Universidade Federal de Roraima
Centro de Ciências e Tecnologia
Departamento de Ciência da Computação
Disciplina de Computação Gráfica
Professor: Dr. Luciano Ferreira Silva
Aluno: Felipe Derkian de Sousa Freitas – 1201424418

Curvas de Bézier com método da equação Paramétrica e Casteljau

Boa Vista, 28 de Novembro de 2018

Sumário

Curvas de Bézier por meio da Equação Paramétrica

Intiução

"Como criar uma curva que começa em um ponto, termina em outro, e a sua forma depende de pontos de controle, que irão "puxar" ou "afastar" a curva de suas proximidades?".

Uma resposta plausível para esta questão seria que temos de construir uma função que estabeleça o "peso" que cada ponto de controle em cada momento ao longo da curva possa usar esses pesos para fazer a curva suavizar de acordo com o peso ou força que esse ponto de controle exerce sobre a linha.

Polinômio de Bernstein

Esse é o polinômio de Bernstein que foi usado por Bézier como base para construção da curva. Onde (n i) é o polinômio de Newton, e o parâmetro x varia de 0 a 1.

$$B_i^n(x) = \binom{n}{i} x^i (1-x)^{n-i}$$

Figura 1: Polinômio de Bernstein

Exemplo:

$$B_0^3(x) = \binom{3}{0} x^0 (1-x)^{3-0} = (1-x)^3$$

Figura 2: Exemplo usando a fórmula

Todo polinômio do 3 grau pode ser escrito dessa forma, que por sinal é a que iremos utilizar para desenvolver o algoritmo de grau 3. onde os C's são os pontos de controle que tem os pesos.

$$P(x) = c_0 B_0^3(x) + c_1 B_1^3(x) + c_2 B_2^3(x) + c_3 B_3^3(x)$$

Figura 3: função para Bernstein de grau 3

Para a curva de Bézier precisamos aplicar o polinômio para cada ponto e para cada coordenada x, y e z do ponto, p(x, y, z). Nesse esperimento usaremos apenas os pares ordenados p(x,y) por ser duas dimensões que analisaremos.

Algoritmo de Bézier para Cúbicas desenvolvido em C/C++.

A função desenvolvida recebe como parâmetro uma matriz m de dimensões MxM, e dois vetores de tamanho N. A matriz será a representação de nossa malha de pixel's e os vetores terão os valores dos pares ordenados de cada ponto de controle p(x, y).

```
void bezier (int m[M][M], int x[N], int y[N]) {
    double t, incremento=0.0005;
    for (t=0.0 ; t<1.0 ; t+=incremento) {
        double xt = pow(1-t,3) * x[0]+3*t*pow(1-t,2)*x[1]+3*pow(t,2)*(1-t)*x[2]+pow(t,3)*x[3];
        double yt = pow(1-t,3) * y[0]+3*t*pow(1-t,2)*y[1]+3*pow(t,2)*(1-t)*y[2]+pow(t,3)*y[3];
        int posX = (int) xt;
        int posY = (int) yt;
        m[posX][posY] = BORDA;
    }
}
```

Pseudo Código de Bézier

```
Funcao Bezier recebe pontos(x, y):
    para i de 0 até 1 com incremento k faça:
        x = função de Bernstein para grau 3 com parâmetro p.x
        y = função de Bernstein para grau 3 com parâmetro p.y
        pinta na posição m[x][y].
```

Experimentos com Bézier

8 → Representa o ponto de controle.
3 → Representa a linha formada pelo algoritmo.

Caso 1:

Pontos de controle: P0(10, 20), P1(30, 40), P2(40, 30) e P3(20, 10).

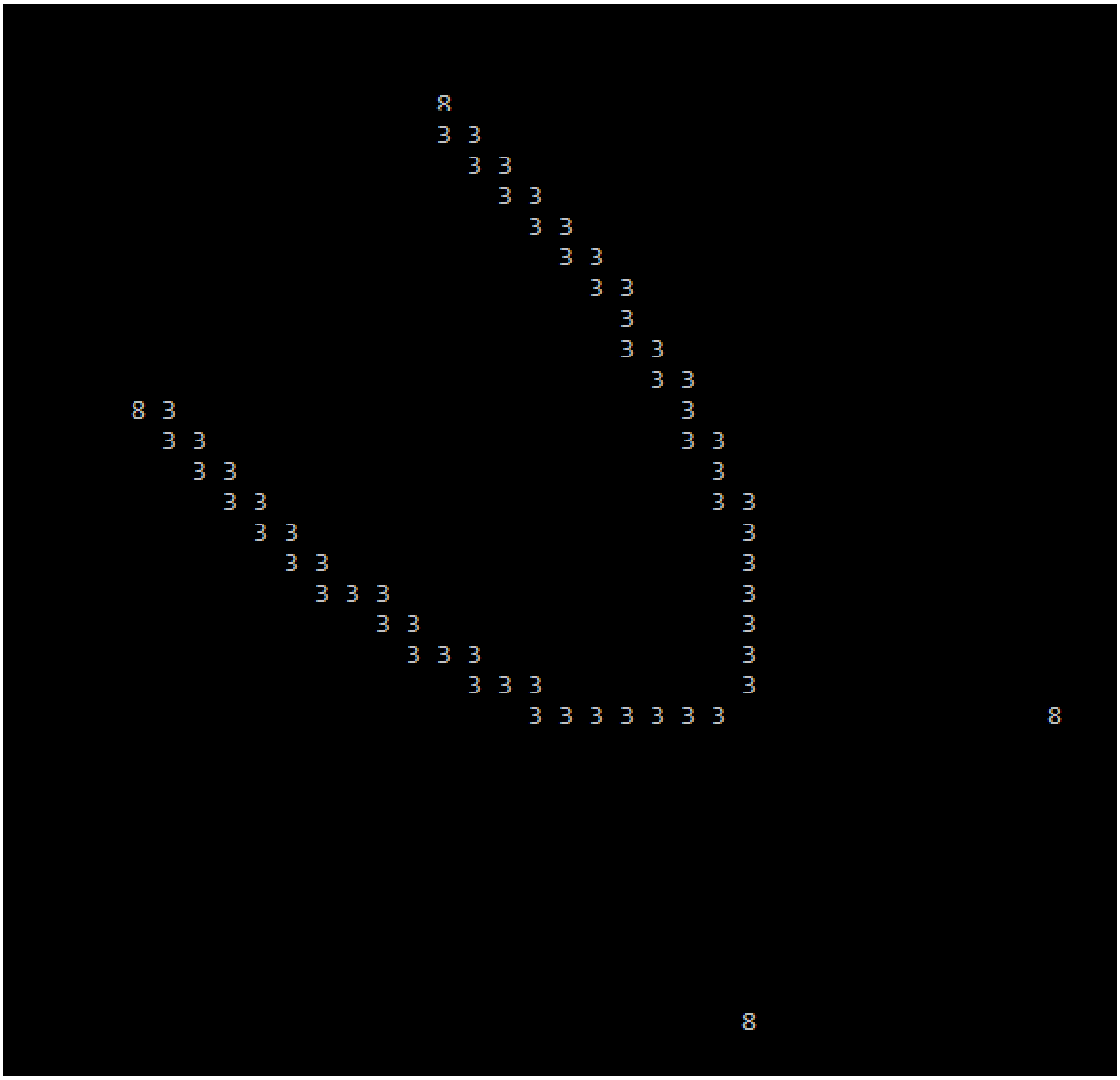
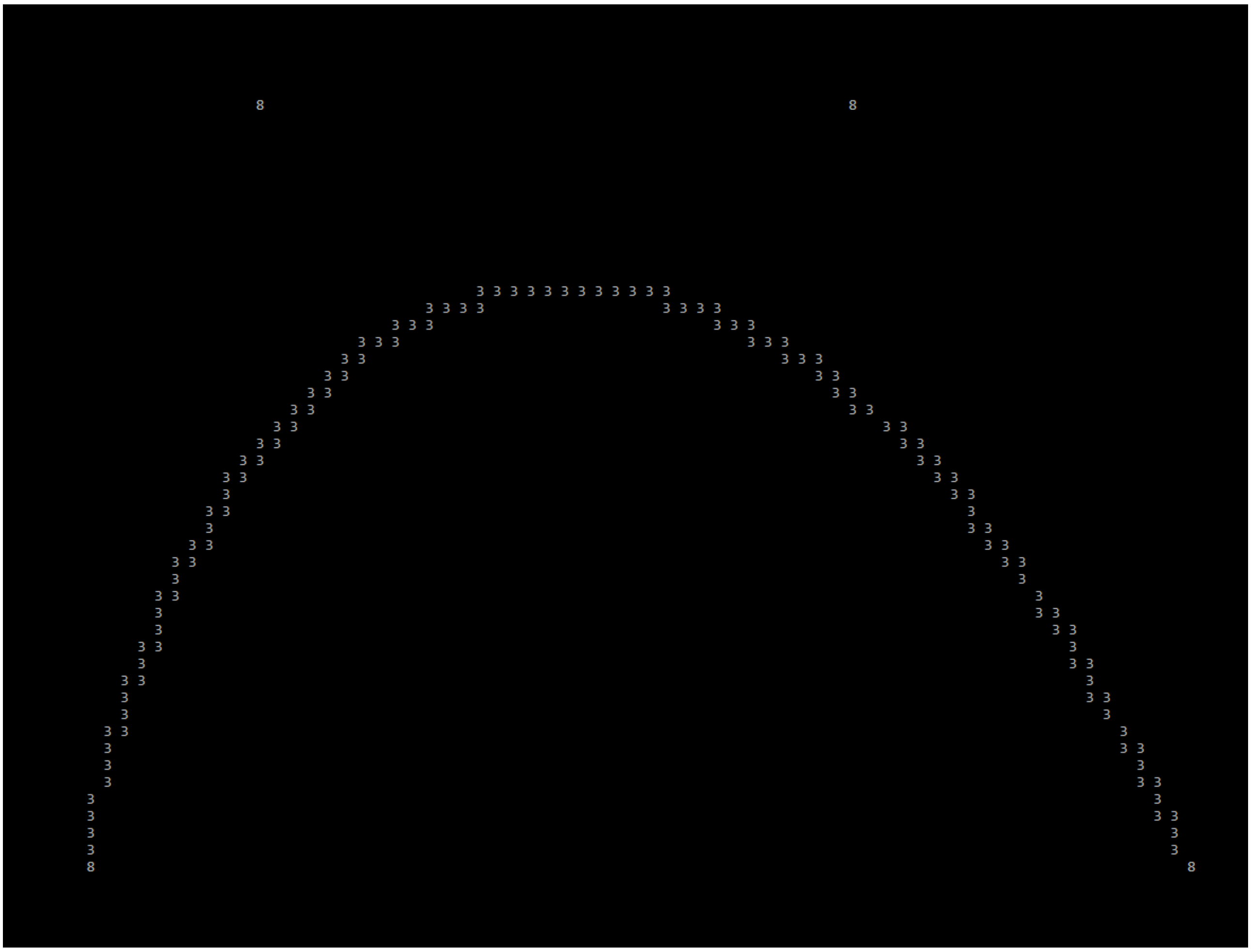


Figura 4: caso 1 - P0(10, 20), P1(30, 40), P2(40, 30) e P3(20, 10).

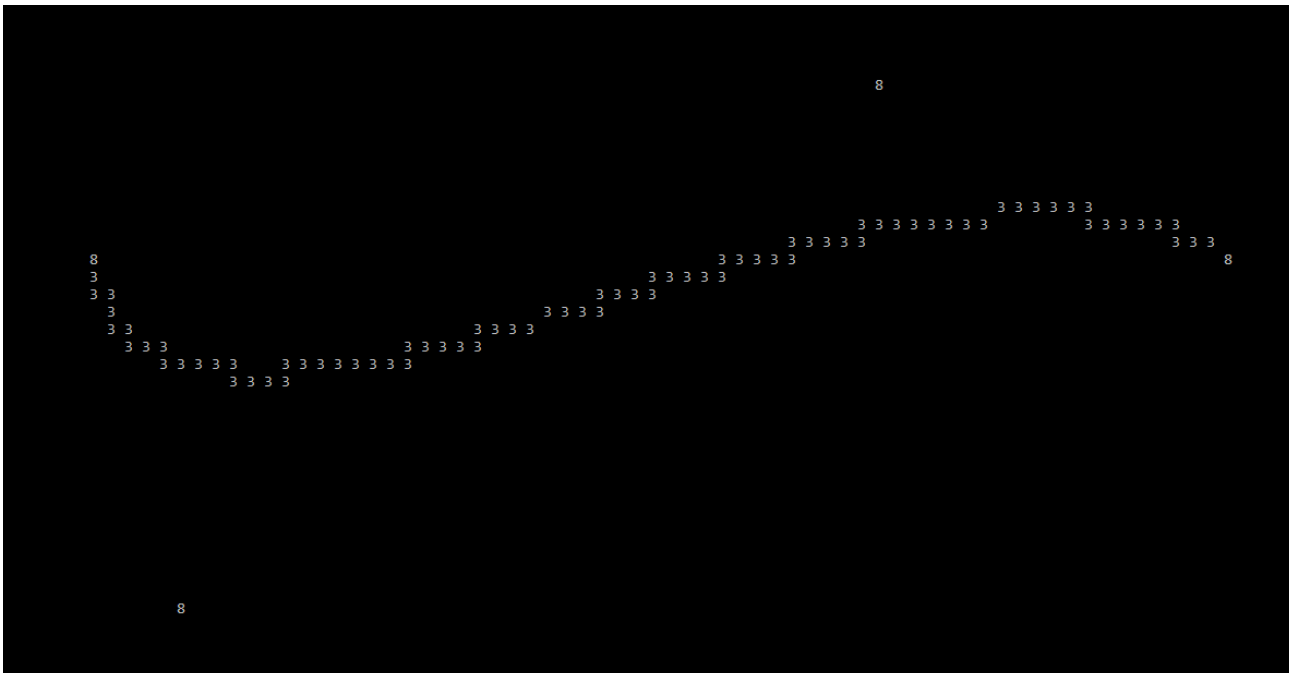
Caso 2:

Pontos de controle: P0(60, 5), P1(15,15), P2(15, 50) e P3(60,70).



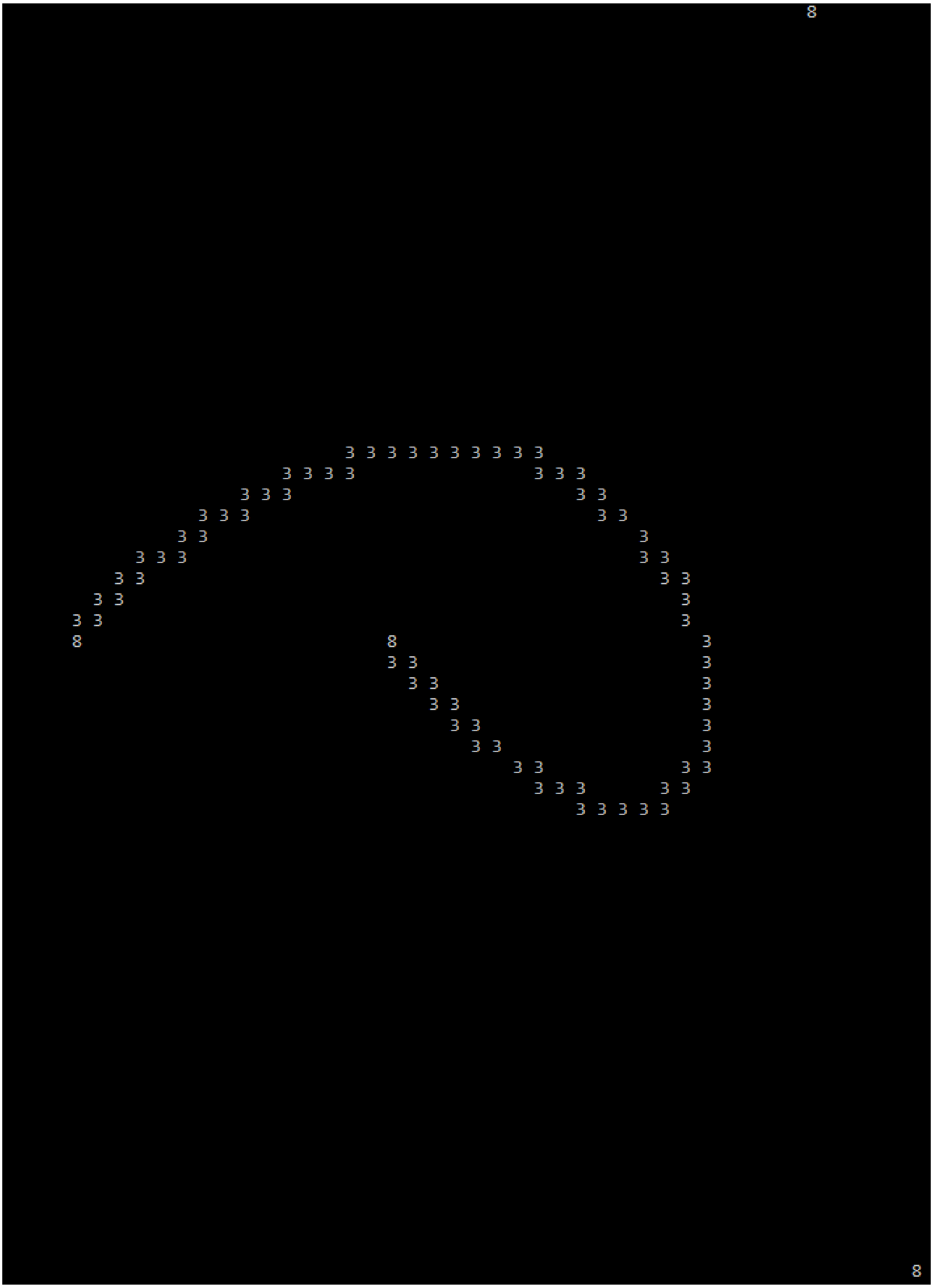
Caso 3:

Pontos de controle: P0(30, 5), P1(50, 10), P2(20, 50) e P3(30, 70).



Caso 4:

Pontos de controle: P0(40, 5), P1(10, 40), P2(70, 45) e P3(40, 20).



Algoritmo de Casteljau

Baseia-se na subdivisão recursiva dos segmentos medianos de reta criados a partir da união dos pontos de controle da curva. Ou seja, é um método recursivo que subdivide cada meio de um ponto de controle para o outro e liga uma divisão a outra e calcula a metade novamente e subdivide por esses novos pontos encontrados.

É um método mais eficiente computacionalmente do que o método paramétrico de Bézier.

O algoritmo de Casteljau usa a relação de recorrência do polinômio de Bernstein para grau 3 que é dada por:

$$B_i^n(x) = (1-x)B_i^{n-1}(x) + xB_{i-1}^{n-1}(x)$$

Figura 5: Relação de Recorrência do polinômio de Bernstein de grau 3

Algoritmo de Casteljau para Cúbicas desenvolvido em C/C++.

Esse algoritmo recebe uma matriz m de dimensões MxM, os pontos de controle B e o total de pontos de controle dado por b_len. Esse algoritmo funciona com um incremento de 0.001, passa os pontos para uma estrutura auxiliar, depois itera em dois for’s usando a equação de recorrência do polinômio de Bernstein. Depois marca as posições com a borda.

```
void Casteljau(int m[M][M], Point *B, size_t b_len) {
	float incremento = 0.001;
	for (float t=0; t<=1.0; t+=incremento) {
		Point q[b_len];
		size_t i, k;
		for (i=0; i<b_len; ++i)
			q[i].x = B[i].x;
			q[i].y = B[i].y;
		}
		for (k=1; k<b_len; ++k) {
			for (i=0; i<(b_len-k); ++i) {
				q[i].x = (1.0-t)*q[i].x + t*q[i+1].x;

				q[i].y = (1.0-t)*q[i].y + t*q[i+1].y;
			}
		}
```

```
        int posX = (int) q[0].x;
        int posY = (int) q[0].y;
        m[ posX ][ posY ] = BORDA;
    }
}
```

Pseudo Código de Casteljau

```
Função casteljau recebe os pontos de controle P e o total de pontos N:
    para i de 0 até 1 com incremento k faça:
        crie uma estrutura auxiliar para armazenar os pontos
        copie os pontos para a estrutura auxiliar
        para i de 0 até N-1 faça:
            para j de 0 até N-i faça:
                x = relação de recorrência com parametro P.x
                y = relação de recorrência com parametro P.y
        pintaPixel na posição P(x, y).
```

Experimentos com Bézier

8 → Representa o ponto de controle.
3 → Representa a linha formada pelo algoritmo.

Caso 1:

Pontos de controle: P0(10, 20), P1(30, 40), P2(40, 30) e P3(20, 10).

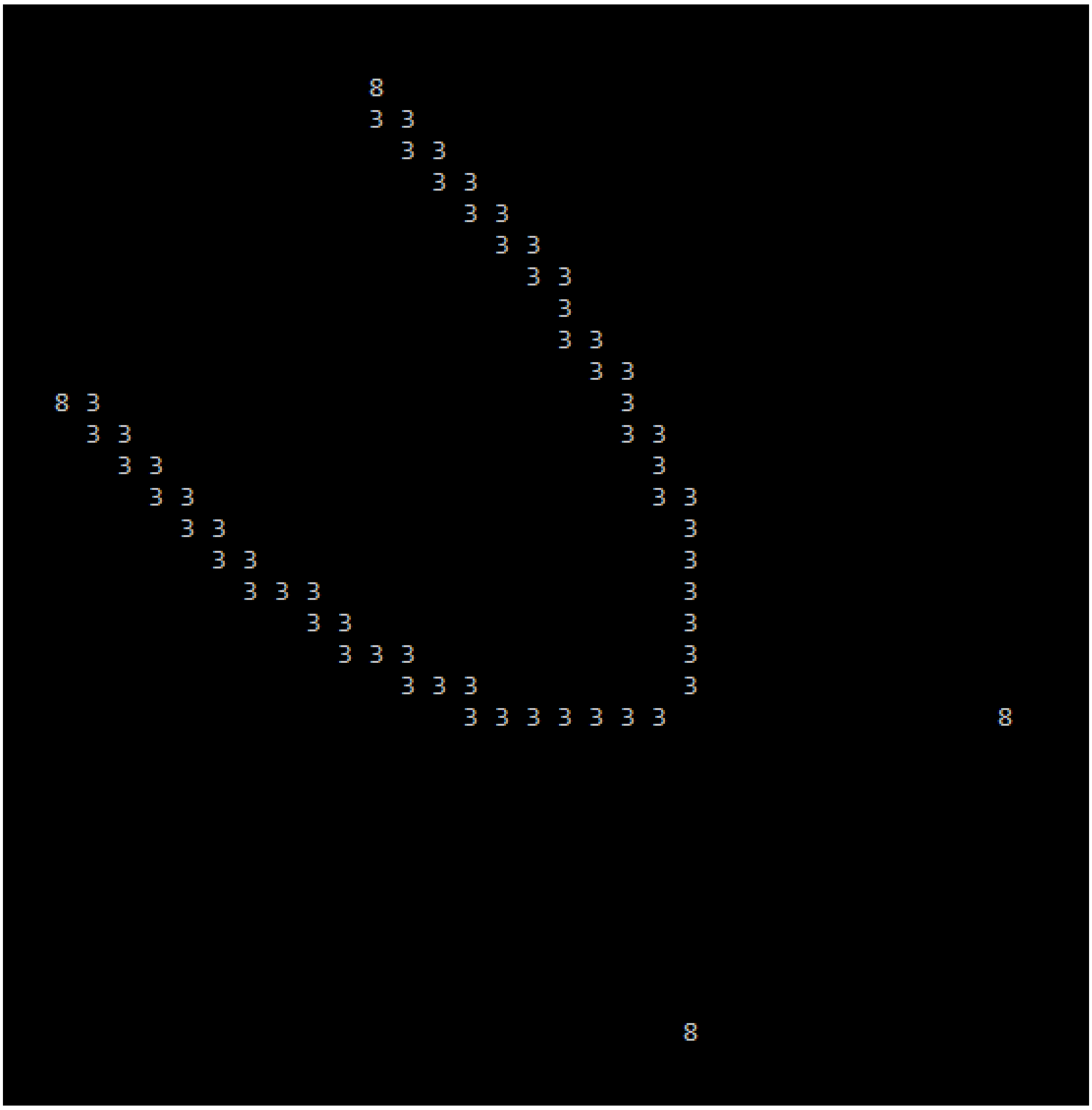


Figura 6: Pontos de controle: P0(10, 20), P1(30, 40), P2(40, 30) e P3(20, 10).

Caso 2:

Pontos de controle: P0(60, 5), P1(15,15), P2(15, 50) e P3(60,70).

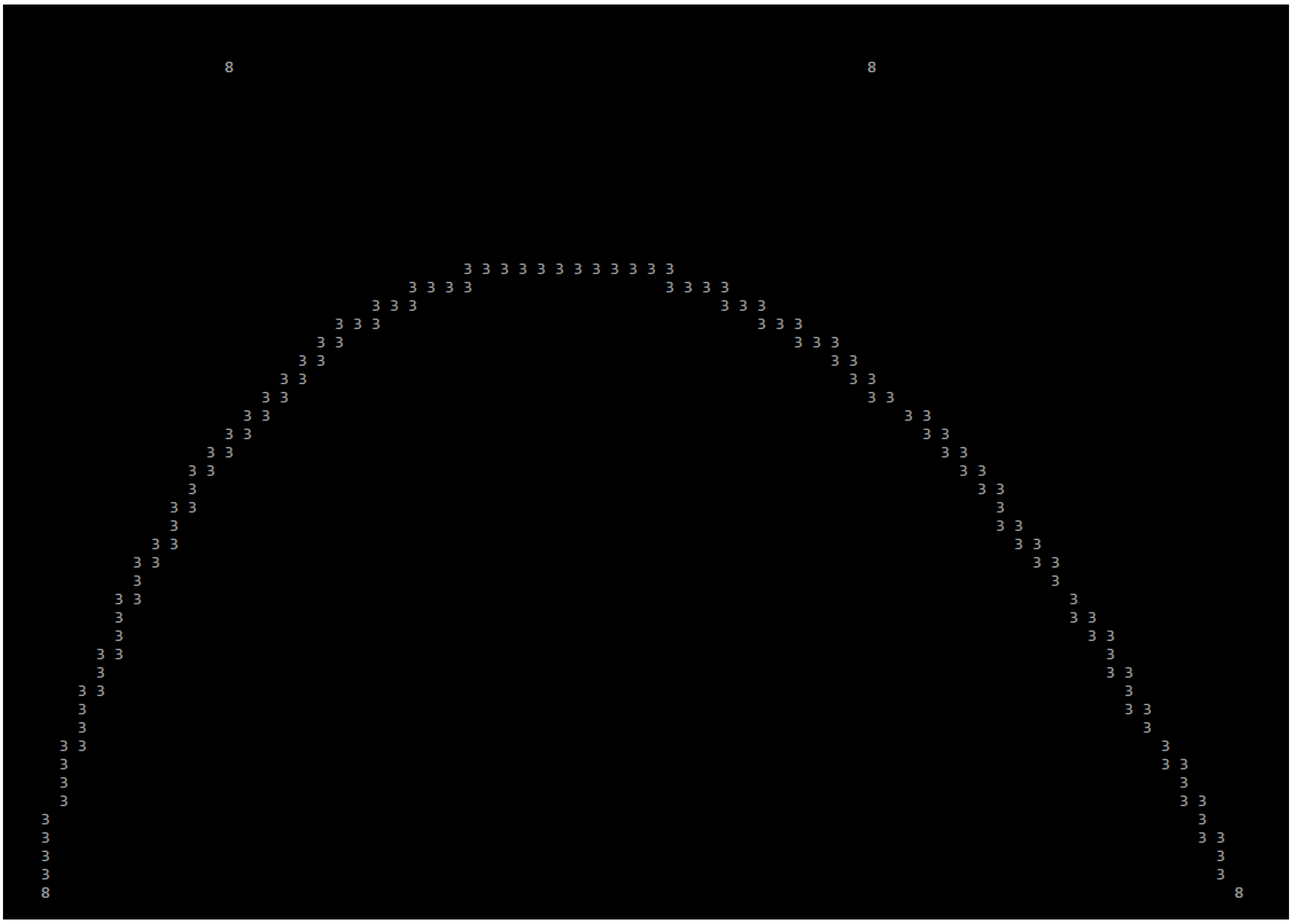


Figura 7: Pontos de controle: P0(60, 5), P1(15,15), P2(15, 50) e P3(60,70).

Caso 3:

Pontos de controle: P0(30, 5), P1(50, 10), P2(20, 50) e P3(30, 70).

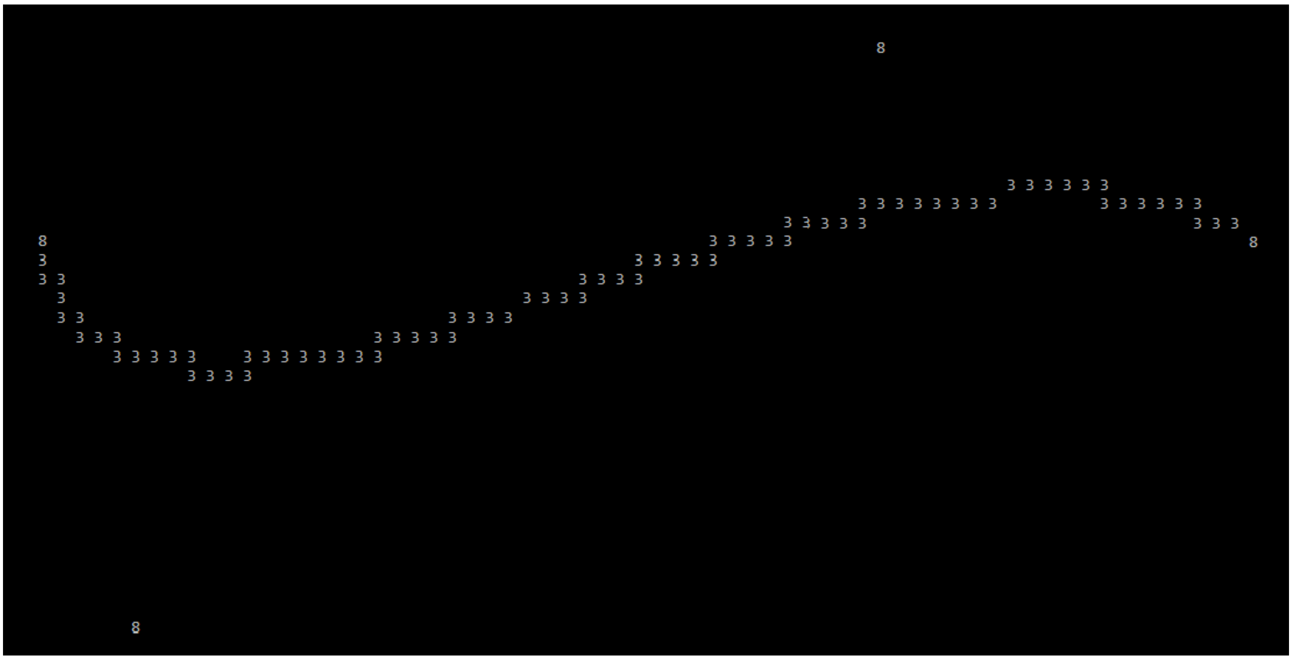


Figura 8: ontos de controle: P0(30, 5), P1(50, 10), P2(20, 50) e P3(30, 70).

Caso 4:

Pontos de controle: P0(40, 5), P1(10, 40), P2(70, 45) e P3(40, 20).

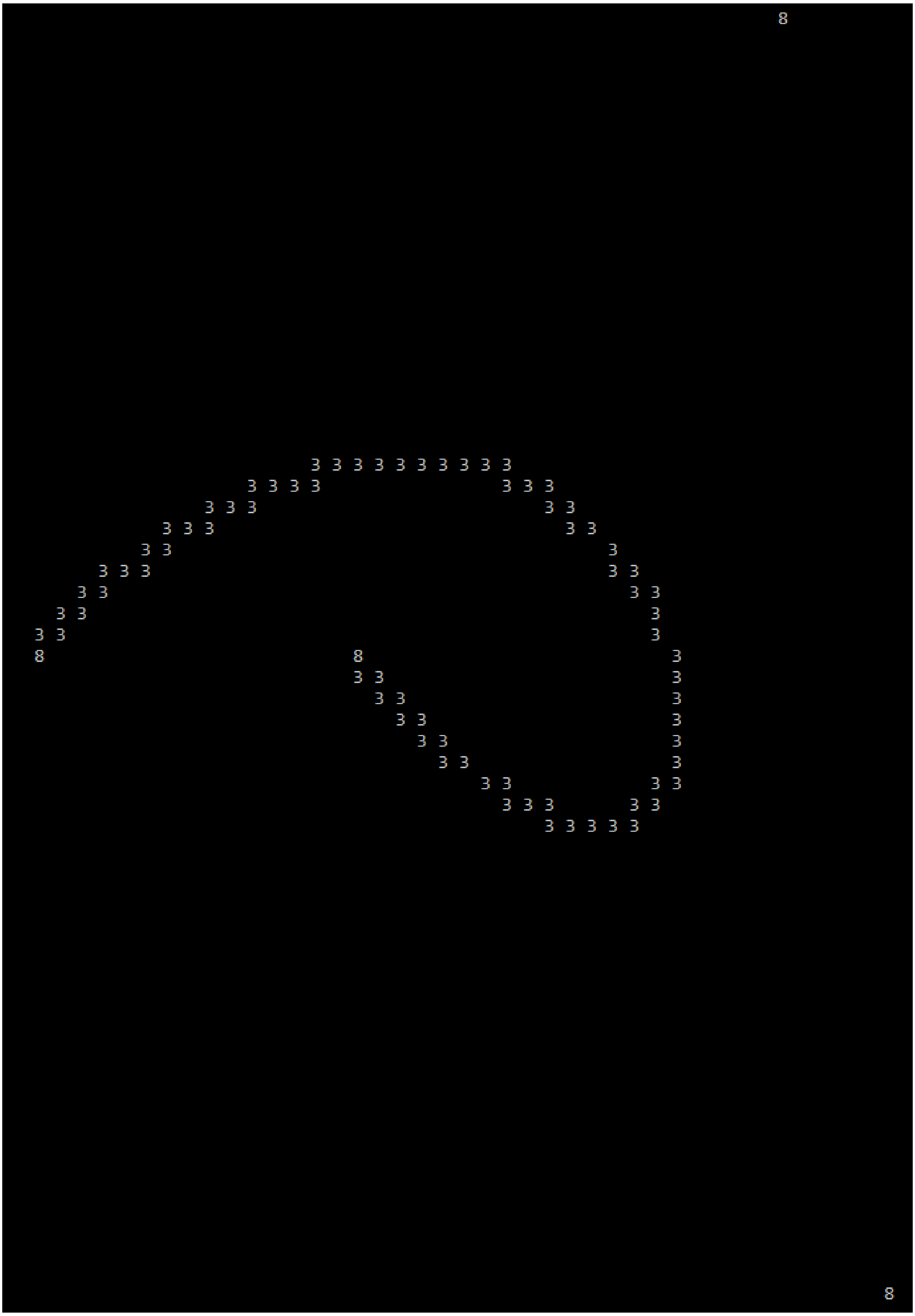


Figura 9: Pontos de controle: P0(40, 5), P1(10, 40), P2(70, 45) e P3(40, 20).

Conclusão

Os resultados mostraram que os dois algoritmos obtém os mesmos resultados para os casos de grau 3 onde temos 4 pontos de controle. A equação paramétrica tem a vantagem de poder ser utilizada com qualquer quantidade N de pontos de controle, enquanto a de Casteljau fica limitada para 4 pontos de controle. Além disso, a curva mais utilizada é a com grau 3 e tendo um bom desempenho computacional com algoritmo de Casteljau.

Referência

Polinómios de Bernstein. Disponível em https://pt.wikipedia.org/wiki/Polin%C3%B3mios_de_Bernstein. Acesso em 17/11/2018.

Curva de Bézier. Disponível em https://pt.wikipedia.org/wiki/Curva_de_B%C3%A9zier. Acesso em 18/11/2018.

Algoritmo de De Casteljau. Disponível em https://pt.wikipedia.org/wiki/Algoritmo_de_De_Casteljau. Acesso em 18/11/2018.

slide da Aula 12 - Curva de Bézier.