TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

A Project Report

On

Interactive Visualization and Learning of Bézier Curves Using Python

Submitted By:

Sayal Shrestha (HCE081BEI041)

Submitted To:

Department of Electronics and Computer Engineering

Himalaya College of Engineering

Chyasal,Lalitpur

Feb-14,2025

ACKNOWLEDGEMENT

I express my sincere gratitude to all those who have supported and guided me throughout the process of conducting this report on Interactive Visualization and Learning of Bézier Curves Using Python. This endeavor would not have been possible without their valuable contributions and assistance.

This project helped me strengthen my understanding of mathematical visualization, curve modeling, and interactive graphics using Python. I would also like to thank my instructor, Sushant Pandey for the guidance and conceptual clarity provided throughout the course. His insightful feedback and continued encouragement have been invaluable in refining the scope and focus of this project.

This work would not have been possible without the collective efforts of these individuals and organizations. Although any shortcomings in this report are solely my responsibility, their contributions have significantly enriched its content.

Thank you.

Sayal Shrestha (HCE081BEI041)

ABSTRACT

This project presents an interactive Computer Graphics application developed using Python, NumPy, Matplotlib, and Tkinter. The system visualizes fundamental mathematical and graphical concepts including complex number transformations, 2D Bézier curves using De Casteljau's algorithm, and 3D Bézier curves using Bernstein polynomials.

The project is divided into modular components controlled by a central UI launcher. It integrates real-time interaction, parameter sliders, draggable control points, quiz-based learning, and parametric equation generation.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

Computer Graphics (CG) focuses on creating, manipulating, and displaying visual content using computers. Modern CG systems rely on mathematical models of geometry, lighting, and imaging to convert a 3D scene into a 2D image. This report presents a mini project titled **Interactive Visualization and Learning of Bezier Curves Using Python**, which demonstrates core CG concepts such as interactive visualization of complex number transformations and Bezier curves using Python..

1.1 Background Introduction

Bézier curves are defined using Bernstein polynomials and are widely used in CAD, animation, and UI rendering. A Bézier curve of degree n is defined as:

$$P(t) = \sum_{k=0}^{n} \binom{n}{k} t^k (1-t)^{n-k} P_k$$

Complex numbers provide a natural way to represent 2D rotation and scaling:

$$z = re^{i\theta}$$

1.2 Motivation

Many students understand formulas theoretically but struggle to visualize their geometric meaning. The motivation behind this project is:

- To convert mathematical exercise questions into interactive visual demonstrations
- To build geometric intuition
- To combine learning and interactivity
- To create a modular and expandable CG system

1.3 Objectives

The main objectives of the project are listed below:

- Implement interactive visualization of complex number transformations
- Implement 2D Bézier curves using De Casteljau algorithm
- Implement 3D Bézier curves using Bernstein basis
- Display parametric equations of Bézier curves

- Integrate quiz-based conceptual validation
- Create a modular UI system to launch different modules

1.4 Scope

This project covers fundamental CG concepts used in 2D rendering:

- 2D complex plane transformations
- Parametric curve generation
- Interactive sliders and control point dragging
- Real-time graphical updates
- 3D curve plotting
- Educational MCQ-based reinforcement

It does not include a full exam oriented mcqs, procedural derivation, or GPU based rendering. Those are suggested as future enhancements.

2. LITERATURE REVIEW

2.1 Complex Numbers

Multiplying a complex number by $e^{i\theta}$ rotates it by angle θ. Multiplying by r scales its magnitude. This provides an elegant method for 2D geometric transformations without explicitly using matrices.

2.2 Bézier curves

Bézier curves were introduced by Pierre Bézier for automobile body design. They are based on Bernstein polynomials and provide smooth parametric curves controlled by a set of control points. Two major approaches exist for computing Bézier curves:

Bernstein Polynomial Formulation

De Casteljau Recursive Interpolation Algorithm De Casteljau's algorithm is numerically stable and geometrically intuitive. A Bézier curve of degree n is defined using Bernstein basis polynomials:

$$B_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}$$

The general Bézier curve equation is:

$$P(t) = \sum_{k=0}^{n} B_{k,n}(t) P_k$$

3. METHODOLOGY

3.1 System Overview

The project consists of four main modules:

- **UI Module (ui.py)** – Tkinter-based launcher that scans and executes project files.
- **Complex Transformation Module (pre.py)** – Visualizes rotation and scaling in the complex plane.
- **2D Bézier Module (bezier.py)** – Implements De Casteljau algorithm with draggable control points.
- **3D Bézier Solver (bezierSolve.py)** – Computes Bernstein polynomial form and generates parametric equations.

3.2 Mathematical Foundation

3.2.1 De Casteljau Algorithm

Recursive interpolation formula:

$$P_i^k = (1-t)P_i^{k-1} + tP_{i+1}^{k-1}$$

Repeated until a single point remains.

3.2.2 Bernstein Polynomial Basis

$$B_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}$$

$$P(t) = \sum_{k=0}^{n} B_{k,n}(t) P_k$$

3.2.3 Complex Rotation

If $z = x + iy$, then multiplying by i rotates the vector by 90 degrees counterclockwise.

3.3 Algorithms Used

3.3.1 De Casteljau Algorithm

Implemented in `bezier.py` for 2D curve construction.

3.3.2 Bernstein Polynomial Evaluation

Implemented in `bezierSolve.py` for 3D parametric generation.

3.3.3 Interactive Event Handling

Mouse drag events update control points dynamically.

3.3.4 Slider-Based Parameter Control

Parameter t varies from 0 to 1 to trace the curve.

3.4 Implementation Snippet

```
def bezier_parametric_eq(points):
    n = len(points) - 1
    x_eq = " + ".join([f"{comb(n,k)}*{points[k,0]}*t**{k}*(1-t)**{n-k}" for k in rang
    y_eq = " + ".join([f"{comb(n,k)}*{points[k,1]}*t**{k}*(1-t)**{n-k}" for k in rang
    z_eq = " + ".join([f"{comb(n,k)}*{points[k,2]}*t**{k}*(1-t)**{n-k}" for k in rang
    return x_eq, y_eq, z_eq


x_eq, y_eq, z_eq = bezier_parametric_eq(control_points)

print("\nParametric equations (Bernstein form):")
print(f"x(t) = {x_eq}")
print(f"y(t) = {y_eq}")
print(f"z(t) = {z_eq}")
```

3.5 Sample Table

Control Points	Degree	Curve Type
2	1	Linear
3	2	Quadratic
4	3	Cubic
5	4	Quartic

Table 3.1: Relationship between control points and curve degree

3.6 Screenshots / Figures

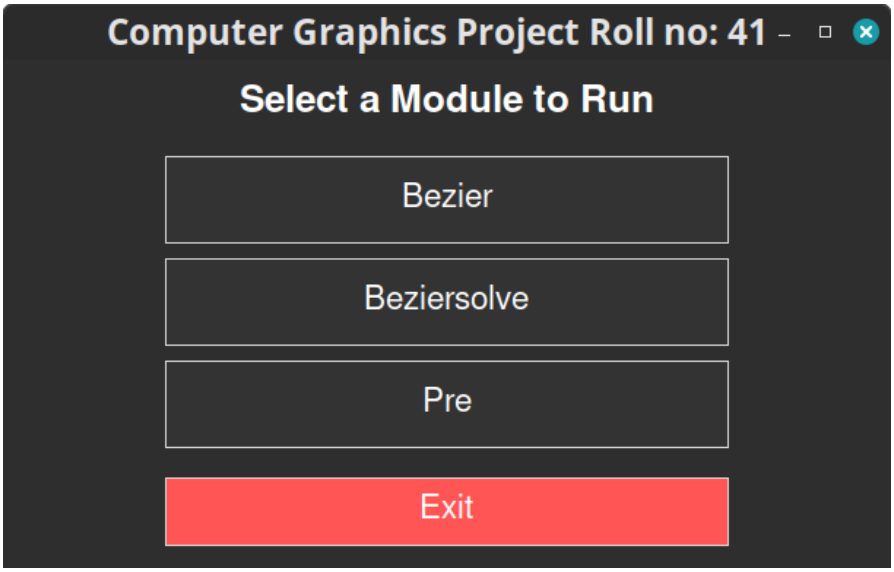

Figure 3.1: Main UI Launcher (ui.py)

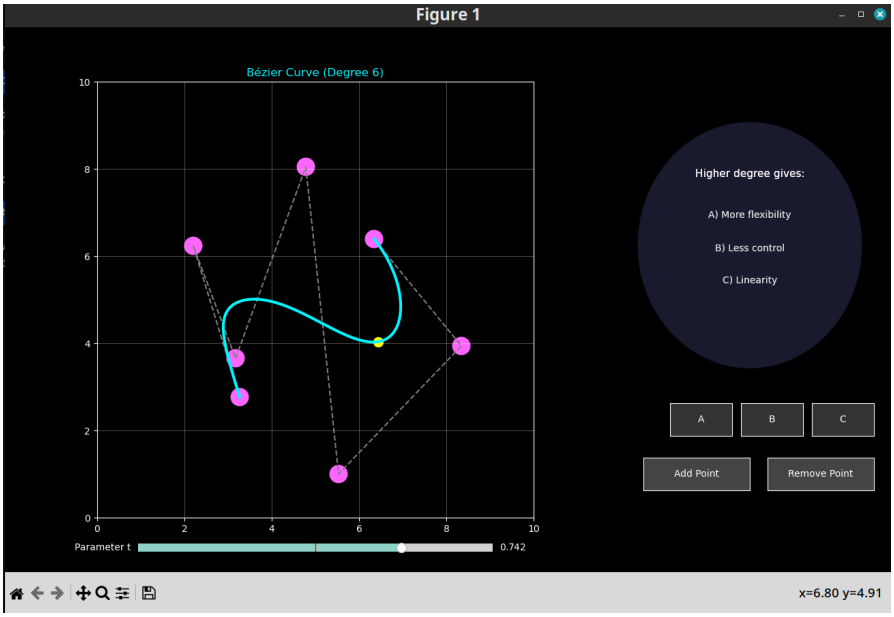

Figure 3.2: 2D Bézier Curve with Control Points

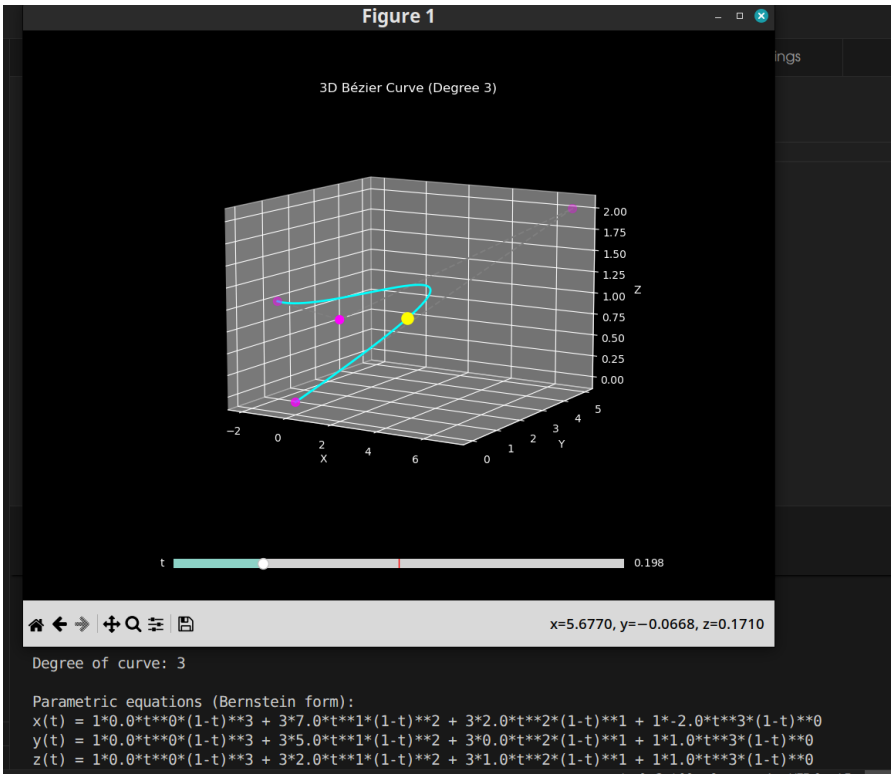

Figure 3.3: 3D Bézier Curve Visualization

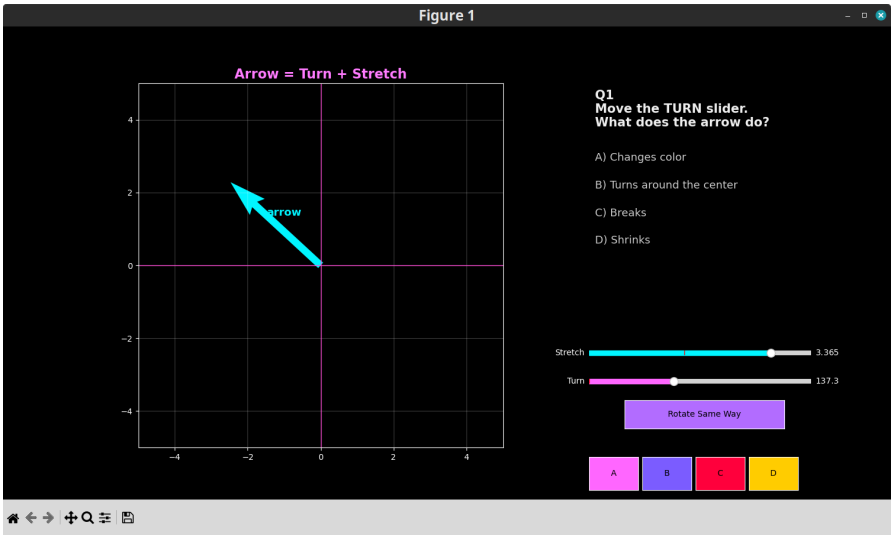

Figure 3.4: Interactive Prerequisite with Mcq

4. RESULT AND ANALYSIS

4.1 Correctness of Rendering Output

The Bézier curves pass through the first and last control points as expected. The curve remains inside the convex hull of the control polygon, validating correctness. The 3D visualization correctly follows the parametric equations generated.

4.2 Transformation Behavior

In `pre.py`:

- Stretch slider changes magnitude only.
- Turn slider changes angle only.
- Four successive multiplications by i return to original orientation.

In `bezier.py`:

- Increasing control points increases curve flexibility.
- Parameter t smoothly traces the curve.

4.3 Performance Discussion

The system runs smoothly for up to 8 control points in real time. Slider updates and drag events are responsive due to efficient NumPy computation.

4.4 Limitations

- Derivations aren't focused as it's intuition central.
- No export of curve data.
- 3D input is terminal-based instead of GUI.
- MCQs lacking in exam oriented mastery.

5. CONCLUSION AND FUTURE ENHANCEMENT

5.1 Conclusion

This project successfully integrates mathematical theory and visualization. It demonstrates how complex transformations and Bézier curves can be implemented interactively using Python. The modular structure allows easy extension and experimentation.

5.2 Future Enhancements

- Implement Bézier surface patches.
- Add GUI-based 3D control point input.
- Export curve as image or animation.
- Make the questions and interaction more geared towards course oriented mastery

A. APPENDICES

A.1 UI Controls

- Mouse Click: Add control point
- Key R: Reset
- Key Q: Exit

Bibliography

[1] E. W. Weisstein, “Complex transformation,” 2026. Accessed: 2026-02-21.

[2] G. Farin, *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide*. San Francisco, CA, USA: Morgan Kaufmann, 5th ed., 2002.

[3] S. Pandey, “Class notes on bezier curves,” 2026. Unpublished lecture notes, Himalaya College of Engineering.

[1] [2] [3]