PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

DCC703 - COMPUTAÇÃO GRÁFICA

RELATÓRIO DO PROJETO DE BÉZIER

ALUNOS:
Marcos Vinícius Tenacol Coêlho - 2021000759

Março de 2025
Boa Vista/Roraima

Resumo

A implementação apresentada tem como objetivo desenhar curvas de Bézier.

O programa permite ao usuário adicionar pontos de controle clicando na tela e visualizar a curva gerada por dois métodos diferentes: a equação paramétrica e o algoritmo de De Casteljau.

O programa define duas abordagens principais para gerar a curva de Bézier:

Equação Paramétrica: Utiliza o polinômio de Bernstein para calcular os pontos da curva.

Algoritmo de De Casteljau: Um método recursivo que divide os segmentos até convergir para um ponto na curva.

Estrutura do Código

Equação paramétrica:

polinomio_bernstein: Calcula o polinômio de Bernstein para um determinado grau e parâmetro t.

curva_bezier_parametrica: Gera pontos da curva de Bézier usando a equação paramétrica.

Castejau:

algoritmo_casteljau: Calcular a curva de Bézier usando o algoritmo de Casteljau, dividindo o pontos, pegando sempre o ponto médio para gerar a curva e chamando a função recursivamente novamente.

Como funciona a equação paramétrica?

A equação paramétrica de Bézier utiliza a combinação linear dos pontos de controle, ponderados pelo polinômio de Bernstein.
A implementação do código gera a curva variando em um intervalo e somando os produtos dos pontos de controle com os coeficientes de Bernstein.

Para um conjunto de pontos de controle, a curva é definida por:

$$C(t) = \sum_{i=0}^{n} B_{i,\,n}(t).P_i$$

Como funciona o algoritmo de De Casteljau?

O algoritmo de De Casteljau é um método recursivo que calcula um ponto da curva de Bézier dividindo iterativamente os segmentos formados pelos pontos de controle. Para um dado valor de , os novos pontos intermediários são calculados.

O processo continua até restar apenas um ponto, que pertence à curva.

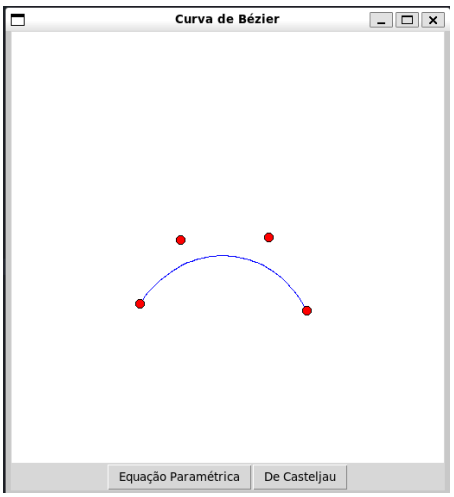
Implementação e testes

Equação paramétrica:

```
def polinomio_bernstein(n, i, t):
    return (np.math.comb(n, i)) * (t ** i) * ((1 - t) ** (n - i))
def curva_bezier_parametrica(pontos_controle, num_pontos=50):
    n = len(pontos_controle) - 1
    t_values = np.linspace(0, 1, num_pontos)
    curva = np.zeros((num_pontos, 2))

    for i in range(n + 1):
         curva += np.array(pontos_controle[i]) * polinomio_bernstein(n,
i, t_values)[:, np.newaxis]
    return curva
```

Curva de bézier utilizando equação paramétrica, com os pontos nas coordenadas: (1.49, 3.16), (1.96, 2.42), (2.98, 2.39), (3.42, 3.24):



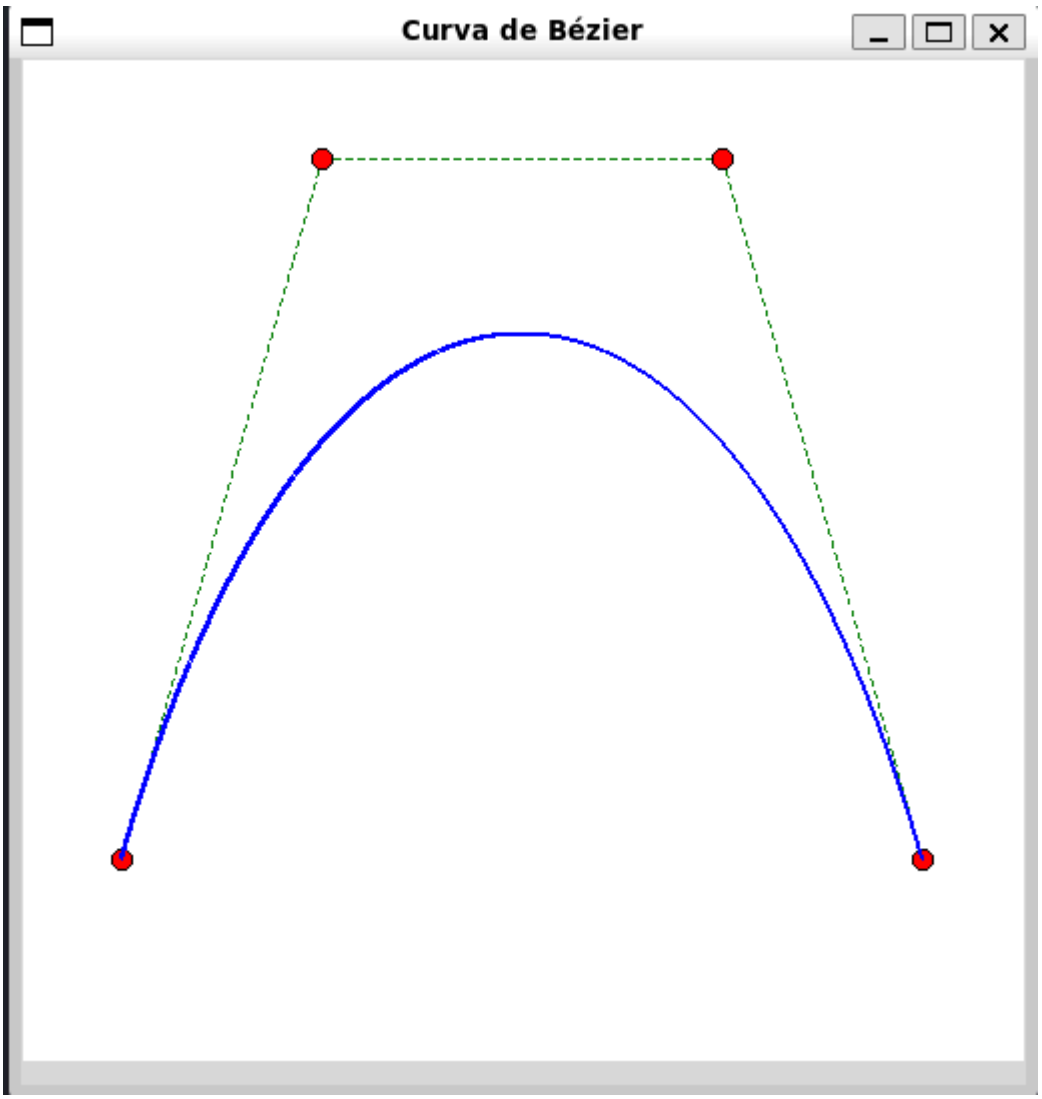
Castejau:

```
def   algoritmo_casteljau(x1,   y1,   x2,   y2,   x3,   y3,   x4,   y4,
conjunto_pixels):
    xm01, ym01 = (x1 + x2) / 2, (y1 + y2) / 2
    xm12, ym12 = (x2 + x3) / 2, (y2 + y3) / 2
    xm23, ym23 = (x3 + x4) / 2, (y3 + y4) / 2
    xm012, ym012 = (xm01 + xm12) / 2, (ym01 + ym12) / 2
    xm123, ym123 = (xm12 + xm23) / 2, (ym12 + ym23) / 2
    xm0123, ym0123 = (xm012 + xm123) / 2, (ym012 + ym123) / 2

    conjunto_pixels.add((round(xm0123), round(ym0123)))

    if math.sqrt((xm012 - xm123) ** 2 + (ym012 - ym123) ** 2) < 0.01:
        return
    else:
          algoritmo_casteljau(x1, y1, xm01, ym01, xm012, ym012, xm0123,
ym0123, conjunto_pixels)
          algoritmo_casteljau(xm0123, ym0123, xm123, ym123, xm23, ym23,
x4, y4, conjunto_pixels)
```

Curva de bézier utilizando casteljau, com os pontos nas coordenadas:
(50, 400), (150, 50), (350, 50), (450, 400):


Curva de Bézier