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

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

по дисциплине Компьютерная графика

Тема: «Формирования различных кривых с использованием ортогонального проектирования на плоскость визуализации (экране дисплея)

Студенты гр. 1307		Грунская Н.Д. Тростин М.Ю. Голубев М.А.
Преподаватель		Матвеева И.В.

Санкт-Петербург

2024

Цель работы:
Практическое закрепление теоретических знаний о формировании кривых с использованием ортогонального проектирования на плоскость визуализации.

Постановка задачи:

Сформировать на плоскости кривую Безье на основе задающей ломаной, определяемой 3 и большим количеством точек. Обеспечить редактирование координат точек задающей ломаной с перерисовкой сплайна Безье.

Краткая теоретическая информация:

Точки задания этих кривых Безье только определяют ход кривой, сама строящаяся кривая в общем случае не проходит через внутренние точки задающего многоугольника

Особенности:

1. Подходит по касательной к внешним ребрам (сторонам) задающего многоугольника, а остальные точки определяют ход кривой. Они позволяют качественно оценить ход кривой в зависимости от вида задающего многоугольника.
2. Кривая задается параметрически в функции от независимого параметра.
3. Это кривая n-ой степени, т.е. сколько ребер у задающего многоугольника – такой степени и получается кривая. Влиять на степень кривой можно только изменением количества задающих ее точек.

Математически такая кривая описывается параметрическим уравнением:

$P(t) = \sum_{i=0}^{n} P_i \times N_{i,n}(t)$,где P(t) – полиномиальная функция,
Pi – вес (координаты) i-ой точки задания,
Ni, n – весовой коэффициент i-той вершины,
i – номер вершины (точки),
n – количество сторон задающего многоугольника
t – задающий параметр, причем 0 <= t <= 1

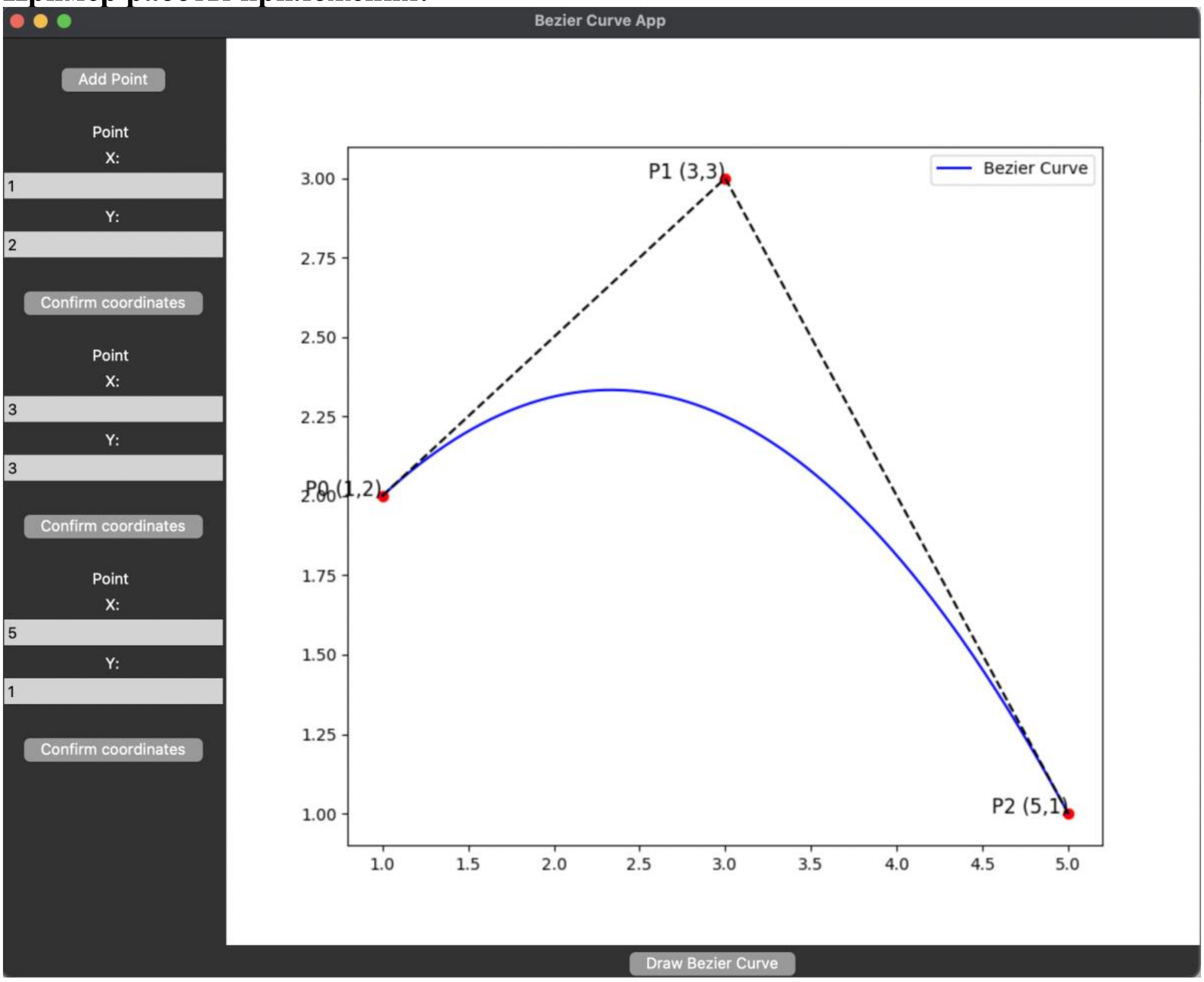
$$N_{i,n}(t) = \frac{n!}{i!(n-i)!} \times t^i \times (1-t)^{n-}$$

Если множество точек задания (n), то строим либо кривую n-ой степени, либо можем построить кривую невысокой степени (обычно кубическую, так как она позволяет обеспечить перегиб и может быть быстро построена). В последнем случаедля такой кривой нужно только четыре последовательные точки задания.

Если n > 4, то мы находимся в противоречии с количеством точек, так как формируя сегменты на основе точек задающего многоугольника в точках стыковки таких сплайнов кривая будет иметь излом (разрыв производных). Чтобы избежать такой ситуации учитывают то свойство кривых Безье, что к внешним ребрам задающего много угольника они подходят по касательной. Поэтому в третье ребро описания при формировании кубической кривой Безье добавляют дополнительную точку P` и ее считают за последнею точку текущего характеристического многоугольника и первой точкой следующего характеристического многоугольника. Обычно ее берут по середине ребра. Тогда кривая будет плавно переходить от предыдущей кубической кривой (кривой третьей степени) к последующей. А вся такая кривая представляет собой составную плавную кривую, состоящую из ряда сегментов, называется составной кривой Безье.

Для расчета последнего сегмента такой составной кривой Безье можно либо понизить степень строящегося участка кривой до второй, так как может остаться только три незадействованной точки, либо при расчете использовать последнюю точку дважды.

Пример работы приложения:



Выводы:

Были практически закреплены теоретические знания о формировании кривых с использованием ортогонального проектирования на плоскость визуализации.

Приложение

Ссылка на видео: : https://youtu.be/irFx5sxIQfg

Исходный код:

beizer_functions.py

```python
import scipy.special


def bernstein_poly(i, n, t):
    return scipy.special.comb(n, i) * t ** i * (1 - t) ** (n - i)
```

lab2.py

```python
import tkinter as tk
from tkinter import messagebox
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from bezier_functions import bernstein_poly


class Point(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.coordinates = [0, 0]
        self.confirm_button = tk.Button
        self.number = ''
        self.point_frame = tk.Frame(self, width=100, height=50)
        self.point_frame.pack()
        new_point_label = tk.Label(self.point_frame, text="Point " +
self.number)
        new_point_label.pack()

        x_label = tk.Label(self.point_frame, text="X:")
        x_label.pack()
        self.x_entry = tk.Entry(self.point_frame)
        self.x_entry.pack()

        y_label = tk.Label(self.point_frame, text="Y:")
        y_label.pack()
        self.y_entry = tk.Entry(self.point_frame)
        self.y_entry.pack()

        self.confirm_button = tk.Button(self.point_frame, text="Confirm
coordinates", command=self.confirm_coordinates)
        self.confirm_button.pack(pady=20)

    def confirm_coordinates(self):
        self.coordinates[0] = int(self.x_entry.get())
        self.coordinates[1] = int(self.y_entry.get())
        self.x_entry.config(bg="lightgrey", fg="black")
        self.y_entry.config(bg="lightgrey", fg="black")


class BezierCurveApp(tk.Tk):
    def __init__(self):
        super().__init__()

        self.canvas = None
        self.title("Bezier Curve App")
        self.geometry("1020x800")
```

```
        self.figs = []
        self.points = []
        self.point_coordinates = []

        self.menu_frame = tk.Frame(self, width=200, height=600)
        self.menu_frame.pack(side="left", fill="y")

        self.add_point_button = tk.Button(self.menu_frame, text="Add
Point", command=self.add_point)
        self.add_point_button.pack(pady=20)

        self.draw_curve_button = tk.Button(self, text="Draw Bezier Curve",
command=self.draw_curve)
        self.draw_curve_button.pack(side="bottom")

    def make_coordinates(self):
        points_correct = []
        for i in range(len(self.points)):
            a = [self.points[i].coordinates[0],
self.points[i].coordinates[1]]
            points_correct.append(a)
        print(points_correct)
        return points_correct

    def add_point(self):
        new_point = Point(self.menu_frame)
        self.points.append(new_point)
        new_point.pack(side='top', fill="both", expand=False)

    def draw_curve(self):
        if len(self.points) < 2:
            messagebox.showerror("Error", "At least 2 points are required
to draw the Bezier curve")
            return

        points = np.array(self.make_coordinates())

        t = np.linspace(0, 1, 1000)
        curve_x = np.zeros_like(t)
        curve_y = np.zeros_like(t)

        for i in range(len(points)):
            curve_x += points[i][0] * bernstein_poly(i, len(points) - 1, t)
            curve_y += points[i][1] * bernstein_poly(i, len(points) - 1, t)

        fig, ax = plt.subplots()
        ax.plot(curve_x, curve_y, label='Bezier Curve', color='blue')
        ax.scatter(points[:, 0], points[:, 1], color='red')  # Отображение
точек управления
        for i, (x, y) in enumerate(points):
            ax.text(x, y, f'P{i} ({x},{y})', fontsize=12, ha='right')

        # Adding dashed lines connecting the control points
        for i in range(len(points) - 1):
            ax.plot([points[i][0], points[i + 1][0]], [points[i][1],
points[i + 1][1]], 'k--')

        ax.legend()
        if self.canvas == None:
            self.canvas = FigureCanvasTkAgg(fig, master=self)
        else:
```

```
            self.canvas.get_tk_widget().destroy()
            self.canvas = FigureCanvasTkAgg(fig, master=self)
        self.canvas.draw()
        self.canvas.get_tk_widget().pack(side="right", fill="both",
expand=True)


if __name__ == "__main__":
    app = BezierCurveApp()
    app.mainloop()
```