Кроме того, расчет квадратного корня –вычислительно дорогостоящая операция.
•На рис. 3 b изображена дуга, построенная по равным приращениям параметра **θ** в пределах 0≤θ≤π/2. Точки располагаются на одинаковом расстоянии вдоль окружности, и окружность выглядит гораздо лучше. Недостаток такого представления — сложность вычисления тригонометрических функций.



- Параметрическом представление кривой не единственно, например,
- $$P(t) = \left[\frac{(1-t^2)}{(1+t^2)} \quad \frac{2t}{(1+t^2)}\right], \quad 0 \le t \le 1 \qquad (1)$$
- также представляет дугу единичной окружности в первом квадранте (рис. 3 c).
- На рис. 3 c показан результат для равных приращений t. Он лучше, чем у явного, но хуже, чем у стандартного параметрического представления. Однако уравнение (1) проще с вычислительной точки зрения, т.е. это компромиссное решение.
- Связь между параметрическим представлением и стандартным параметрическим представлением показана на рис. 4. Из него видно, что для единичной окружности

$$x = \cos\theta = \frac{1-t^2}{1+t^2}, \qquad 0 \le \theta \le \pi/2, \qquad 0 \le t \le 1,$$
$$y = \sin\theta = \frac{2t}{1+t^2}, \qquad 0 \le \theta \le \pi/2, \qquad 0 \le t \le 1.$$



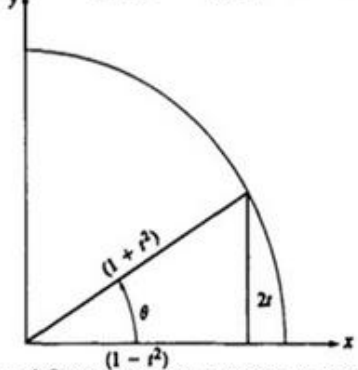
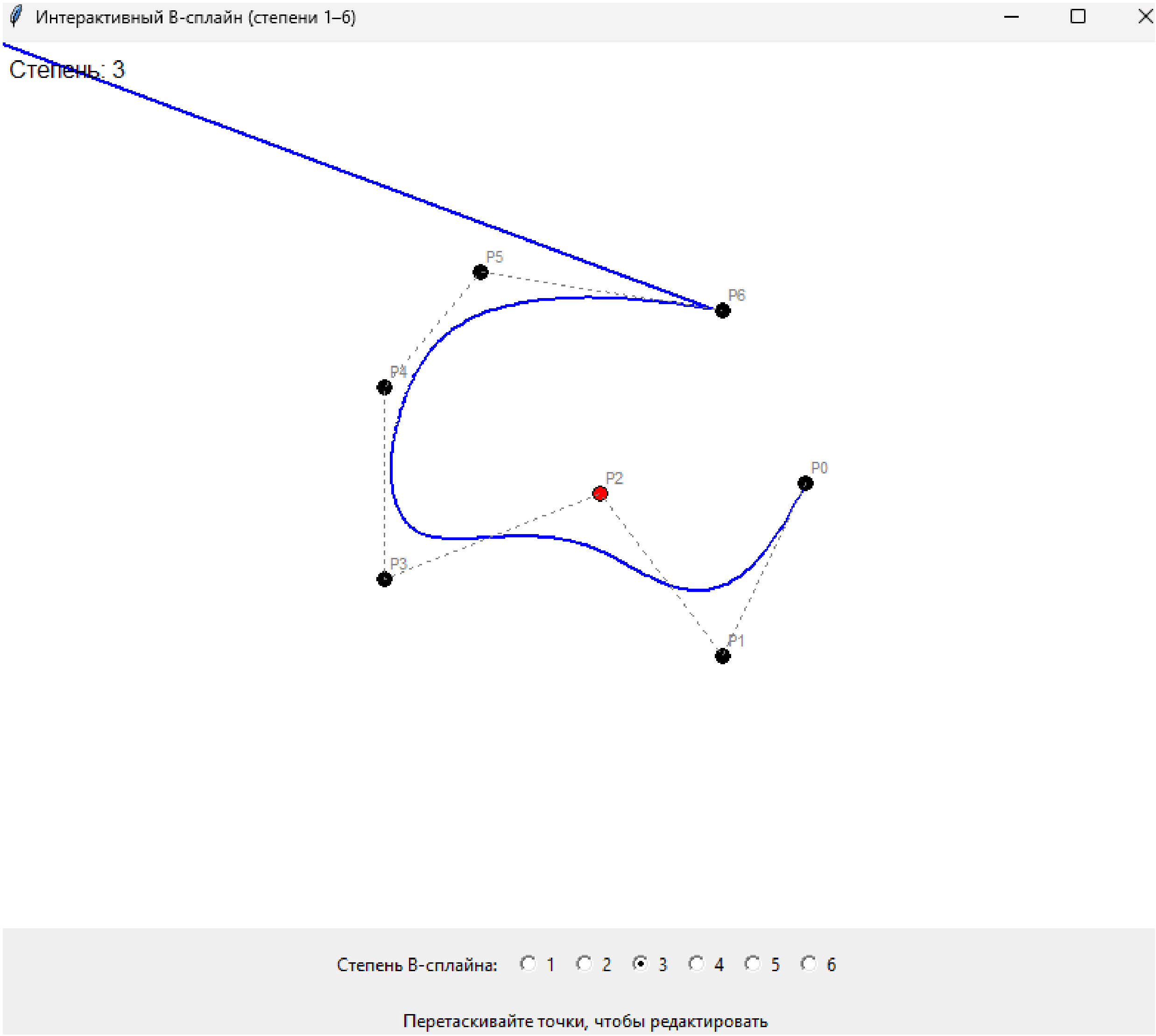


Рис. 4 Связь между параметрическими представлениями.

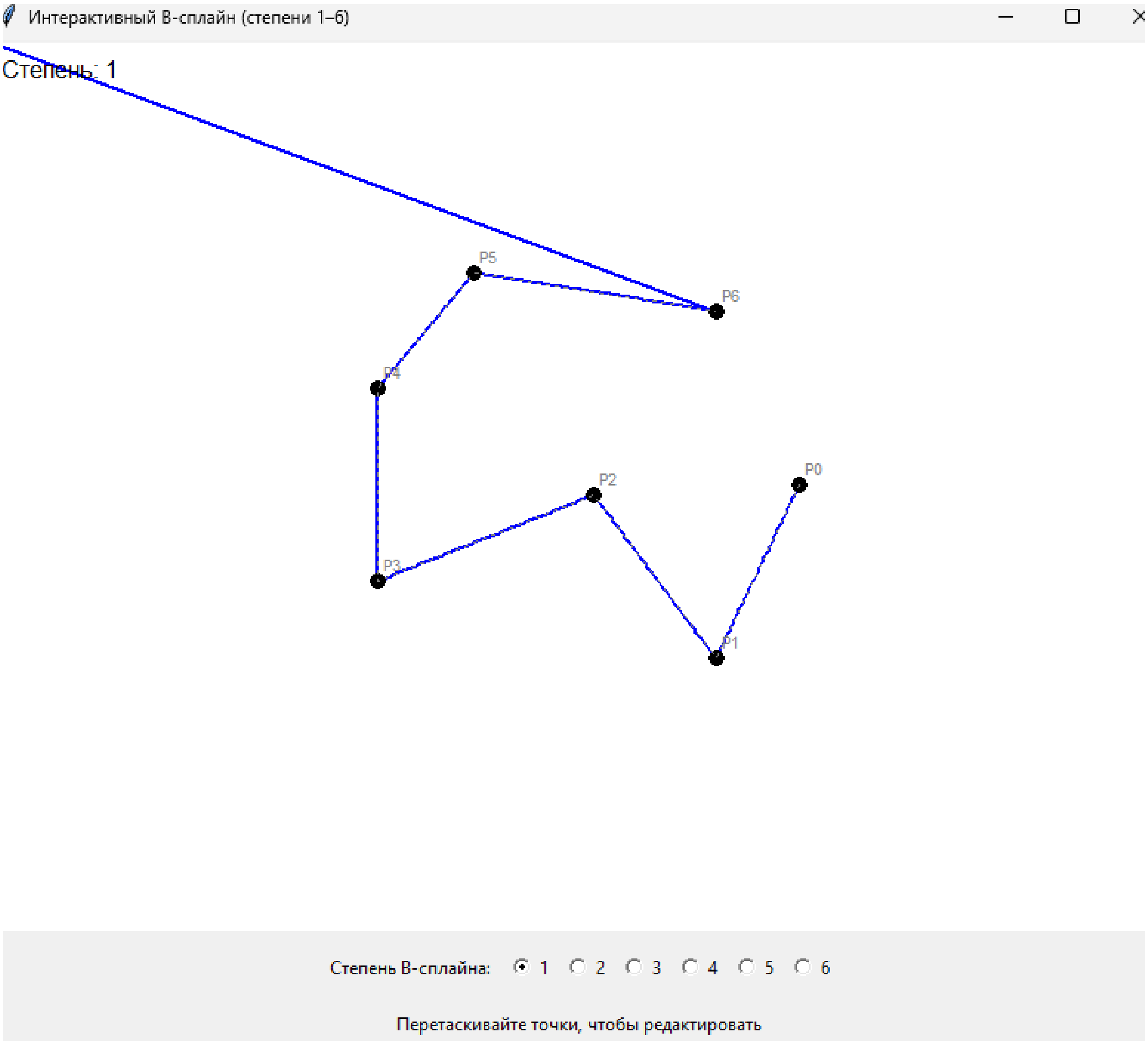
Выполнение лабораторной работы

В качестве языка программирования для выполнения данной лабораторной работы выберем Python. Для быстрого запуска достаточно иметь установленный Python 3 (модули tkinter и math, которые входят в стандартную библиотеку). Демонстрация работы программы (https://youtu.be/Nyw5CAazh-I)

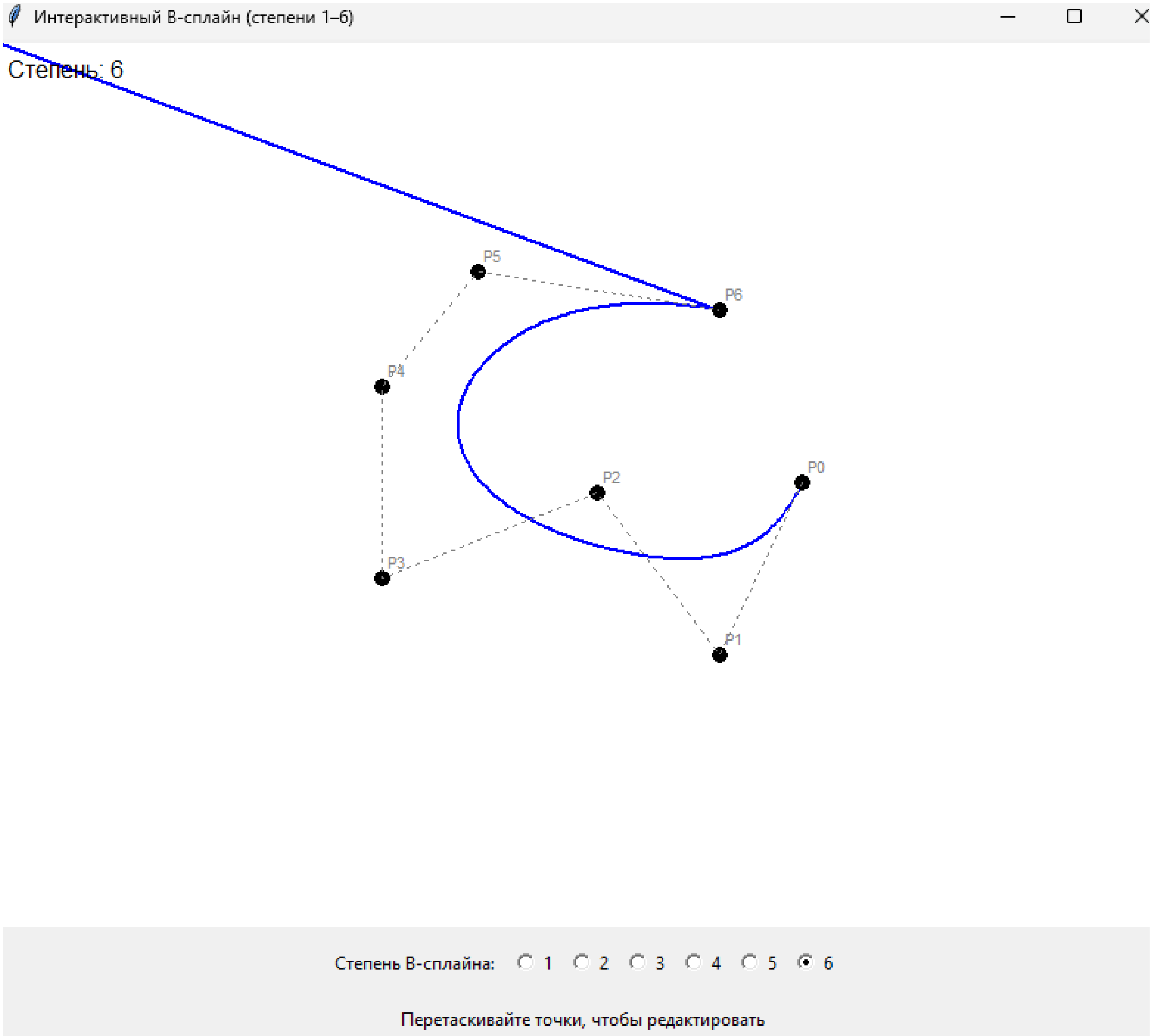
Интерфейс программы



Демонстрация выполненной лабораторной работы №1. Степень B-сплайна равно 3



Демонстрация выполненной лабораторной работы №1. Степень B-сплайна равно 1



Демонстрация выполненной лабораторной работы №1. Степень B-сплайна равно 6

Вывод

Реализовано интерактивное приложение для построения B-сплайнов степени 1–6 по семи управляющим точкам с возможностью перетаскивания и моментальной перерисовки. При степени 1 кривая совпадает с ломаной, с ростом степени она становится заметно плавнее и шире область влияния контрольных точек, при этом крайние точки фиксируются на концах кривой. Работа наглядно подтверждает основные идеи параметрических кривых и служит удобным инструментом для исследования влияния степени и расположения точек.

Код программы

```
import tkinter as tk
import math

class BSplineApp:
    # инициализация окна, добавление текста и привязка мыши к реакции
    def __init__(self, root):
        self.root = root
        self.root.title("Интерактивный B-сплайн (степени 1-6)")
        self.canvas_width = 800
        self.canvas_height = 600
        # создаем холст для области рисования и добавляем в окно
        self.canvas = tk.Canvas(root, width=self.canvas_width,
height=self.canvas_height, bg="white")
        self.canvas.pack()

        # вычисляем центр холста и расставляем 7 точек в радиусе 150 пикселей
        cx, cy = self.canvas_width // 2, self.canvas_height // 2
        radius = 150
        self.control_points = [
            (cx + radius * math.cos(2 * math.pi * i / 7), cy + radius *
math.sin(2 * math.pi * i / 7))
            for i in range(7)
        ]
        # self.selected_point индекс точки, которую сейчас тащят мышью или none
если нет действий
        self.selected_point = None
        self.degree = 3  # по умолчанию кубический

        # создает рамку фрейм где у нас располагаются инструменты для
редактирования степени
        control_frame = tk.Frame(root)
        control_frame.pack(pady=10)

        tk.Label(control_frame, text="Степень B-сплайна:").pack(side=tk.LEFT,
padx=5)
        # создаем degree_var для связи с радиокнопками которые будут иметь 6
степеней при их выборе вызывается on_degree_change
        self.degree_var = tk.IntVar(value=self.degree)
        for d in range(1, 7):
            tk.Radiobutton(control_frame, text=str(d), variable=self.degree_var,
value=d,
                           command=self.on_degree_change).pack(side=tk.LEFT,
padx=2)

        tk.Label(root, text="Перетаскивайте точки, чтобы
редактировать").pack(pady=5)

        # привязка событий мыши
        self.canvas.bind("<Button-1>", self.on_click)# нажатие мыши
```

```
        self.canvas.bind("<B1-Motion>", self.on_drag)# нажатие мыши и движение
        self.canvas.bind("<ButtonRelease-1>", self.on_release)# отпусскание мыши
        # рисуем все в первый раз
        self.draw_all()

    # при смене степени обновляем self.degree и перерисовываем
    def on_degree_change(self):
        self.degree = self.degree_var.get()
        self.draw_all()

    # проверяем кликнули ли в пределах 10 пикселей от какой-то точки
    def on_click(self, event):
        for i, (x, y) in enumerate(self.control_points):
            if (x - event.x) ** 2 + (y - event.y) ** 2 <= 100:  # 10^2
                self.selected_point = i #запоминаем ее индекс
                break
    # если точка выбрана перересовываем и оновляем координаты
    def on_drag(self, event):
        if self.selected_point is not None:
            self.control_points[self.selected_point] = (event.x, event.y)
            self.draw_all()
    # когда отпустили сбрасываем выбор
    def on_release(self, event):
        self.selected_point = None

    # B-сплайн реализация

    # n-индекс последней управляющей точки
    # p-степень сплайна
    # начинаем узловой вектор с p+1 нулей (для clamped-сплайна)
    def make_knot_vector(self, n, p):
        m = n + p + 1
        knot = [0] * (p + 1)
        # сколько внутренних узлов нужно добавить между 1 и 0
        inner_knots = m - 2 * (p + 1) + 1
        # добавляем равномерные внутренние узлы
        if inner_knots > 0:
            step = 1.0 / (inner_knots + 1)
            for i in range(1, inner_knots + 1):
                knot.append(i * step)
        else:
            # если точек мало для внутренних узлов просто повторяем
            pass
        # завершаем p+1 единицами
        knot += [1.0] * (p + 1)
        return knot

    def basis_function(self, i, p, t, knot):
        # базовый случай - степень 0 -> функция равна 1, если t в интервале
        if p == 0:
            return 1.0 if knot[i] <= t < knot[i + 1] else 0.0
```

```
        # рекурсивно вычисляем первую часть формулы кокса–де бура
        else:
            denom1 = knot[i + p] - knot[i]
            c1 = 0.0
            if denom1 > 1e-10:
                c1 = (t - knot[i]) / denom1 * self.basis_function(i, p - 1, t,
knot)

            # вторая часть + сумма -> полная базисная функция.
            denom2 = knot[i + p + 1] - knot[i + 1]
            c2 = 0.0
            if denom2 > 1e-10:
                c2 = (knot[i + p + 1] - t) / denom2 * self.basis_function(i + 1,
p - 1, t, knot)

            # соединяем
            return c1 + c2
    # для параметра t вычисляем взвешенную сумму управляющих точек с весами
N_{i,p}(t)
    def evaluate_bspline(self, t, control_points, p, knot):
        n = len(control_points) - 1  # индекс последней точки
        x = y = 0.0
        for i in range(n + 1):
            N = self.basis_function(i, p, t, knot)
            xi, yi = control_points[i]
            x += N * xi
            y += N * yi
        return x, y
    # если степень слишком высока (например, 6 точек → макс. степень 5),
ограничиваем
    def generate_curve_points(self, control_points, p, num_samples=300):
        n = len(control_points) - 1
        if p > n:
            p = n  # нельзя выше, чем n
        # диапазон параметра t от knot[p] до knot[n+1]
        knot = self.make_knot_vector(n, p)
        t_start = knot[p]
        t_end = knot[n + 1]
        # защита от деления на ноль
        if abs(t_end - t_start) < 1e-10:
            return []
        # генерируем 301 точку (включая концы) для плавной кривой.
        curve_points = []
        for i in range(num_samples + 1):
            t = t_start + (t_end - t_start) * i / num_samples
            # обработка конечной точки
            if i == num_samples:
                t = t_end
            pt = self.evaluate_bspline(t, control_points, p, knot)
            curve_points.append(pt)
```

```
        return curve_points

    # Отрисовка

    def draw_all(self):
        # стираем все на холсте
        self.canvas.delete("all")

        # генерация и отрисовка B-сплайна
        # преобразуем список [(x1,y1), (x2,y2), ...] в плоский список
[x1,y1,x2,y2,...]
        curve_pts = self.generate_curve_points(self.control_points, self.degree)
        if len(curve_pts) > 1:
            flat = [coord for pt in curve_pts for coord in pt]
            self.canvas.create_line(flat, fill="blue", width=2, smooth=False)

        # рисуем каждую управляющую точку: чёрная или красная
        for i, (x, y) in enumerate(self.control_points):
            # точка
            r = 5
            color = "red" if i == self.selected_point else "black"
            self.canvas.create_oval(x - r, y - r, x + r, y + r, fill=color,
outline="black")
            # подпись
            self.canvas.create_text(x + 10, y - 10, text=f"P{i}", fill="gray",
font=("Arial", 8))

        # рисуем серую пунктирную полилинию между управляющими точками
        if len(self.control_points) > 1:
            flat_ctrl = [coord for pt in self.control_points for coord in pt]
            self.canvas.create_line(flat_ctrl, fill="gray", dash=(3, 3), width=1)

        # подпись степени
        self.canvas.create_text(50, 20, text=f"Степень: {self.degree}",
fill="black", font=("Arial", 12))


# запуск
if __name__ == "__main__":
    root = tk.Tk()
    app = BSplineApp(root)
    root.mainloop()
```