UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DCC703 - COMPUTAÇÃO GRÁFICA

RAFAEL NÓBREGA DE LIMA

ALGORITMOS DE CURVAS DE BÉZIER - RELATÓRIO

BOA VISTA - RR
2025

1. BASE DOS ALGORITMOS E ORGANIZAÇÃO

O ambiente gráfico foi desenvolvido com a biblioteca “pygame”, permitindo que o usuário defina pontos de controle clicando na tela e visualize a construção da curva em tempo real. A tela foi configurada com resolução 800x600 pixels, garantindo uma boa experiência visual.

```
# Configuração do pygame
WIDTH, HEIGHT = 800, 600
BACKGROUND_COLOR = (30, 30, 30)
CURVE_COLOR = (255, 0, 0)
CONTROL_COLOR = (0, 0, 255)
MIDPOINT_COLOR = (0, 255, 0)
LINE_COLOR = (100, 100, 100)
FPS = 60
```

Nessa configuração já foram definidas as configurações de cores, telas e algumas configurações da animação.

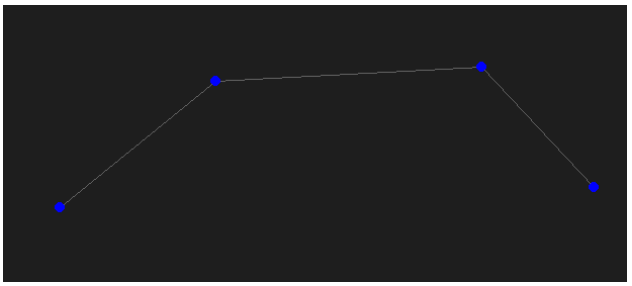
- **Eventos implementados:**
  - Botão esquerdo do mouse: Adiciona um ponto de controle.
  - Botão direito do mouse: Inicia a animação da curva.
  - Tecla espaço: Realiza a subdivisão da curva (apenas no algoritmo de Casteljau).

1.1. DESENHO DOS PONTOS DE CONTROLE

- A função **draw_control_polygon( )** mostra a influência de cada ponto na curva Bézier.

```
def draw_control_polygon(screen, control_points):
    """Desenha os pontos de controle e suas conexões."""
    for i in range(len(control_points) - 1):
        pygame.draw.line(screen, LINE_COLOR, control_points[i], control_points[i + 1], 1)
    for point in control_points:
        pygame.draw.circle(screen, CONTROL_COLOR, (int(point[0]), int(point[1])), 5)
```

- É desenhado linhas conectando os pontos de controle, criando o polígono de controle.
- É desenhado pequenos círculos nos pontos de controle para identificá-los visualmente.

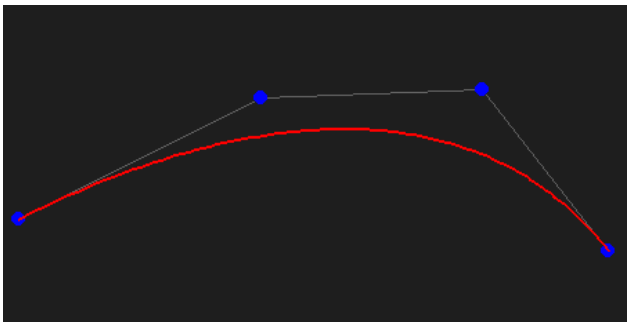


1.2. DESENHO DAS CURVAS

- A função **draw_curve( )** permite que a curva seja desenhada progressivamente, criando uma animação visualmente suave.

```python
def draw_curve(screen, curve_points, progress):
    """Desenha a curva de Bézier na tela com animação suave."""
    num_points = int(len(curve_points) * progress)
    for i in range(num_points - 1):
        pygame.draw.line(screen, CURVE_COLOR, curve_points[i], curve_points[i + 1], 2)
```

- **progress** controla o quanto da curva já foi desenhada, variando de 0 a 1.
- **num_points** determina a quantidade de pontos da curva que serão renderizados com base no progresso.
- O loop percorre os pontos já calculados e desenha segmentos de reta entre eles usando **pygame.draw.line( ).**



2. CURVA DE BÉZIER

As curvas de Bézier são curvas polinomiais que são expressas como a interpolação linear entre pontos de controle que representam formas suaves e complexas. Este relatório apresenta a implementação e análise comparativa dos algoritmos Paramétrico e Casteljau para a construção dessas curvas.

2.1. ALGORITMO PARAMÉTRICO DE BÉZIER

O algoritmo paramétrico de Bézier usa os polinômios de Bernstein para calcular os pontos da curva de forma direta.

- **CÓDIGO IMPLEMENTADO**

O algoritmo Paramétrico foi dividido em três funções: **bernstein_poly( )**, **bezier_generalized( )** e **bezier_curve_parametric( ).**

No **bernstein_poly**( ) é onde se é calculado o coeficiente de Bernstein para um determinado índice i e grau n da curva. Essa função é fundamental para calcular a contribuição de cada ponto de controle para a curva em um determinado valor de t. Os polinômios de Bernstein são definidos como:

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

onde:

- $\binom{n}{i}$ (coeficiente binomial) é calculado usando o comb(n,i).
- t^i e $(1-t)^{n-i}$ controlam a influência dos pontos de controle na curva.

Já na função **bezier_generalized( )** tem como objetivo calcular um ponto específico da curva de Bézier, baseado nos pontos de controle e no valor de t. onde:

- t: Valor do parâmetro no intervalo [0,1].
- control_points: Lista de pontos de controle da curva.

É por último a função **bezier_curve_parametric( )**, nela calcula-se múltiplos pontos da curva de Bézier para diferentes valores de t, criando a curva completa.

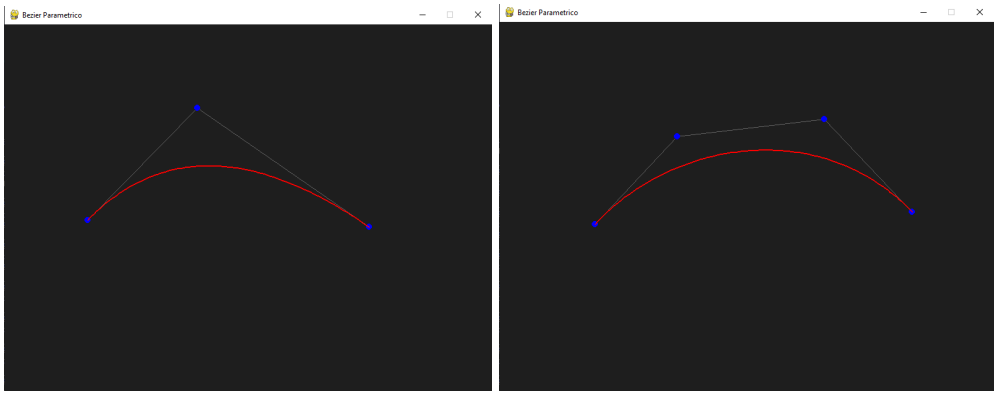
```python
def bernstein_poly(i, n, t):
    """Calcula o polinômio de Bernstein para a curva de Bézier"""
    from scipy.special import comb
    return comb(n, i) * (t ** i) * ((1 - t) ** (n - i))

def bezier_generalized(t, control_points):
    """Calcula um ponto na curva de Bézier para um valor de t."""
    n = len(control_points) - 1
    point = np.zeros(2)
    for i in range(n + 1):
        point += bernstein_poly(i, n, t) * control_points[i]
    return point

def bezier_curve_parametric(control_points, num_points=100):
    """Gera os pontos da curva de Bézier usando a equação paramétrica."""
    t_values = np.linspace(0, 1, num_points)
    return [bezier_generalized(t, control_points) for t in t_values]
```

- **PASSOS**
  - **Definir os pontos de controle**: O usuário insere os pontos iniciais.
  - **Aplicação dos polinômios de Bernstein**: Para cada valor de **t**, os pesos dos pontos de controle são calculados.
  - **Cálculo da curva**: A soma dos pontos de controle multiplicados pelos polinômios de Bernstein gera os pontos da curva.
  - **Iteração para valores de t**: O processo é repetido para valores de **t** de 0 a 1, formando a curva completa.

- **RESULTADOS**



- **VANTAGENS E DESVANTAGENS**

  O algoritmo paramétrico de Bézier permite calcular diretamente os pontos da curva, tornando-se uma abordagem eficiente e rápida para renderização. Sua principal vantagem está na capacidade de gerar curvas suavemente com um menor custo computacional.

No entanto, apresenta algumas limitações. Como cada ponto de controle influencia globalmente a curva, pequenas alterações nos pontos podem modificar significativamente sua forma. Além disso, para curvas de grau elevado, podem surgir problemas numéricos, como oscilações inesperadas e perda de precisão devido ao cálculo dos coeficientes binomiais nos polinômios de Bernstein.

2.2. ALGORITMO DE CASTELJAU

O Algoritmo de Casteljau é um método recursivo para calcular curvas de Bézier. Ele baseia-se na interpolação linear sucessiva entre os pontos de controle até obter um único ponto na curva.

- **CÓDIGO IMPLEMENTADO**
  - **ponto_medio(p1, p2):**
    - **Calcula o ponto médio** entre dois pontos. Essencial para o algoritmo de Casteljau.

```python
def ponto_medio(p1, p2):
    """Calcula o ponto médio entre dois pontos."""
    return (p1 + p2) / 2
```

  - **casteljau(P0, P1, P2, P3, t, pontos=[ ])**
    - Ela implementa **recursivamente** a subdivisão da curva de Bézier.
    - Calcula os pontos médios entre cada par de pontos.
    - Continua dividindo até t ser suficientemente pequeno ($t \leq 0.005$).
    - Chama a si mesma duas vezes, para subdividir a curva em duas partes.
    - Armazena os pontos intermediários na lista **pontos**, garantindo a sequência correta.

```python
def casteljau(P0, P1, P2, P3, t, pontos=[]):
    """Recursão do algoritmo de Casteljau baseado na subdivisão da curva."""
    M01 = ponto_medio(P0, P1)
    M12 = ponto_medio(P1, P2)
    M23 = ponto_medio(P2, P3)

    M012 = ponto_medio(M01, M12)
    M123 = ponto_medio(M12, M23)

    M0123 = ponto_medio(M012, M123)

    if t > 0.005:
        t /= 2
        casteljau(P0, M01, M012, M0123, t, pontos)
        pontos.append(M0123)
        casteljau(M0123, M123, M23, P3, t, pontos)
    else:
        pontos.append(P0)
        pontos.append(P3)

    return pontos
```

  - **draw_curve_by_depth(screen, pontos_controle, profundidade, progress)**
    - Realiza a renderização da curva na tela

- Chama-se **Casteljau** para calcular os pontos da curva com base na profundidade.
- Limita a profundidade entre 0 e 1.
- Usa um fator de progressão (progress) para animar o desenho da curva.

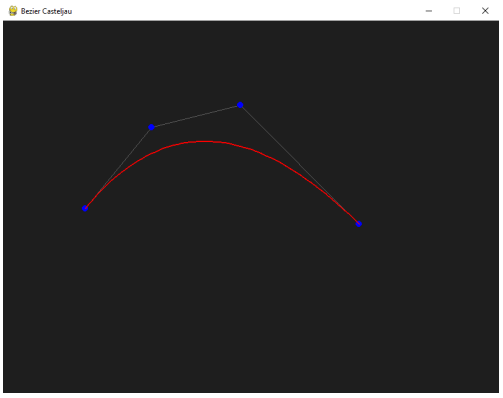
```python
def draw_curve_by_depth(screen, pontos_controle, profundidade, progress):
    """Desenha a curva de Bézier com animação baseada na profundidade."""
    profundidade = max(0, min(profundidade, 1))  # Limita profundidade entre 0 e 1
    if len(pontos_controle) == 4:
        pontos_curva = casteljau(pontos_controle[0], pontos_controle[1], pontos_controle[2], pontos_controle[3], profundidade, [])
        num_points = int(len(pontos_curva) * progress)
        if num_points > 1:
            pygame.draw.lines(screen, CURVE_COLOR, False, pontos_curva[:num_points], 2)
```

- **draw_control_polygon(screen, control_points)**
  - Desenha linhas conectando os pontos de controle (cada ponto é representado como um círculo azul)

```python
def draw_control_polygon(screen, control_points):
    """Desenha os pontos de controle e suas conexões."""
    for i in range(len(control_points) - 1):
        pygame.draw.line(screen, LINE_COLOR, control_points[i], control_points[i + 1], 1)
    for point in control_points:
        pygame.draw.circle(screen, CONTROL_COLOR, (int(point[0]), int(point[1])), 5)
```

- **PASSOS**
  - **Definir os pontos de controle**: O usuário insere os pontos que influenciarão a forma da curva.
  - **Início da subdivisão da curva:** Após a escolha dos 4 pontos de controle, com o botão direito do mouse ativa-se a subdivisão, aumentando a profundidade da subdivisão progressivamente. A subdivisão usa a função **casteljau**, que é recursiva.
  - **Interpolação linear**: Para cada chamada recursiva da função **casteljau**, são calculados os pontos médios entre os pontos de controle.
  - **Continuação da interpolação**: O processo continua aplicando a interpolação aos novos pontos gerados, reduzindo progressivamente a quantidade de pontos e refinando a
  - **Obtenção do ponto da curva**: Quando sobra apenas um ponto, ele é registrado como parte da curva.
  - **Renderização progressiva da curva**: A função draw_curve_by_depth desenha os pontos gerados pela recursão.O valor de progress controla a animação da curva:
    - Inicialmente a curva começa curta.
    - Gradualmente os segmentos da curva são revelados.
  - **Reset e interação:** A Tecla Espaço reseta a subdivisão e permite começar uma nova curva. Novos cliques no botão esquerdo adicionam novos pontos e redefine a curva.

- **RESULTADO**



- **VANTAGENS E DESVANTAGENS**

  O algoritmo de De Casteljau destaca-se por sua estabilidade numérica e sua capacidade de subdividir a curva de forma eficiente, permitindo um controle local mais preciso. Essa subdivisão facilita a manipulação e o refinamento da curva, tornando-o ideal para aplicações que exigem ajustes finos e suavização.

  No entanto, essa abordagem apresenta algumas limitações. O algoritmo é mais lento para cálculos diretos, pois exige múltiplas interpolações sucessivas. Além disso, requer um número significativamente maior de cálculos intermediários, aumentando o custo computacional em comparação com o método paramétrico. Por conta disso, não é a opção mais eficiente para renderização rápida, especialmente em aplicações que exigem desempenho otimizado.

3. CONCLUSÃO

Ambos os algoritmos são eficientes para a construção de curvas de Bézier, cada um possuindo vantagens em diferentes cenários. O método de Casteljau é ideal para refinamento local e subdivisão, enquanto o método paramétrico é mais adequado para renderização direta e eficiência computacional.