UNIVERSIDADE FEDERAL DE RORAIMA CENTRO DE CIÊNCIA E TECNOLOGIA BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DCC703 – COMPUTAÇÃO GRÁFICA (2024.2)

Data de entrega: 19/03/2025

DISCENTES:
FELIPE RUBENS DE SOUSA BORGES (2020020120)

COMPUTAÇÃO GRÁFICA

Relatório Curvas de Bézier

Boa Vista-RR
2025.2

Relatório de Curvas de Bézier

Relatório apresentado para o projeto de construção das Curvas de Bézier por meio do algoritmo dos algoritmos: Equação Paramétrica e de Casteljau, da disciplina de Computação Gráfica, ofertada pelo curso de Ciência da Computação da Universidade Federal de Roraima.

Prof. Luciano Ferreira

Boa Vista-RR

2025.2

Sumário

Resumo

Este relatório descreve a implementação de uma interface gráfica para visualização das Curvas de Bézier, sua construção e os resultados obtidos utilizando as bibliotecas Matplotlib, Numpy e Scipy, em Python. O relatório aborda dois tipos de implementação da Curva de Bézier, os algoritmos: Equação Paramétrica que utiliza os polinômios de Bernstein para calcular a curva diretamente. Já o algoritmo de Casteljau utiliza uma abordagem recursiva para subdividir a curva até que ela seja precisa. O programa tem como objetivo realizar a amostragem da Curva através de diferentes abordagens na implementação, mas resultados idênticos.

Algoritmos e Fundamentação

Algoritmo Equação Paramétrica

A Curva de Bézier é definida por uma fórmula matemática que combina dois pontos de controle com os Polinômios de Beirstein:

$$B(t) = \sum_{i=0}^{n} B_{i,n}(t) \cdot P_i, \quad t \in [0,1]$$

onde B_{i,n}(t) = {n i} t^i (1-t)^{n-i} é o polinômio de Bernstein, P_i são os pontos de controle, e (n) é o grau da curva (número de pontos de controle menos 1).

Assim calculando a curva diretamente para vários valores de (t), gerando pontos ao longo da trajetória.

Algoritmo de Casteljau

- É um método geométrico e recursivo que subdivide os pontos de controle iterativamente até alcançar uma aproximação suave da curva.
- Utiliza interpolação linear repetida entre os pontos de controle para encontrar pontos na curva, controlando a precisão com um critério de distância.
- O processo pode ser visto como uma construção passo a passo, onde cada subdivisão refina a curva até que ela seja menor que um limiar.

Os dois métodos produzem a mesma curva de Bézier, mas diferem na abordagem: o paramétrico é analítico e contínuo, enquanto o Casteljau é iterativo e adaptável.

No código:

- ❖ Definição da Função **bernstein**:
  - Calcula o polinômio de Bernstein B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}.

- Parâmetros:
  - n: Grau da curva (número de pontos de controle - 1).
  - i: Índice do ponto de controle atual.
  - t: Valor do parâmetro (array de 0 a 1).
- Retorna o valor do polinômio para cada (t), usado na equação paramétrica.

❖ Definição da Função **subdivisao_curva**:

- Implementa a subdivisão geométrica dos pontos de controle, baseada no algoritmo de Casteljau.
- Para quatro pontos P_0, P_1, P_2, P_3:
  - Calcula pontos intermediários pela média:
    - $M_{01} = (P_0 + P_1)/2$
    - $M_{12} = (P_1 + P_2)/2$
    - $M_{23} = (P_2 + P_3)/2$
  - Calcula pontos de segunda ordem:
    - $M_{012} = (M_{01} + M_{12})/2$
    - $M_{123} = (M_{12} + M_{23})/2$
  - Calcula o ponto médio da curva:
    - $M_{0123} = (M_{012} + M_{123})/2$
- Retorna duas novas curvas (primeira e segunda metades) e o ponto médio M_{0123}

.

❖ Definição da Função **parametrica**:

- Calcula a curva de Bézier usando a **equação paramétrica**.
- Passos:
  - Define n = len(points) - 1(grau da curva).
  - Gera 100 valores de (t) entre 0 e 1 com np.linspace.
  - Para cada ponto de controle P_i, calcula $B_{i,n}(t) \cdot P_i$ e soma os resultados.
- Retorna um array de pontos (x, y) que formam a curva.

❖ Definição da Função **casteljau**:

- Implementa o algoritmo de Casteljau com subdivisão recursiva.
- Parâmetros:
  - pontos: Lista de pontos de controle.
  - u: Tolerância para decidir se a curva é "reta" o suficiente.
- Função interna **subdivisão_recursiva**:
  - Calcula a distância entre o primeiro e o último ponto de controle.
  - Se a distância for maior que (u):
    - Subdivide a curva em duas metades com subdivisao_curva.
    - Chama a si mesma recursivamente para cada metade.

- Se menor ou igual a (u):
    - Adiciona o último ponto à lista de pontos da curva.
- Retorna os pontos calculados como um array NumPy.

❖ Definição dos **Pontos de Controle**:

- **pontos_controle**: Array NumPy com quatro pontos ([[0, 0], [1, 3], [3, 3], [4, 0]]), definindo uma curva cúbica de Bézier.
- **pontos_controle_2**: Cópia idêntica para o método Casteljau.

❖ Geração das Curvas:

- **curva_parametrica** = bezier_curva(pontos_controle): Calcula a curva com a equação paramétrica.
- **curva_casteljau** = casteljau(pontos_controle_2): Calcula a curva com o algoritmo de Casteljau.

Implementação

```python
# calcula os polinômios de Bernstein
def bernstein(n, i, t):
    return scipy.special.comb(n, i) * (t ** i) * ((1 - t) ** (n - i))
```

```python
# subdivide os pontos de controle igual ao slide
def subdivisao_curva(points):
    M01 = (points[0] + points[1]) / 2
    M12 = (points[1] + points[2]) / 2
    M23 = (points[2] + points[3]) / 2

    M012 = (M01 + M12) / 2
    M123 = (M12 + M23) / 2

    M0123 = (M012 + M123) / 2

    primeira_metade = [points[0], M01, M012, M0123]
    segunda_metade = [M0123, M123, M23, points[3]]

    return primeira_metade, segunda_metade, M0123
```

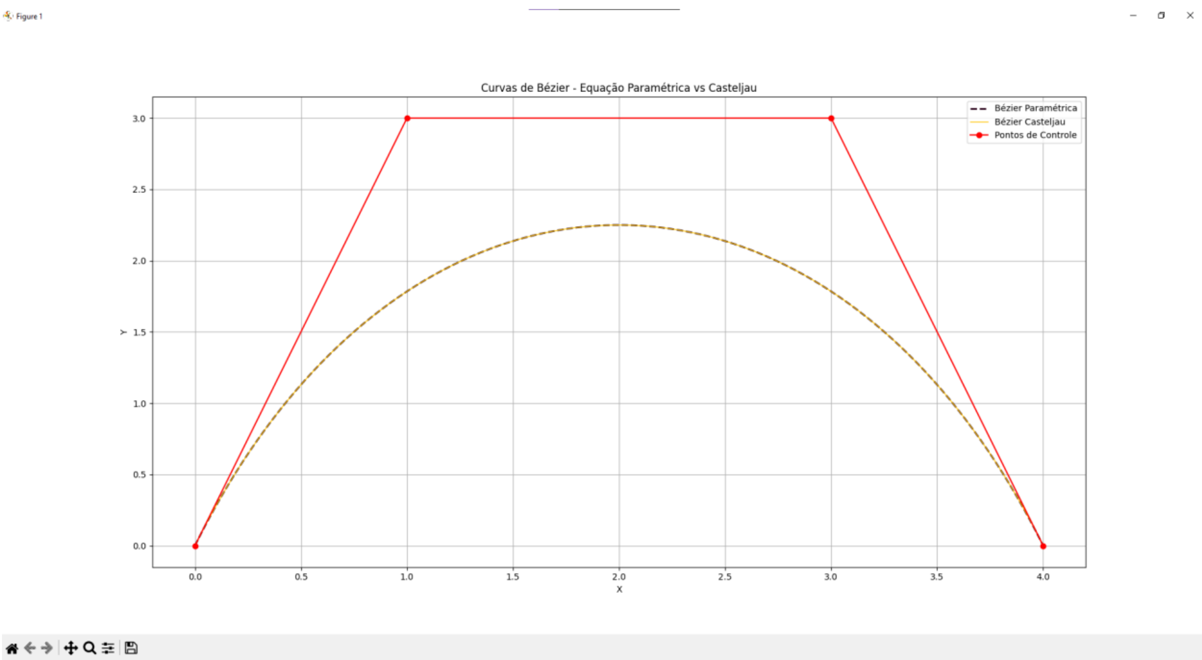
```python
# calcula a curva de Bézier pela equação paramétrica
def bezier_curva(points, num_points=100):
    n = len(points) - 1
    t_values = np.linspace(0, 1, num_points)
    curva = np.zeros((num_points, 2))

    for i in range(n + 1):
        curva += np.outer(bernstein(n, i, t_values), points[i])

    return curva
```

```python
# calcula a curva de Bézier pelo algoritmo de Casteljau
def casteljau(points, u=0.005):
    pontos_curva = []

    def recursive_subdivision(ctrl_pontos):
        p0, p3 = ctrl_pontos[0], ctrl_pontos[-1]
        dist = np.linalg.norm(p3 - p0)

        if dist > u:
            primeira_metade, segunda_metade, midpoint = subdivisao_curva(ctrl_pontos)

            recursive_subdivision(primeira_metade)
            recursive_subdivision(segunda_metade)
        else:
            pontos_curva.append(ctrl_pontos[-1])

    recursive_subdivision(points)
    return np.array(pontos_curva)
```

```python
# define pontos de controle
pontos_controle = np.array([[0, 0], [1, 3], [3, 3], [4, 0]])
pontos_controle_2 = np.array([[0, 0], [1, 3], [3, 3], [4, 0]])

# gera das curvas
curva_parametrica = bezier_curva(pontos_controle)
curva_casteljau = casteljau(pontos_controle_2)
```

```python
plt.figure(figsize=(8, 6))

# curva da equação paramétrica
plt.plot(curva_parametrica[:, 0], curva_parametrica[:, 1], label="Bézier Paramétrica", linestyle="--", color="#2C041C", linewidth=2)

# curva do algoritmo de Casteljau
plt.plot(curva_casteljau[:, 0], curva_casteljau[:, 1], label="Bézier Casteljau", linestyle="-", color="#FFC300", linewidth=1)

# exibe os pontos de controle
plt.plot(pontos_controle[:, 0], pontos_controle[:, 1], "ro-", label="Pontos de Controle")

plt.legend()
plt.title("Curvas de Bézier - Equação Paramétrica vs Casteljau")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid()
plt.show()
```

Resultado



Como resultado, foi obtida uma representação dentro do esperado, onde a Equação Paramétrica é calculada usando a equação matemática direta a partir dos polinômios de Bernstein gerando uma curva suave pois o número de pontos é definido, é uniforme e rápido e de simples implementação. Já o Casteljau é calculado recursivamente, não possui o número de pontos definidos pois eles são subdivididos, tornando-se um algoritmo adaptativo servindo para curvas mais complexas sendo mais complexo.

Em termos de comparação, para curvas simples e número de pontos moderado, o método paramétrico é mais eficiente, enquanto para curvas complexas ou alta precisão, Casteljau pode ser mais eficiente ao evitar pontos desnecessários, mas a recursão aumenta o custo computacional.

Conclusão

O código apresentado implementa os algoritmos Equação Paramétrica e Casteljau, sendo ambos eficientes para gerar curvas de Bézier. No contexto do código, ambos demonstram a versatilidade das curvas de Bézier, com a visualização destacando sua equivalência prática apesar das abordagens distintas. Conclui-se que o Paramétrico é ideal para renderização rápida e uniforme (ex.: gráficos 2D pré-calculados) e o Casteljau é melhor para animações ou sistemas adaptativos, onde a curva pode ser refinada dinamicamente.