4 CURVAS PARAMÉTRICAS

Neste capítulo vamos abordar o desenho de curvas com naturalidade e eficiência. Para isso veremos um pouco da história do desenvolvimento de métodos cálculo e plotagem de curvas e depois os principais modelos de curvas paramétricas utilizados em Computação Gráfica. Por fim veremos uma técnica de desenho eficiente de curvas paramétricas baseada unicamente em somas e também como realizar o recorte de curvas. Todo o capítulo é voltado para 2D, mas os métodos são apresentados de forma genérica e podem ser diertamente aplicados ao caso 3D.

4.1. CURVAS

Para prover realismo e exatidão é necessária a representação de curvas de forma:

- fiel aos dados originais
- rápida e eficiente

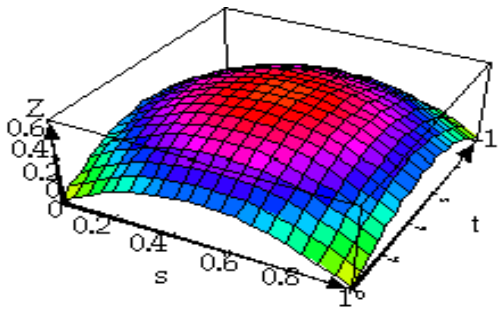
Para isso deve ser possível:

- determinar-se a qualquer momento posição no espaço de qualquer ponto em uma curva,
- uma representação compacta,
- realização de zoom eficiente

Isto é muito custoso através de representações geométricas tradicionais

Não podemos sempre representar todos os pontos que necessitamos. Para isso é necessária a utilização de Representações Paramétricas de Curvas, onde com poucos parâmetros e um algoritmo eficiente podemos gerar uma curva com as características desejadas.

Figura 4.1. Uma superfície curva: Para desenhá-la de forma eficiente os "retalhos" que a formam devem poder ser calculados por um algoritmo a partir de um conjunto de parâmetros.



4.2. Curvas Paramétricas em 2D

A representação de curvas mais usada em CG foi descoberta independentemente por Pierre Bézier, engenheiro da Renault e Paul de Casteljau, engenheiro da Citroën.

Ambos os engenheiros desenvolveram um esquema de plotagem de curvas que possui ao mesmo tempo raízes paramétricas e analíticas:

- Os valores dos parâmetros são pontos no espaço.
- Estes pontos são ligados através de polinômios.

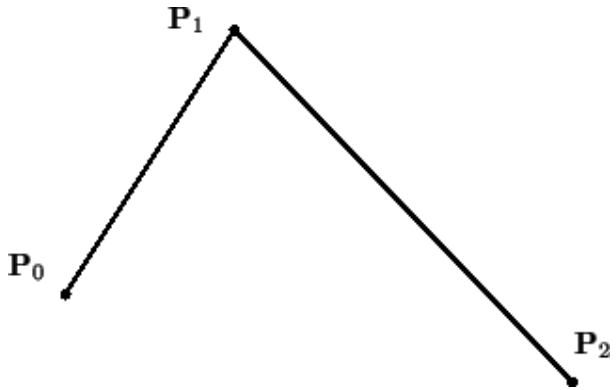
O trabalho de de Casteljau foi pouco anterior ao de Bézier, mas nunca foi publicado. O modelo leva o nome de Bézier. O algoritmo básico para desenho foi inventado por Casteljau.

Subdivisão de curvas usando Dividir-para-Conquistar: O Algoritmo de de Casteljau

Vamos construir uma curva pelo método desenvolvido por de Casteljau através de divisões sucessivas (Vide Figura 4.2.):

- Nesta curva P_0 e P_2 definem os extremos
- P_1 atua como um parâmetro de curvatura ou "ponto de controle".

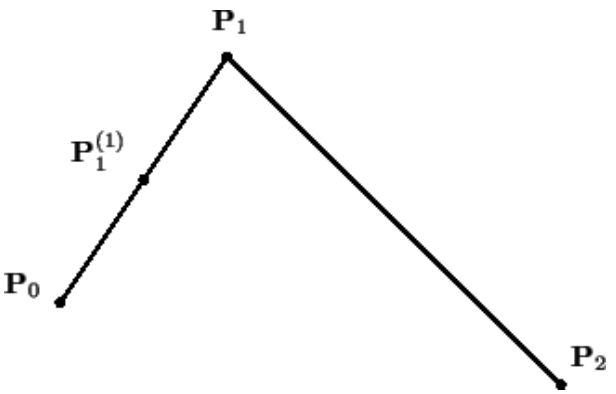
Figura 4.2. Situação inicial



- Recursivamente dividimos e criamos a curva.

Próximo passo: Seja $P_1^{(1)}$ o ponto médio do segmento $\overline{P}_0\overline{P}_1$:

Figura 4.3. Fazemos $P_1^{(1)}$ o ponto médio do segmento $\overline{P}_0\overline{P}_1$



Agora fazemos $P_2^{(1)}$ ser o ponto médio do segmento $\overline{P}_1\overline{P}_2$. Dessa forma dividimos cada um dos dois segmentos ao meio (Vide Figura 4.4.).

Seja agora o ponto $P_2^{(2)}$ o ponto médio do segmento $\overline{P}_1^{(1)}\ \overline{P}_2^{(1)}$ (Vide Figura 4.5.)..

Assim criamos um novo ponto. Pelo algoritmo de de Casteljau este ponto estará sobre a curva.

Figura 4.4. Fazemos $P_2^{(1)}$ o ponto médio do segmento $\overline{P}_1\overline{P}_2$

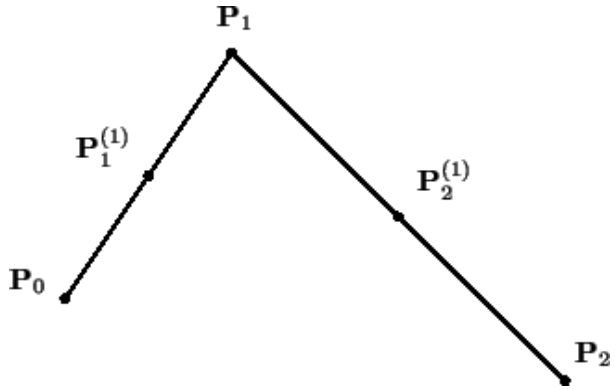
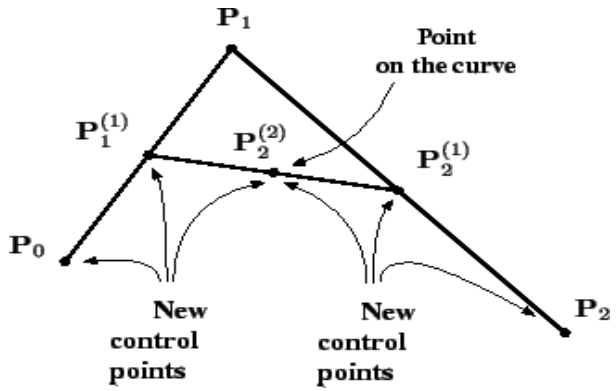


Figura 4.5. Fazemos $P_2^{(2)}$ o ponto médio do segmento $\overline{P}_1^{(1)}\ \overline{P}_2^{(1)}$.



Agora podemos utilizar dois novos conjuntos de pontos como novos pontos de controle para continuar subdividindo a nossa "curva" de forma hierárquica: [P_0, $P_1^{(1)}$ e $P_2^{(2)}$] e [$P_2^{(2)}$ $P_2^{(1)}$ e P_2]. Recursivamente recriamos o problema com mais três novos pontos, fazendo primeiro [P_0, $P_1^{(1)}$ e $P_2^{(2)}$] serem P_0 , P_1 e P_2 e reaplicando o método (Vide Figura 4.6.):.

Seja o novo $P_1^{(1)}$ o ponto médio do segmento que rebatizamos de $\overline{P}_0\overline{P}_1$.(Vide Figura 4.7.)

Figura 4.6. Criamos uma nova sub-curva à esquerda, a qual começaremos a subdividir.

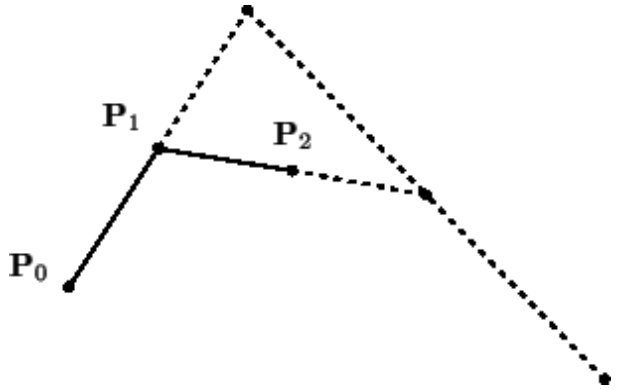
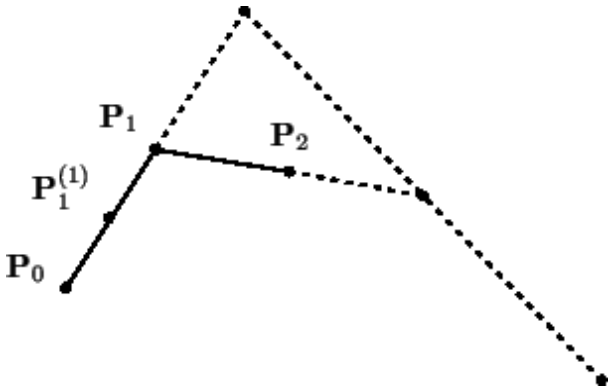


Figura 4.7. Subdividir à esquerda.



Seja o novo $P_2^{(1)}$ o ponto médio do segmento que rebatizamos de $\overline{P}_1\overline{P}_2$. (Vide Figura 4.8.)

Seja agora um novo ponto $P_2^{(2)}$ o ponto médio do segmento $\overline{P}_1^{(1)}$ $\overline{P}_2^{(1)}$ dessa subcurva esquerda (Vide Figura 4.9.)..

Esta seqüência demonstra o processo para a subárvore esquerda[1]. O processo é o mesmo para o outro lado. Demonstraremos a seguir apenas a seqüência de operações, dispensando explicações..

1. Faalmos de ***subárvore*** porque este método é evidentemente um método recursivo e a sua execução gerará uma árvore de chamadas como qualquer outro algoritmo usando a filosofia divide-and-conquer (vide por exemplo o algoritmo do método de ordenação Quicksort).

Figura 4.8. Subdividir à direita do segmento esquerdo.

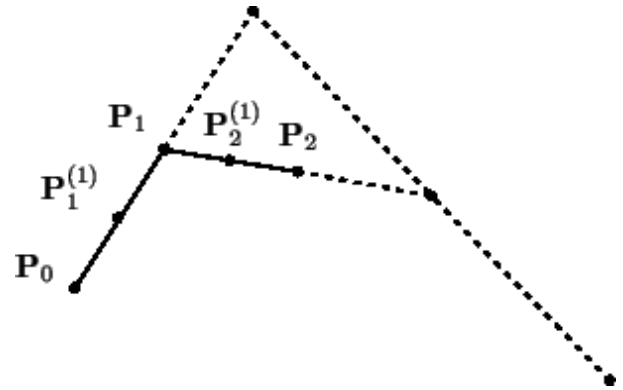
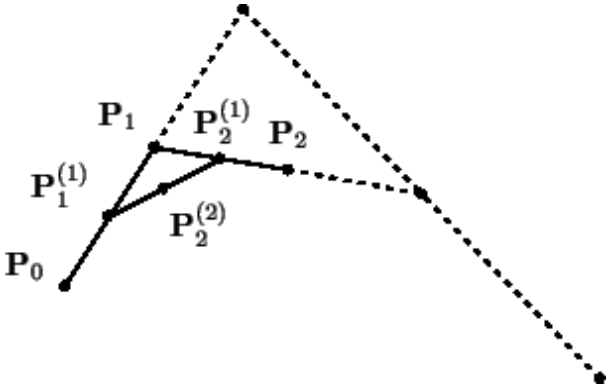


Figura 4.9. Fazemos $P_2^{(2)}$ o ponto médio do novo segmento $\overline{P}_1^{(1)}\,\overline{P}_2^{(1)}$.

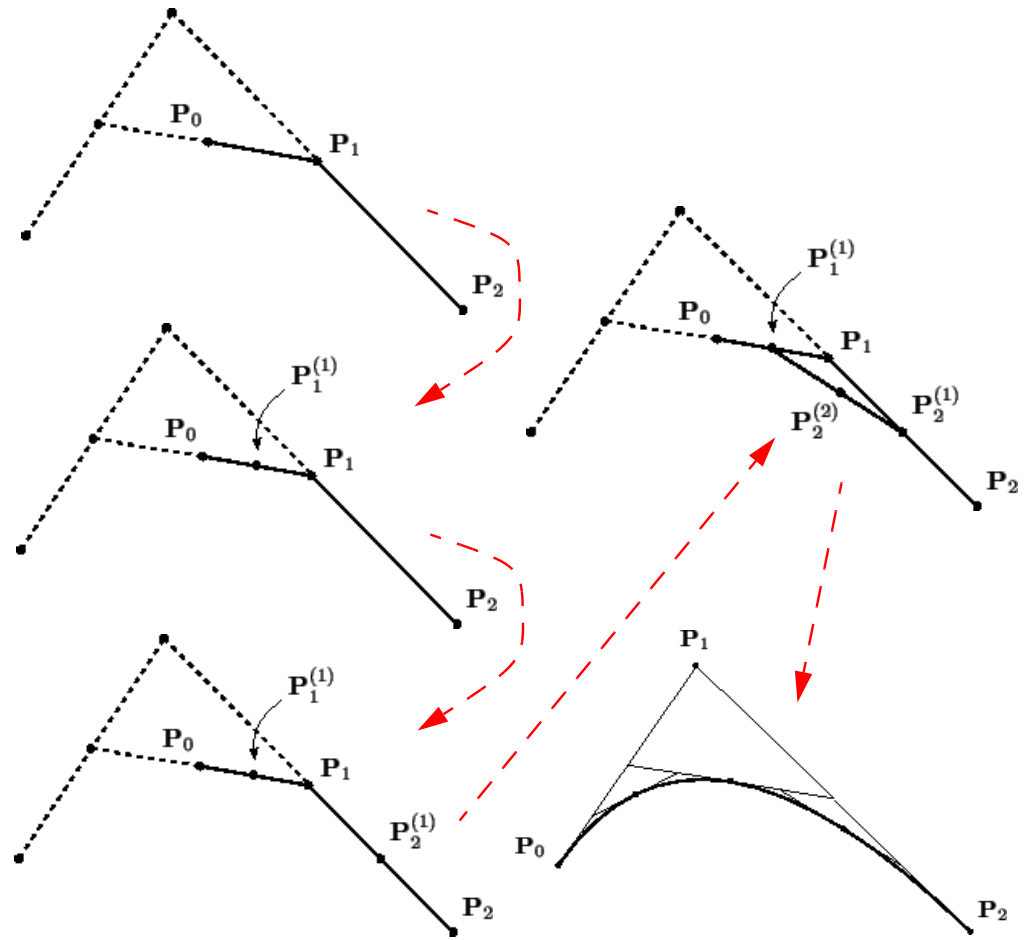


Quando decidimos terminar, podemos plotar a curva desejada. A condição de término é a profundidade da árvore de chamadas recursivas e é definida pelo usuário/programador. É a condição de término que determina a qualidade da curva.

Vantagens:

- Precisão ilimitada
- Suporte a Efeitos de Zoom

Figura 4.10. Fazemos $P_2^{(2)}$ o ponto médio do novo segmento $\overline{P}_1^{(1)}\,\overline{P}_2^{(1)}$.



Desvantagens:

- Muitos cálculos de pontos médios
- Recursão: Consumo de muita memória
- Muito custoso realizar desenho incremental partindo-se de um ponto inicial
- Impossível determinar a posição de um ponto a uma dada distância do início sem desenhar toda a curva

4.3. Curvas Cúbicas Paramétricas em 2D

Antes de discutirmos algum tipo de curva cúbica paramétrica específica, vamos dar uma olhada em alguns princípios matemáticos básicos das curvas cúbicas na sua representação paramétrica.

A forma geral de representação de um ponto em uma curva paramétrica é:

$$P(t) = [x(t) \quad y(t) \quad z(t)] \qquad \textbf{(EQ. 4.1)}$$

A fórmula geral de uma Curva Cúbica na Forma Paramétrica:

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \qquad 0 \le t \le 1 \\ z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z \end{aligned} \qquad \textbf{(EQ. 4.2)}$$

As derivadas com relação a t são todas iguais. Aqui a derivada de x em relação a t:

$$\frac{dx}{dt} = a_x t^2 + b_x t + c_x \qquad \textbf{(EQ. 4.3)}$$

As três derivadas formam o vetor tangente. As inclinações da curva são funções das componentes tangentes:

$$\frac{dy}{dx} = \frac{dy/dt}{dx/dt} \quad , \qquad \frac{dx}{dz} = \frac{dx/dt}{dz/dt} \quad , \quad \text{etc} \qquad \textbf{(EQ. 4.4)}$$

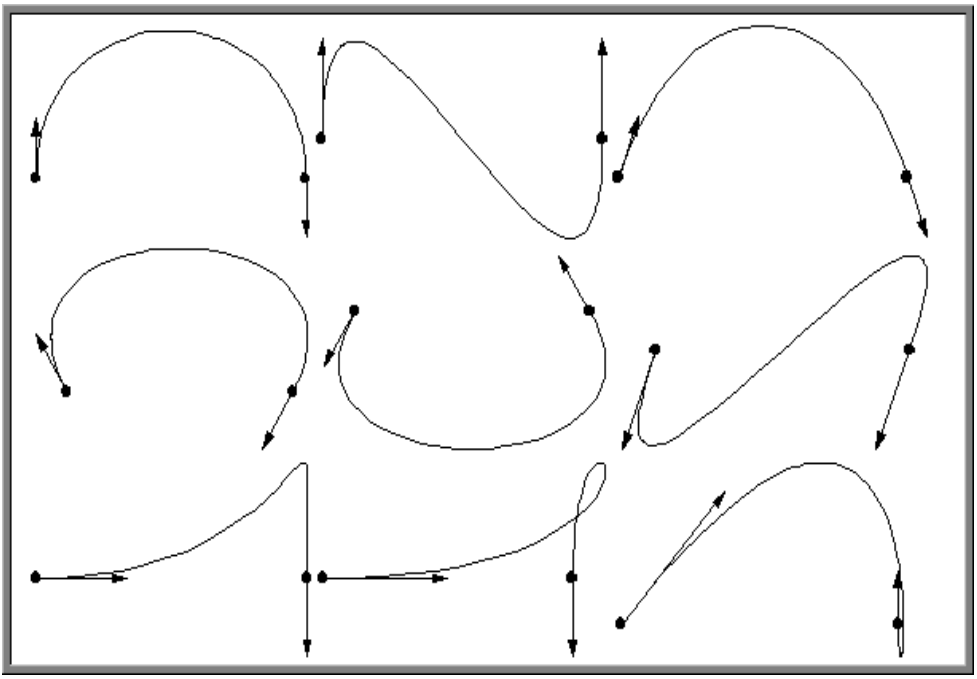
Com estas definições em mente, podemos passar a ver as curvas específicas usadas em CG.

4.4. Curva de Hermite

A curva de Hermite é determinada por dois pontos finais P_1 e P_4 e suas tangentes R_1 e R_4. A partir daí queremos encontrar os coeficientes $a_{(x,y,z)}$,$b_{(x,y,z)}$,$c_{(x,y,z)}$ e $d_{(x,y,z)}$ tais que:

$$\begin{aligned} &x(0) = P_{1_x}, x(1) = P_{4_x}, x'(0) = R_{1_x}, x'(1) = R_{4_x} \\ &y(0) = P_{1_y}, y(1) = P_{4_y}, y'(0) = R_{1_y}, y'(1) = R_{4_y} \\ &x(0) = P_{1_z}, x(1) = P_{4_z}, x'(0) = R_{1_z}, x'(1) = R_{4_z} \end{aligned} \qquad \textbf{(EQ. 4.5)}$$

Figura 4.11. Várias curvas de Hermite. Observe-se os vetores R_1 e R_4, que são responsáveis pela curvatura.



Reescrevendo matricialmente x(t) temos:

$$x(t) = [t^3 t^2 t 1] \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}_x \equiv a_x t^3 + b_x t^2 + c_x t + d_x \qquad \textbf{(EQ. 4.6)}$$

Reescrevendo o vetor coluna dos coeficientes como C_x:

$$x(t) = [\, t^3 \;\; t^2 \;\; t \;\; 1] \cdot C_x \qquad \textbf{(EQ. 4.7)}$$

Reescrevendo o vetor linha das potências de t como T:

$$x(t) = T \cdot C_x \qquad \textbf{(EQ. 4.8)}$$

Podemos expressar as condições da Eq. 5.5 (para x) usando a Eq. 5.7:

$$x(0) = P_{1_x} = [\,0\;0\;0\;1\,] \cdot C_x$$
$$x(1) = P_{4_x} = [\,1\;1\;1\;1\,] \cdot C_x \qquad \textbf{(EQ. 4.9)}$$

Para expressar as condições para os vetores tangentes, diferenciamos 5.7 em relação a t:

$$x'(t) = [\,3t^2\;2t\;1\;0\,] \cdot C_x \qquad \textbf{(EQ. 4.10)}$$

Assim:

$$x'(0) = R_{1_x} = [\,0\;0\;1\;0\,] \cdot C_x$$
$$x'(1) = R_{4_x} = [\,3\;2\;1\;0\,] \cdot C_x \qquad \textbf{(EQ. 4.11)}$$

Dessa forma, agora as condições em 5.9 e 5.11 podem ser expressas em uma única equação matricial:

$$\begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot C_x \qquad \textbf{(EQ. 4.12)}$$

Invertendo a matriz 4x4 atingimos o objetivo de calcular C_x:

$$C_x = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x = M_H \cdot G_{H_x} \qquad \textbf{(EQ. 4.13)}$$

Onde M_H é a **Matriz de Hermite** e G_H é o **vetor de geometria**. Desta forma obtemos:

$$\begin{aligned} x(t) &= T \cdot M_H \cdot G_{H_x} \\ y(t) &= T \cdot M_H \cdot G_{H_y} \\ z(t) &= T \cdot M_H \cdot G_{H_z} \end{aligned} \qquad \textbf{(EQ. 4.14)}$$

Substituindo na forma geral de representação de um ponto em uma curva paramétrica:

$$P(t) = T \cdot M_H \cdot G_H \qquad \textbf{(EQ. 4.15)}$$

Dessa forma podemos calcular qualquer ponto entre P_1 e P_4 em uma curva dada pelos vetores tangentes R_1 e R_4. Como? Se tomamos o produto TM_H temos:

$$TM_H = [(2t^3 - 3t^2 + 1)(-2t^3 + 3t^2)(t^3 - 2t^2 + t)(t^3 - t^2)]$$

(EQ. 4.16)

Multiplicando (EQ. 4.16) por G_{Hx} obtemos:

$$\begin{aligned} x(t) = T \cdot M_H \cdot G_{H_x} = {} & P_{1_x}(2t^3 - 3t^2 + 1) + P_{4_x}(-2t^3 + 3t^2) \\ & + R_{1_x}(t^3 - 2t^2 + t) + R_{4_x}(t^3 - t^2) \end{aligned} \qquad \textbf{(EQ. 4.17)}$$

As quatro funções de t no produto TM_H são chamadas de funções de suavização ou ***blending functions***.

Continuidade de Curvas de Hermite Subseqüentes

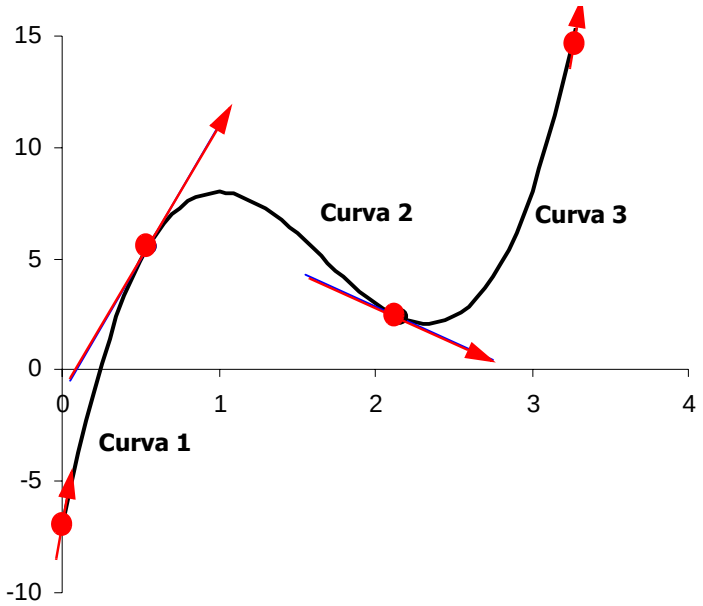
É possível ligar-se facilmente várias curvas de Hermite para obter uma forma mais complexa, basta que algumas poucas condições estejam satisfeitas para que se obtenha um efeito bastante natural.

Para obter-se continuidade de primeira derivada (continuidade de tangência) $C^{(1)}$ em P devemos modelar curvas de Hermite subseqüentes da seguinte forma:

$$\begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x \gg \begin{bmatrix} P_4 \\ P_7 \\ k_1 R_4 \\ R_7 \end{bmatrix}_x \gg \begin{bmatrix} P_7 \\ P_{10} \\ k_2 R_7 \\ R_{10} \end{bmatrix}_x \qquad \textbf{(EQ. 4.18)}$$

A repetição do primeiro ponto de controle e da direção do vetor de direção garante a continuidade de primeira derivada de várias curvas interligadas.

Figura 4.12. Continuidade $C^{(1)}$: Curvas de Hermite possuem continuidade de primeira derivada se os vetores final de uma e inicial da seguinte forem colineare e de mesma direções.



4.5. CONTINUIDADE DE CURVAS INTERLIGADAS

Para conseguirmos modelar a estrutura curva que desejamos através de cúbicas, necessitamos utilizar várias curvas interligadas:

- para obter um efeito de realismo essas curvas têm de ser contínuas

- existem vários graus de continuidade

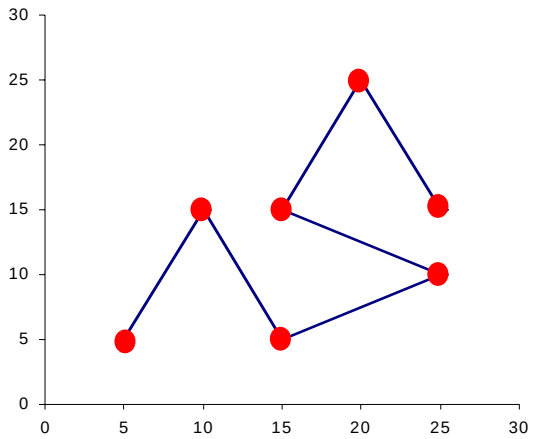
Designamos os graus de continuidade como $\mathbf{C^{(grau)}}$ (continuidade analítica) ou $\mathbf{G^{(grau)}}$ (continuidade geométrica). A seguir veremos alguns exemplos. Mais tarde, ao apresentarmos as próximas formas de curva, discutiremos em que condições qual grau de continuidade pode ser alcançado.

Continuidade $C^{(0)}$ (direta) ou continuidade $G^{(0)}$ (geométrica)

São:

- curvas que se tocam fisicamente
- descontínuas sob qquer outro ponto de vista

Figura 4.13. Um polígono é um objeto com continuidades direta e geométrica.



Continuidade $C^{(1)}$

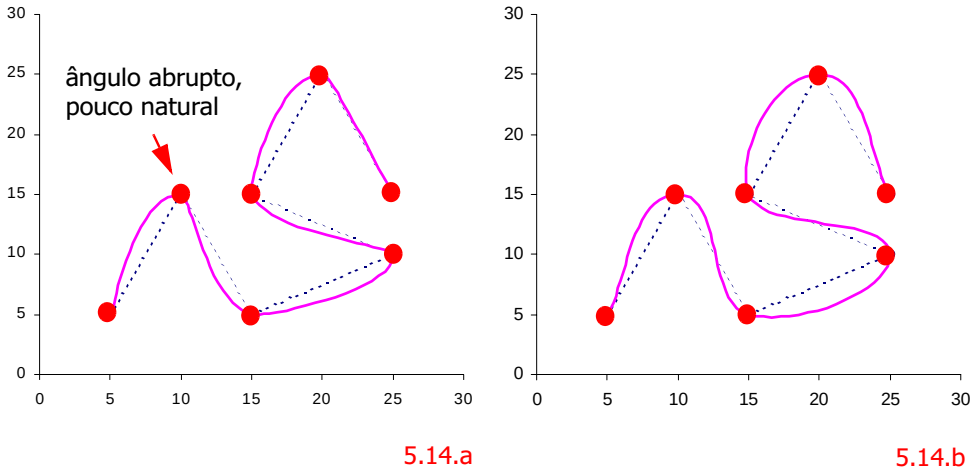
Existe em curvas que possuem vetores tangentes idênticos no ponto de interpolação onde se tocam. Estes vetores são idênticos tanto em direção como em magnitude.

Continuidade $G^{(1)}$

As curvas necessitam possuir apenas vetores tangentes de mesma direção no ponto de interpolação onde se tocam

As magnitudes podem ser diferentes, ocasionando curvaturas diferentes antes e depois da junção

Figura 4.14. A Continuidade C(1) não produz necessariamente curvas realistas



5.14.a 5.14.b

Continuidade C(2):

Os pontos finais das curvas encontram-se e as curvas possuem vetores tangentes exatamente **idênticos** no ponto de interpolação onde se tocam:

- tanto em direção como em magnitude

Além disso, a segunda derivada de ambas as curvas no ponto de interpolação também é idêntica. Esta condição é importante para o efeito de realismo a ser provido pela curva.

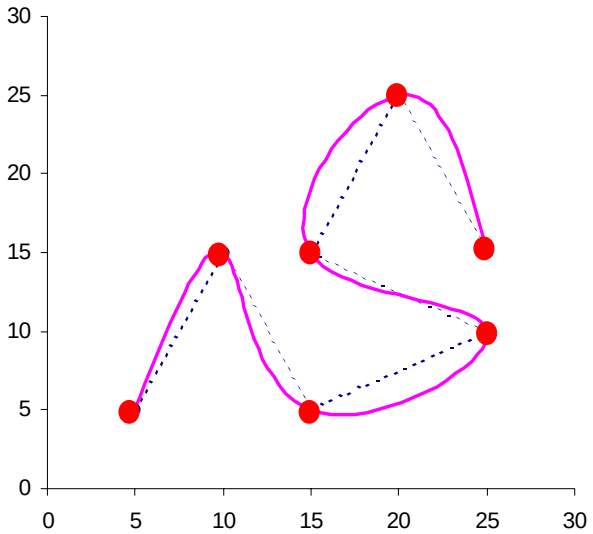
4.6. CURVA DE BÉZIER

A curva desenvolvida por Bézier em 1972 é similar ao modelo de Hermite, mas difere na definição dos vetores tangentes dos pontos extremos.

O modelo de Bézier utiliza 4 pontos (P_1 a P_4), dois extremos e dois de controle internos. As tangentes são definidas pelos segmentos $\overline{P}_1\overline{P}_2$ e $\overline{P}_3\overline{P}_4$.

As tangentes de Hermite possuem a seguinte relação com os pontos de Bézier:

Figura 4.15. Esta curva parece bastante "natural", sendo porém a parte direita difícil de ser distingüida a olho nu da curva 5.14b, da figura anterior. Na parte esquerda vemos que aqui o ângulo abrupto se comporta de tal forma que o efeito é agradável, apesar de ser bastante fechado.



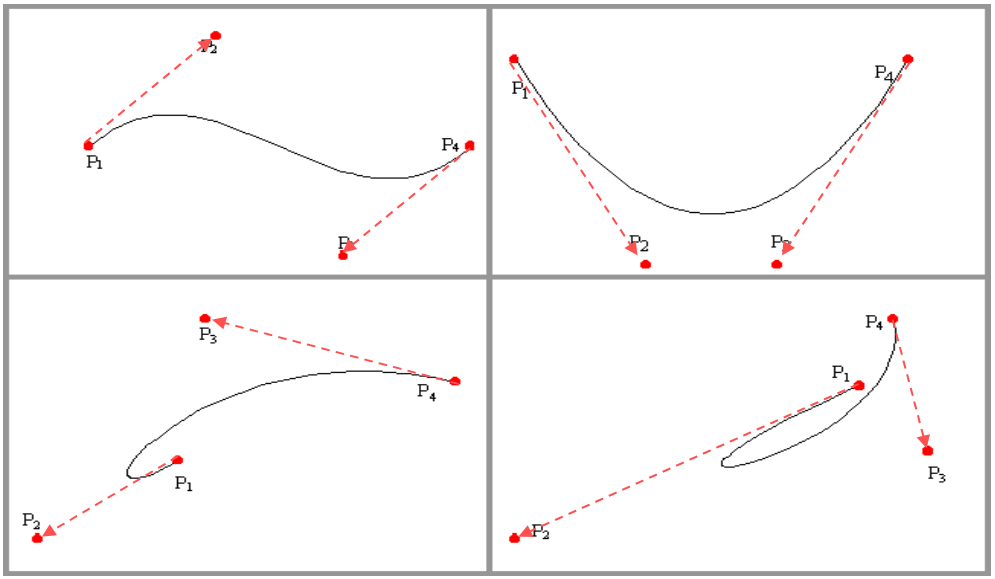
$$R_1 = 3(P_2 - P_1) = P'(0)$$
$$R_4 = 3(P_4 - P_3) = P'(1) \qquad \textbf{(EQ. 4.19)}$$

A relação entre a matriz de geometria de Hermite G_H e a matriz de geometria de Bézier G_B é:

$$G_H = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = M_{HB} G_B \qquad \textbf{(EQ. 4.20)}$$

Substituindo na (EQ. 4.14) obtemos:

Figura 4.16. Relação entre forma das curvas e posição dos pontos de controle.



$$x(t) = TM_H G_{H_x} = TM_H M_{HB} G_{B_x}$$
$$y(t) = TM_H G_{H_y} = TM_H M_{HB} G_{B_y} \qquad \textbf{(EQ. 4.21)}$$

E, definindo o produto $M_H M_{HB}$ em Eq.5.21 como M_B, temos que $x(t) = TM_B G_{Bx}$. A matriz M_B obtida do produto $M_H M_{HB}$ é:

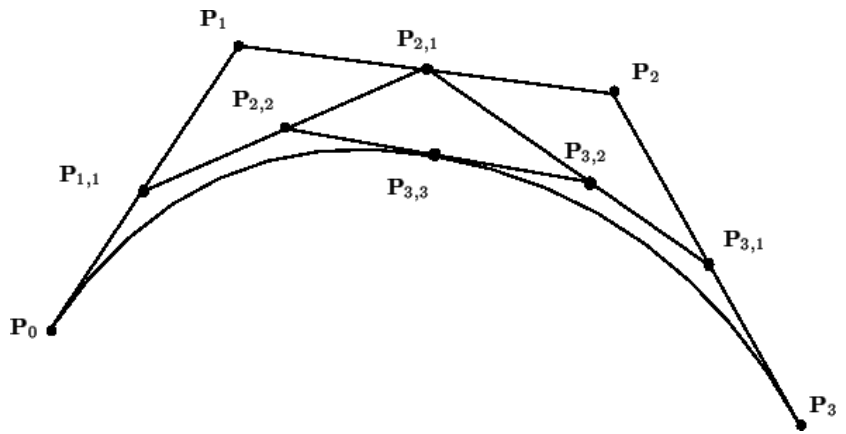
$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad \textbf{(EQ. 4.22)}$$

As ***blending functions*** para Bézier podem ser calculadas da mesma forma que foram em (EQ. 4.16) e (EQ. 4.17) para Hermite.

Condição de continuidade $C^{(1)}$ está garantida quando:

- Duas curvas adjacentes [P_1 a P_4] e [P_4 a P_7] compartilham P_4
- $P_3P_4 = kP_4P_5$

Figura 4.17. Podemos desenhar uma curva com as propriedades da Curva de Bézier através do Algoritmo de de Casteljau, se quisermos...



4.7. B-SPLINES UNIFORMES

O termo Spline remonta às longas tiras de aço flexíveis utilizadas antigamente na construção naval para determinar a forma do casco dos navios.

As splines eram deformadas através de pesos chamados "Ducks", que eram atados a estas em posições determinadas para deformá-las em várias direções da forma desejada.

As splines, a não ser que puxadas de forma muito extrema, possuíam a propriedade de manterem continuidade de segunda ordem.

O equivalente matemático dessas tiras, as ***splines cúbicas naturais***, são portanto curvas polinomiais cúbicas com continuidades C^0, C^1 e C^2, que ***passam através*** dos pontos de controle. Em função disso, as splines, por possuírem um grau de continuidade a mais que a continuidade inerente às curvas de Bézier e de Hermite, são consideradas mais suaves como elementos de interpolação.

Os coeficientes polinomiais das splines cúbicas naturais, no entanto, são dependentes de todos os ***n*** pontos de controle. Isto implica em inversões de matrizes ***n***+1 por ***n***+1, que devem ser repetidas toda vez que um ponto de controle é movido, pois cada ponto afeta toda a curva. Isto é computacionalmente desinteressante.

B-splines são segmentos de curva cujo comportamento depende de apenas uns poucos pontos de controle.

As b-splines são tratadas de forma um pouco diferente das curvas de Bézier ou Hermite:

Nas curvas anteriores, a únicas interdependência que existia, era que para conseguir continuidade C1, repetíamos um ponto e garantíamos a colinearidade do segemnto de reta formado pelos pontos de controle na junção.

As b-splines são curvas com muitos pontos de controle, mas que são tratadas como uma **seqüência de segmentos de ordem cúbica**.

Assim: Uma b-spline com $\mathbf{P_0}$ ... $\mathbf{P_m}$ pontos de controle, onde $\boldsymbol{m} >= \mathbf{3}$, é formada por $\boldsymbol{m} - \mathbf{2}$ segmentos cúbicos.

Denotamos estes segmentos por $\mathbf{Q_3}$ a $\mathbf{Q_m}$.

Cada um dos $\boldsymbol{m}$ - 2 segmentos de curva de uma b-spline é definido por quatro dos m + 1 pontos de controle.

Segmento Q_m é definido pelos pontos P_{m-3}, P_{m-2}, P_{m-1} e P_m.

Figura 4.18. Conjunto de 3 splines em seqüência, todas sobre a mesma reta.

O vetor de geometria b-spline G_{BSi} para aproximar o segmento Q_i, que inicia próximo de P_{i-2} e termina próximo de P_{i-1} é:

$$G_{BS_i} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}, \quad 3 \le i \le m \qquad \textbf{(EQ. 4.23)}$$

Esta matriz será recalculada e usada, no caso de b-spline, para interpolar cada um dos segmentos definidos por um **par de pontos P_{i-2}, P_{i-1}**. Assim, a formulação de um segmento i de b-spline fica:

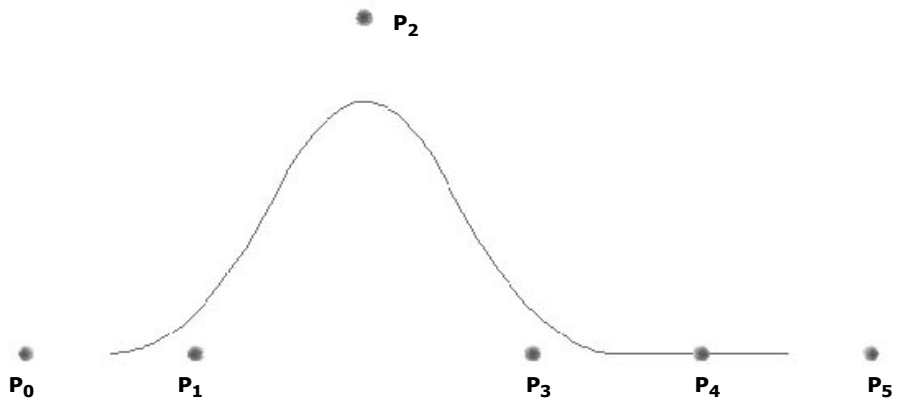
$$Q_i(t) = T_i \cdot M_{BS} \cdot G_{BS_i} \qquad \textbf{(EQ. 4.24)}$$

...onde a ***matriz-b-spline-base***, que relaciona o vetor de geometria às blending functions e os coeficientes polinomiais é:

$$M_{BS} = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \qquad \textbf{(EQ. 4.25)}$$

As blending functions podem ser calculadas similarmente a Bézier e Hermite.

Figura 4.19. Splines com um ponto de controle fora da reta.

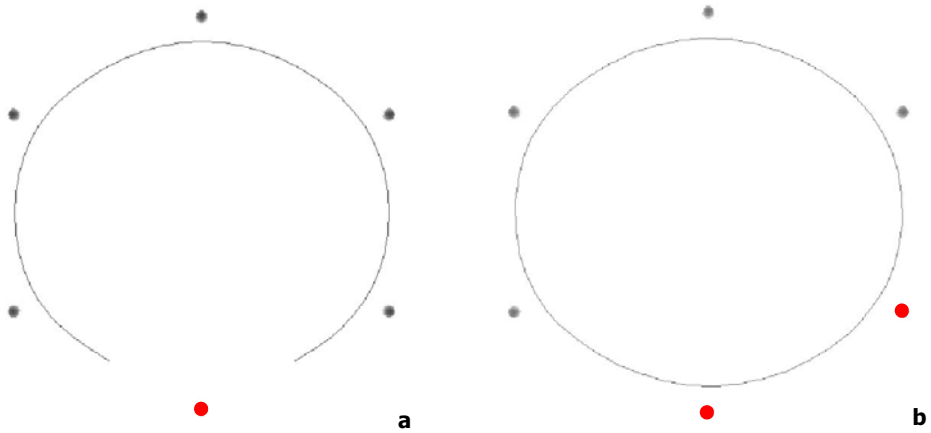


Complexidade

4.8. COMENTÁRIOS

A geração de curvas utilizando as ***blending functions*** possue 3 grandes vantagens sobre o método global hierárquico usando ***divide-and-conquer*** proposto por de Casteljau:

Figura 4.20. Splines semifechadas e fechadas: Se repetimos um ponto (a) e se repetimos dois pontos (b)



- **incremental**: posso ir gerando a curva até onde quero. Por de Casteljau eu tinha de gerar a curva toda e ir refinando.
- **acurácia e passo variáveis**: defino um passo t = 1/k e gero a curva como um conjunto de k-1 polígonos
- **resolve o problema do clipping**: verifico se o fim do próximo segmento t/k está dentro do window usando clipping de pontos.

Desvantagem:

- para cada passo t, tenho de calcular T = [t^3 t^2 t 1], o que é computacionalmente caro.

Otimização do Desenho: Clipping

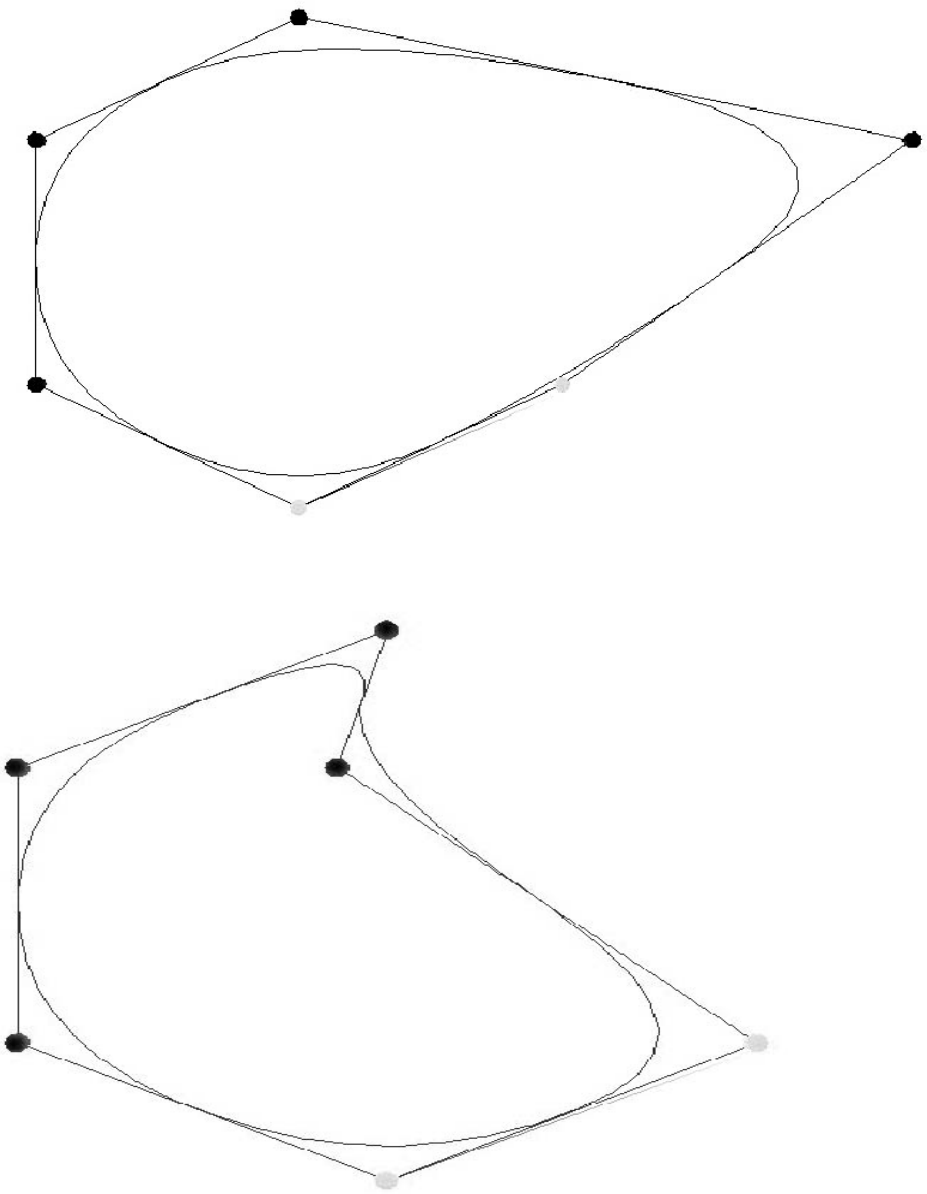
Posso realizar a clipagem de uma forma muito simples: vou desenhando a curva com passo t/k.

A cada ponto gerado, verifico se está dentro da janela.

Se estiver, desenho o segmento de reta.

Se não estiver, executo clipping de segmento de reta para ver onde cortei.

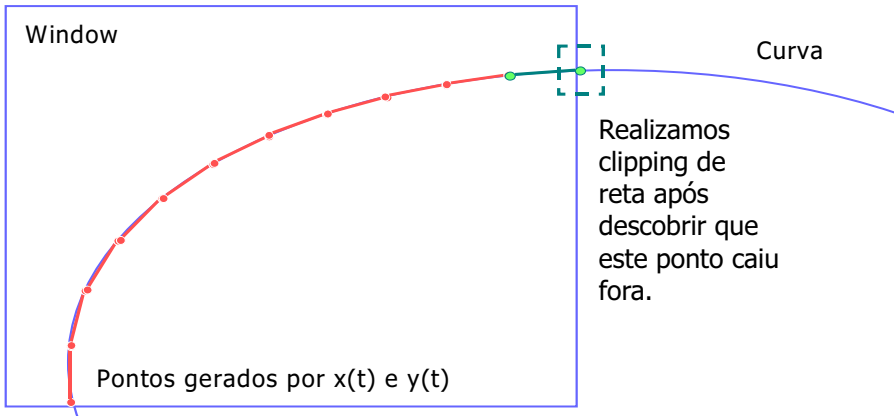
Figura 4.21. Propriedade do Casco Convexo em Splines



Para situações onde existe a possibilidade de uma curva sair e entrar de novo na janela, posso usar pesquisa binária, começando dos dois extremos da curva, para determinar onde entra e onde sai.

A ilustra o procedimento mais simples, sem a utilização da pesquisa binária.

Figura 4.22. Como fazer o clipping de uma curva...



4.9. Desenhando Curvas de Forma Eficiente

Existem duas maneiras básica de se desenhar curvas cúbicas paramétricas:

- Através do cálculo iterativo de x(t), y(t) e z(t) para valores de t incrmentais, plotando-se os pontos calculados e ligando-se estes através de retas, como mostrado na Figura 4.22. Possui a desvantagem de se ter de calcular o valor das bvlending functions para cada passo e, por conseguinte, os valores de t^3 e t^2.
- Através da subdivisão recursiva pelo método de divisão do ponto médio visto no início. As desvantagens deste método jáforma discutidas.

Uma terceira forma muito mais eficiente de se repetidamente calcular o valor de um polinômio é através das ***forward differences***.

Forward Differences

A ***forward difference*** $\Delta f(t)$ de uma função f(t) é dada por:

$$\Delta f(t) = f(t+\delta) - f(t), \quad \delta > 0 \qquad \textbf{(EQ. 4.26)}$$

que pode ser reescrita da seguinte forma:

$$f(t+\delta) = f(t) + \Delta f(t) \qquad \textbf{(EQ. 4.27)}$$

se reescrevermos isto em termos iterativos, teremos:

$$f_{n+1} = f_n + \Delta f_n \qquad \textbf{(EQ. 4.28)}$$

onde n é calculado para intervalos iguais de tamanho δ, de forma que $t_n = n\delta$ e $f_n = f(t_n)$.

Para um polinômio de terceiro grau:

$$f(t) = at^3 + bt^2 + ct + d = T \cdot C \qquad \textbf{(EQ. 4.29)}$$

A ***forward difference*** fica então:

$$\begin{aligned}\Delta f(t) &= a(t+\delta)^3 + b(t+\delta)^2 + c(t+\delta) + d - (at^3 + bt^2 + ct + d) \\ &= 3at^2\delta + t(3a\delta^2 + 2b\delta) + a\delta^3 + b\delta^2 + c\delta\end{aligned} \qquad \textbf{(EQ. 4.30)}$$

Dessa forma, vemos que $\Delta f(t)$ é um polinômio de segundo grau. Isto é ruim, porque para calcular a (EQ. 4.30), temos de calcular $\Delta f(t)$ e uma adição.

Mas podemos aplicar forward differences a $\Delta f(t)$ também. Isto simplifica seu cálculo. Podemos escrever:

$$\Delta^2 f(t) = \Delta(\Delta f(t)) = \Delta f(t+\delta) - \Delta f(t) \qquad \textbf{(EQ. 4.31)}$$

Aplicando isto à Eq. 5.30, temos:

$$\Delta^2 f(t) = 6a\delta^2 t + 6a\delta^3 + 2b\delta^2 \qquad \textbf{(EQ. 4.32)}$$

o que é uma equação de primeira ordem em t.

Reescrevendo 5.31 usando o índice n, obtemos:

$$\Delta^2 f_n = \Delta f_{n+1} - \Delta f_n \qquad \textbf{(EQ. 4.33)}$$

Se reorganizarmos a equação, substituindo n por n - 1, temos:

$$\Delta f_n = \Delta f_{n-1} + \Delta^2 f_{n-1} \quad \textbf{(EQ. 4.34)}$$

Agora, para calcular Δf_n para usar na Eq.5.28, nós calculamos $\Delta^2 f_{n-1}$ e adicionamos o resultado a Δf_{n-1}. Uma vez que $\Delta^2 f_{n-1}$ é linear em t, isto é muito menos trabalho do que calcular Δf_n diretamente a partir do polinômio de segundo grau.

Este processo é repetido mais uma vez para evitar o cálculo direto de (EQ. 4.32). para acharmos $\Delta^2 f(t)$:

$$\Delta^3 f(t) = \Delta(\Delta^2 f(t)) = \Delta^2 f(t+\delta) - \Delta^2 f(t) = 6a\delta^3 \quad \textbf{(EQ. 4.35)}$$

Como a terceira forward difference é uma constante, não é necessário levar este processo adiante, pois a equação não é diferenciável além da terceira derivada. Reescrevendo a (EQ. 4.35) usando o iterador n e $\Delta^3 f(t)$ como uma constante, temos:

$$\Delta^2 f_{n+1} = \Delta^2 f_n + \Delta^3 f_n = \Delta^2 f_n + 6a\delta^3 \quad \textbf{(EQ. 4.36)}$$

Se agora substituirmos n por n-2, completamos o desenvolvimento necessário da equação:

$$\Delta^2 f_{n-1} = \Delta^2 f_{n-2} + 6a\delta^3 \quad \textbf{(EQ. 4.37)}$$

Este resultado pode então ser usado na (EQ. 4.34) para calcular Δf_n, que por usa vez pode então ser usado na (EQ. 4.28) para calcular f_{n+1}.

Para se utilizar as forward differences em um algoritmo que itera de n=0 até nδ=1, nós computamos as condições iniciais com (EQ. 4.29), , (EQ. 4.32) e (EQ. 4.35) para t=0. Estas condições são:

$$\begin{aligned} f_0 &= d \\ \Delta f_0 &= a\delta^3 + b\delta^2 + c\delta \\ \Delta^2 f_0 &= 6a\delta^3 + 2b\delta^2 \\ \Delta^3 f_0 &= 6a\delta^3 \end{aligned} \quad \textbf{(EQ. 4.38)}$$

Pode-se determinar as condições iniciais através do cálculo direto das equações. Chamando de D o vetor das diferenças iniciais, temos:

$$D = \begin{bmatrix} f_0 \\ \Delta f_0 \\ \Delta^2 f_0 \\ \Delta^3 f_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ \delta^3 & \delta^2 & \delta & 0 \\ 6\delta^3 & 2\delta^2 & 0 & 0 \\ 6\delta^3 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = E(\delta) \cdot C \qquad \textbf{(EQ. 4.39)}$$

Com base nessa representação, o algoritmo para plotar uma curva através da utilização de forward differences é apresentado abaixo.

Algoritmo para Desenho de Curvas Paramétricas usando Forward Differences

```
DesenhaCurvaFwdDiff( n,  x, Δx, Δ²x, Δ³x,
                         y, Δy, Δ²y, Δ³y,
                         z, Δz, Δ²z, Δ³z )
início
  inteiro i = 0;
  mova(x, y, z);/* Move ao início da curva */
  enquanto i < n faça
    i <- i + 1;
    x <- x + Δx; Δx <- Δx + Δ²x; Δ²x <- Δ²x + Δ³x;
    y <- y + Δy; Δy <- Δy + Δ²y; Δ²y <- Δ²y + Δ³y;
    z <- z + Δz; Δz <- Δz + Δ²z; Δ²z <- Δ²z + Δ³z;
    desenheAté(x, y, z); /* Desenha reta */
  fim enquanto;
fim DesenhaCurvaFwdDiff;
```

4.10. COMO EU CALCULO OS COEFICIENTES *A,B,C* E *D* ?

A técnica das forward differences exige que se possua n, x, Δx, $\Delta^2 x$, $\Delta^3 x$, y, Δy, $\Delta^2 y$, $\Delta^3 y$, z, Δz, $\Delta^2 z$ e $\Delta^3 z$ para o primeiro ponto da curva como valores iniciais para a primeira iteração. Como estes valores surgem durante o processo iterativo é fácil de perceber: sur-

gem da iteração anterior, mas como fazemos para calcular estes valores para a primeira iteração ?

Para calcular estes valores, eu só preciso jogar os valores dos coeficientes constantes *a, b, c* e *d* na (EQ. 4.39) e assim obtenho os deltas que preciso para iniciar o processo iterativo. De onde surgem estas constantes ?

Para entendermos de onde surgem, basta recapitularmos o que vimos no início deste capítulo: Lembremos que a forma geral de representação de um ponto em uma curva paramétrica é dada pela (EQ. 4.1) Daí construímos a fórmula geral da curva paramétrica, dada pela (EQ. 4.2). Se observarmos este sistema de equações, vamos ver ali os coeficientes *a, b, c* e *d* que precisamos para o nosso cálculo inicial. São os coeficientes da curva paramétrica!

Estes coeficientes são constantes no correr de uma curva e somente precisam ser calculados uma vez, para toda ela.

Para fazer isso, reescrevemos matricialmente estas equações e representamos os coeficientes através da matriz dos coeficientes, como na (EQ. 4.6). Agora só temos de encontrar uma forma de isolá-los em uma equação onde conheçamos os outros dados.

Para tanto seguimos o caminho desenvolvido anteriormente, até chegarmos à equação matricial para uma forma de curva paramétrica, como para Hermite, mostrada na (EQ. 4.12), repetida abaixo para a coordenada x.

$$C_x = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x = M_H \cdot G_{H_x} \qquad \textbf{(EQ. 4.12)}$$

Invertendo a matriz 4x4 atingimos o objetivo de calcular C_x:

$$C_x = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x = M_H \cdot G_{H_x} \qquad \textbf{(EQ. 4.13)}$$

O mesmo processo temos de fazer para os eixos y e z. Esta possibilidade provavelmente não havia chamado nossa atenção antes, pois se usarmos as blending functions não precisamos explicitamente desta equação. Para as forward differences vamos ter de calcular explicitamente Cx, Cy e Cz para podermos calcular os valores iniciais dos deltas para o algoritmos iterativo.

Agora ficou fácil. Para aplicarmos este método para as outras curvas só temos de considerar o seguinte:

Hermite

$C_x(\text{Hermite}) = M_H \,.\, G_{xH}$

$C_y(\text{Hermite}) = M_H \,.\, G_{yH}$

$C_z(\text{Hermite}) = M_H \,.\, G_{zH}$

Bèzier

$C(\text{Bèzier}) = M_B \,.\, G_B$ com G_B matriz de geometria para x,y e z

Spline

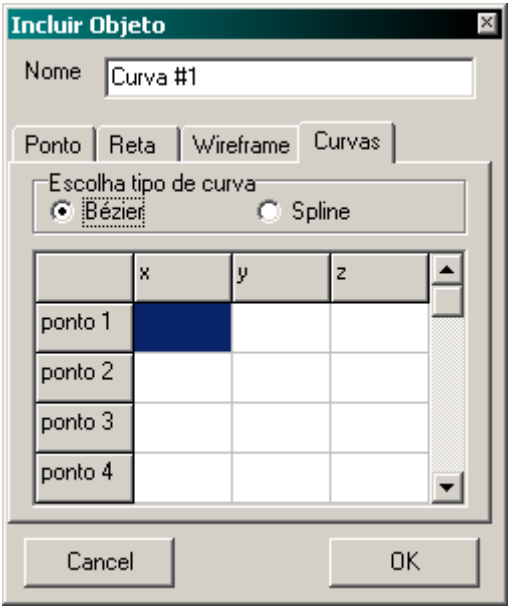
$C(\text{b-Spline}) = M_{BS} \,.\, G_{BS}$ com G_{BS} matriz de geometria para x,y e z

4.11. Exercício 1.4: Implementando Curvas em 2D

1.4a: Blending Functions

Implemente a curva de Hermite, Spline ou Bézier como mais um objeto gráfico de seu sistema. Utilize as blending functions como método de cálculo. Um objeto Curva2D poderá conter uma ou mais curvas com continuidade no mínimo G(0). Crie uma interface para entrar com estes dados. Implemente a clipagem para esta curva utilizando o método descrito anteriormente.

Figura 4.23. Possível forma de implementar a janela para entrada de dados para um objeto do tipo Curva. Este exemplo já mostra a janela prevendo objetos-curva em 3D e oferece possibilidade de entrada da coordenada *z*. A barra de rolagem ao lado permite rolar a interface e incluir tantos pontos quantos se desejar no caso de B-Splines.



1.4b: Forward Differences

Implemente no seu programa gráfico b-splines utilizando Forward Differences. Deve ser possível ao usuário definir uma curva com quantos pontos de controle desejar, entrando simplesmente com uma lista de pontos, desde que tenhamos n > 3.