**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)»**

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

Санкт-Петербург

2025

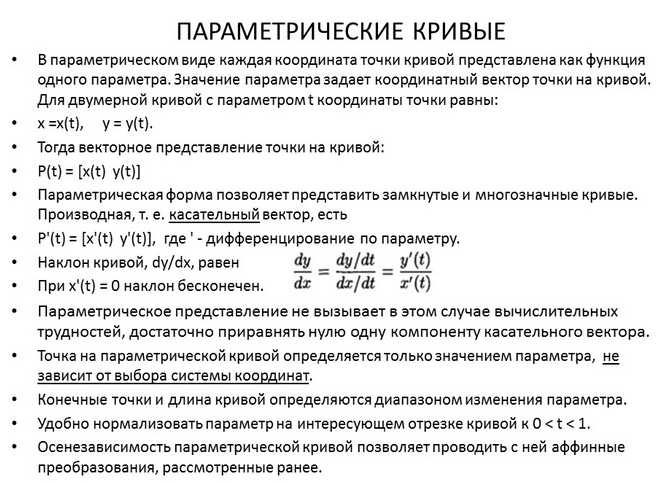
**Цель работы**

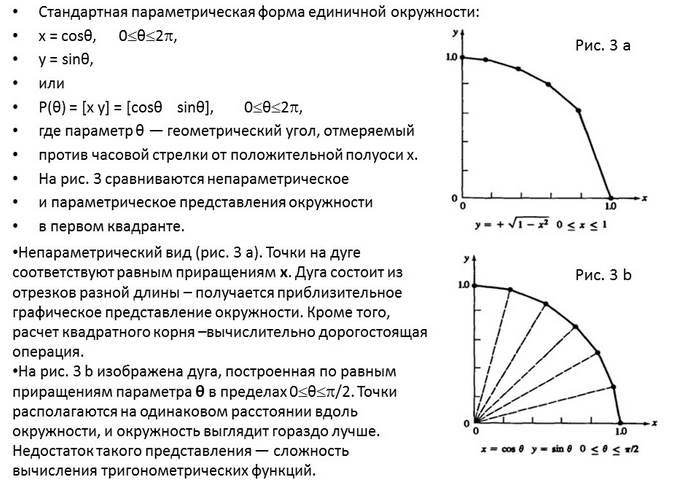
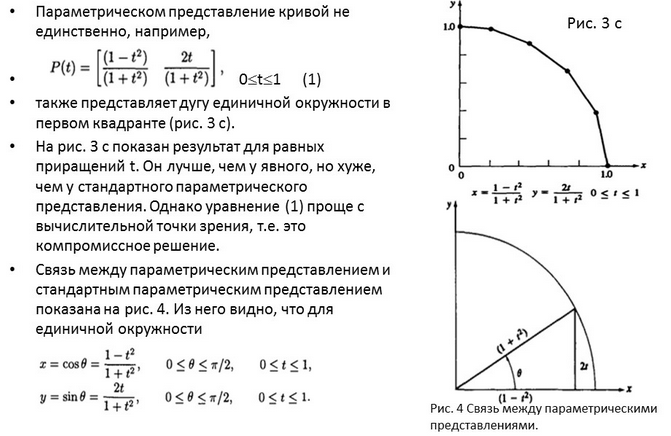
Исследование и формирование различных кривых с использованием ортогонального проектирования на плоскость визуализации (экране дисплея)

**Задание (вариант 9):**

Сформировать на плоскости В-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**



**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 степений 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()