The Cyclops Project
German-Brazilian Cooperation Programme on IT
CNPq GMD DLR

Computação Gráfica:

Aula 5:
Curvas Paramétricas em 2D
Parte 2: B-Splines e Forward Differences

Prof. Dr. rer.nat. Aldo von Wangenheim

Departamento de Informática e Estatística - INE/CTC/UFSC

Parte I:

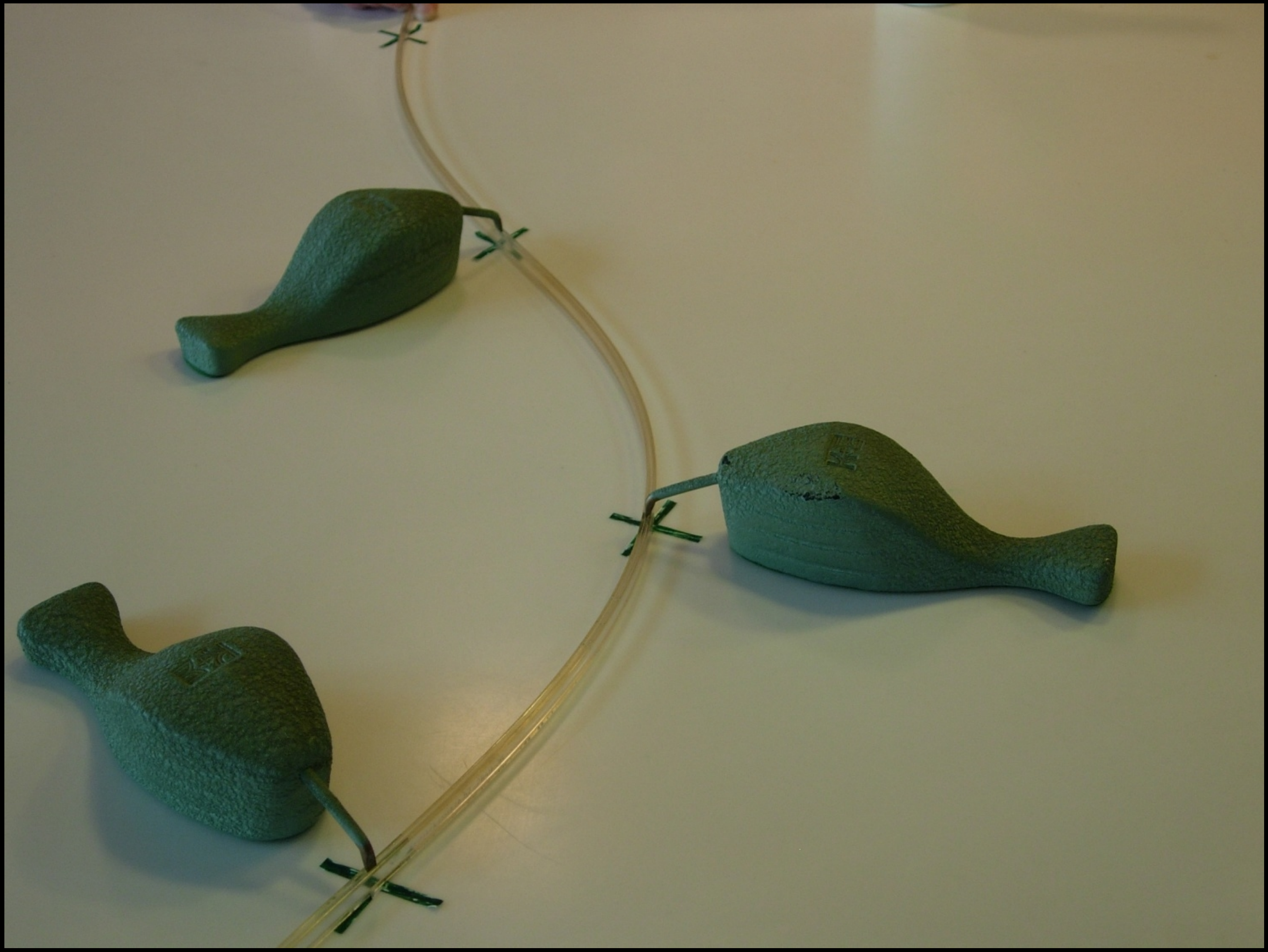
5.4. B-splines Uniformes

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

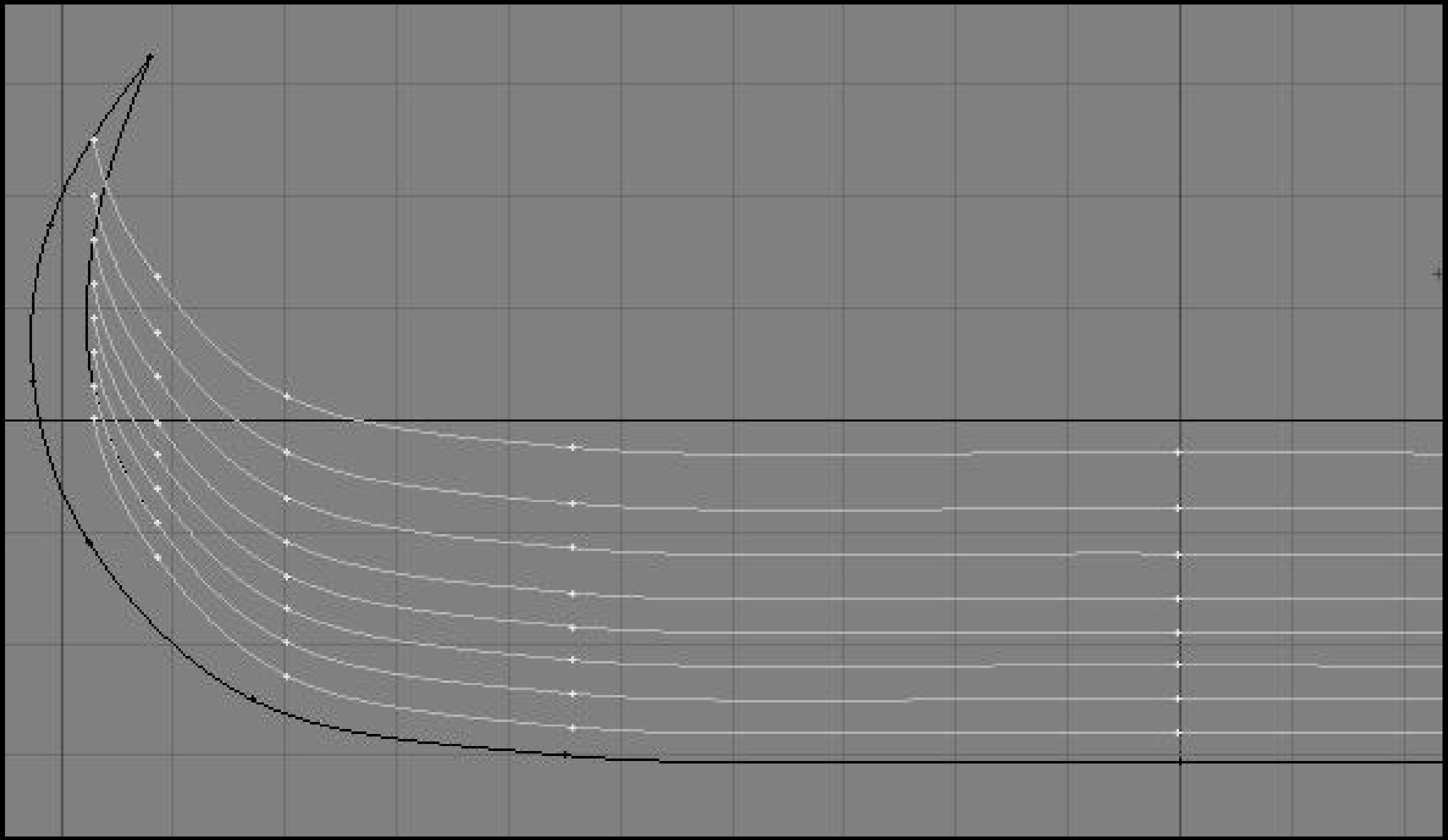
As splines eram deformadas através de pesos chamados "Ducks", que eram atados a estas em posições determinadas para deformá-las em várias direções da forma desejada.

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

Parte I: 5.4. B-splines Uniformes



Parte I: 5.4. B-splines Uniformes



Parte I: 5.4. B-splines Uniformes

Parte I:

5.4. B-splines Uniformes

O equivalente matemático dessas tiras, as splines cúbicas naturais, são 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.

Parte I: 5.4. B-splines Uniformes

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.

Parte I: 5.4. B-splines Uniformes

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

Foram desenvolvidas entre 196
 e 1972 por Carl-Wilhelm Reinhold de Boor, matemático da General Motors

Parte I: 5.4. B-splines Uniformes

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

- Nas curvas anteriores, a única interdependência que existia, era que para conseguir continuidade C^1:
  - repetíamos um ponto e
  - garantíamos a colinearidade do segmento de reta formado pelos pontos de controle na junção.

Parte I: 5.4. B-splines Uniformes

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 P_0 ... P_m pontos de controle, onde m >= 3, é formada por m - 2 segmentos cúbicos.

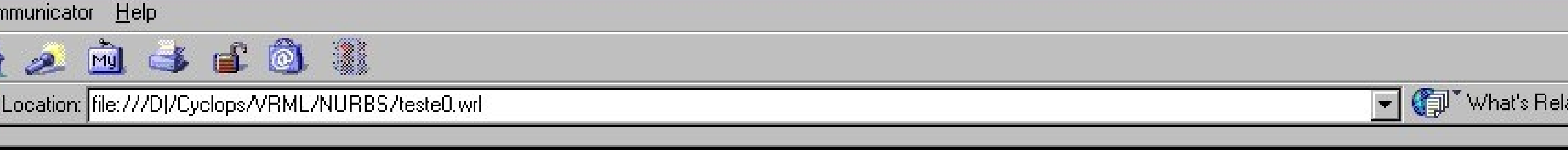
Denotamos estes segmentos por Q_3 a Q_m.

Parte I:

5.4. B-splines Uniformes

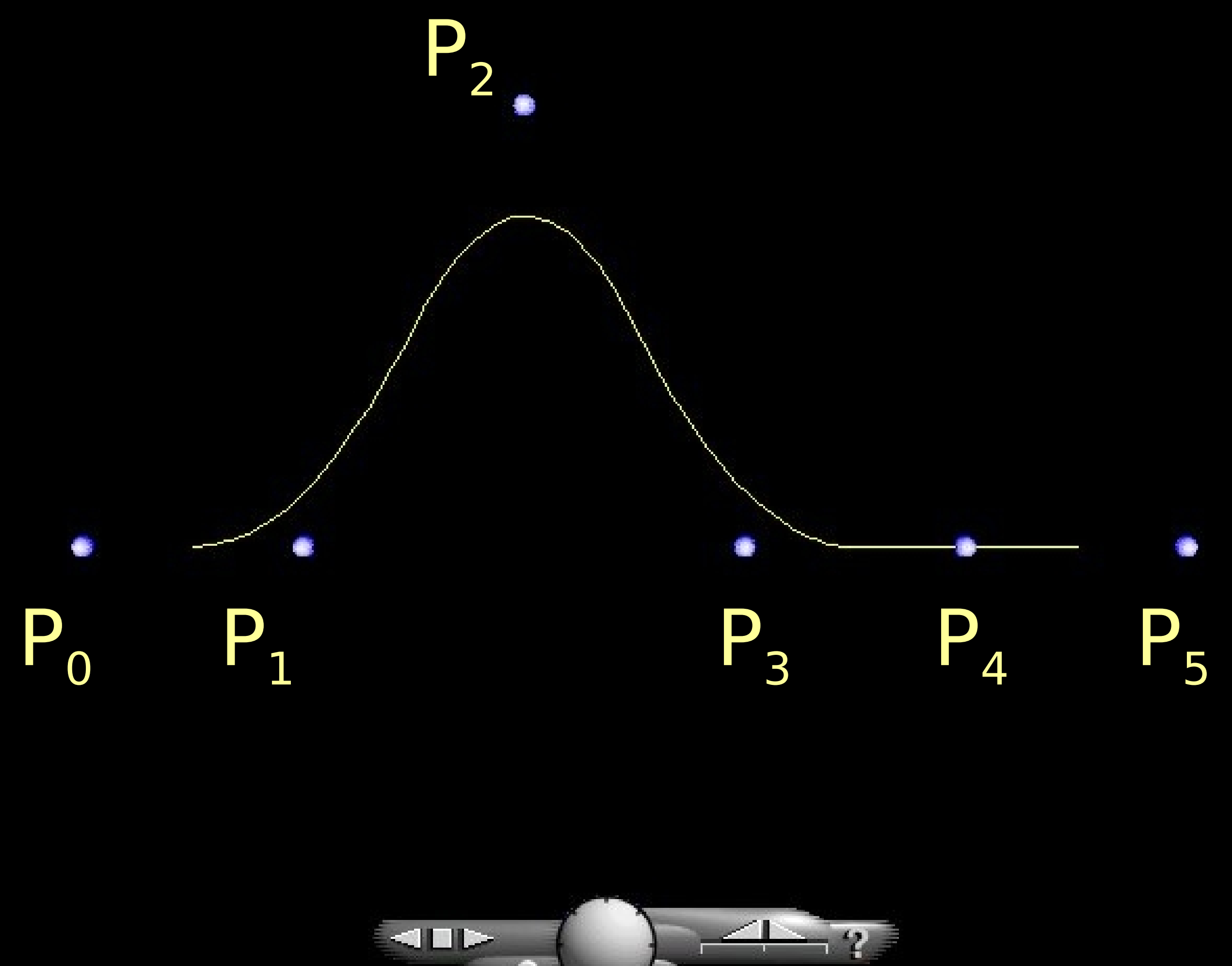
Cada um dos m - 2 segmentos de curva de uma b-spline é definido por quatro dos m + 1 pontos de controle.

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



Netscape com Blaxxun

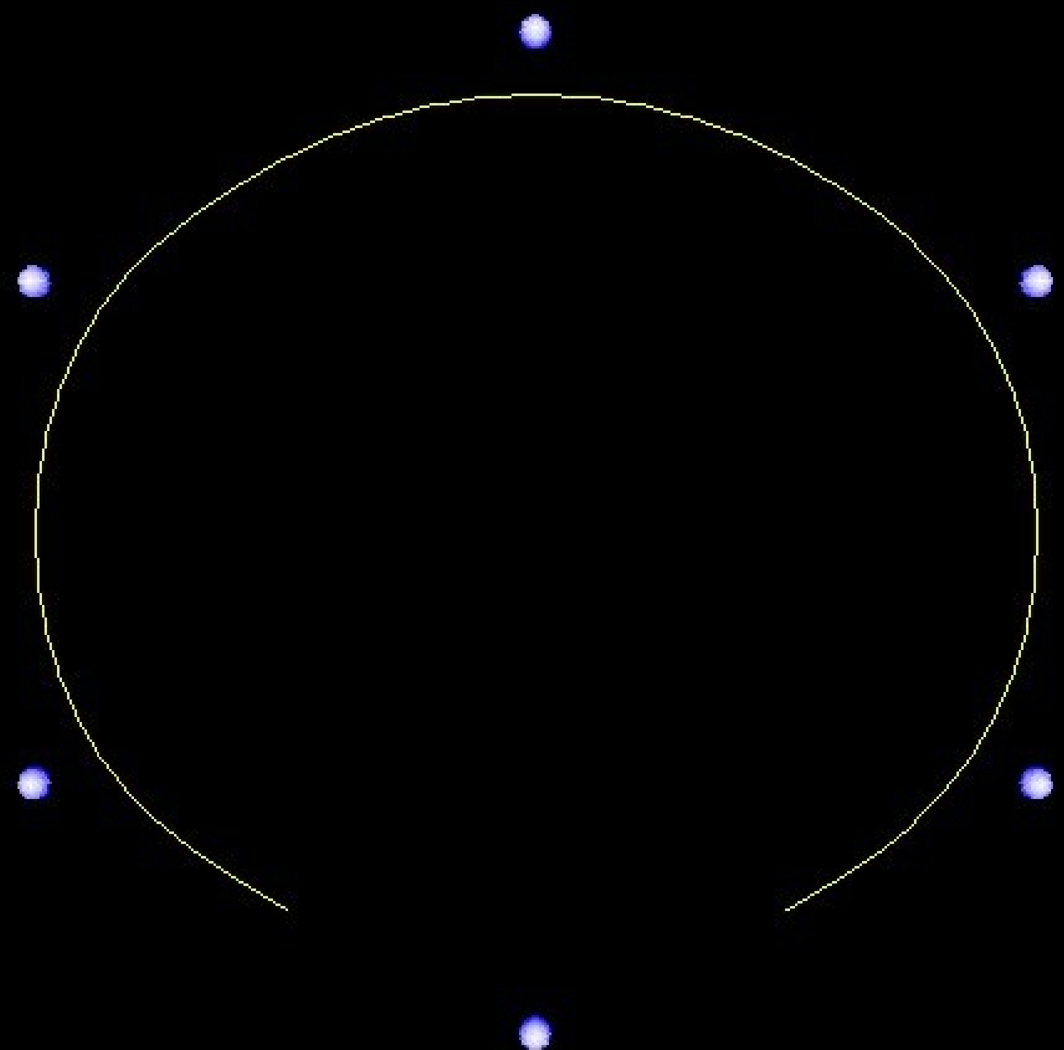
Netscape com Blaxxun

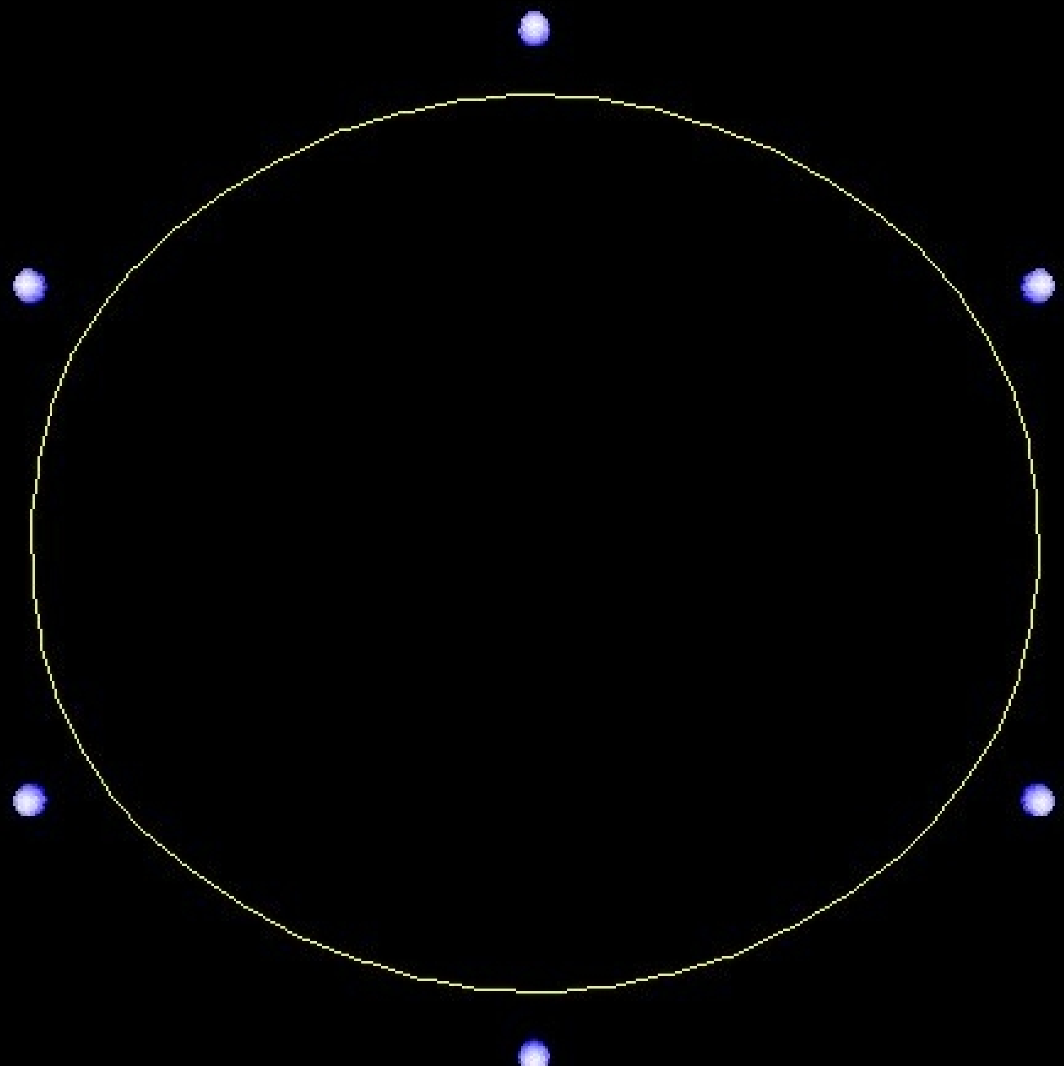


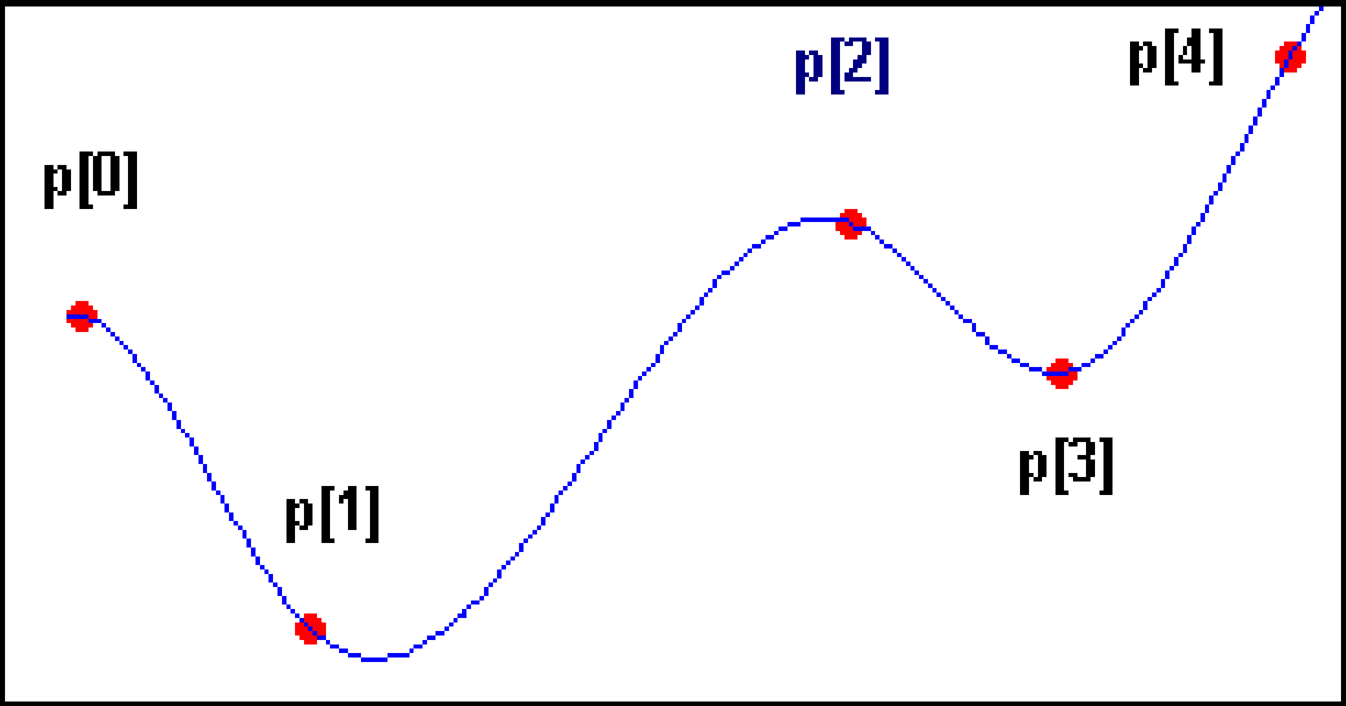
Parte I: 5.4. B-splines Uniformes

Notação VRML/NURBS:

```
geometry NurbsCurve {
        knot [ 0 1 2 3 4 5 6 7 8 ]
        tessellation 50
        controlPoint [0 0 0, 1 0 0, 2 2 0, 3 0 0, 4 0 0, 5 0 0]
        weight [ 1 1 1 1 1 1 ]
}
```






p[0]
p[1]
p[2]
p[3]
p[4]

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:

$$G_{BS_i} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}, \quad 3 \le i \le m \qquad \text{(EQ. 5.23)}$$

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:

$$Q_i(t) = T_i \cdot M_{BS} \cdot G_{BS_i} \qquad \text{(EQ. 5.24)}$$

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

$$M_{BS} = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \qquad \text{(EQ. 5.25)}$$

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

Parte I: 5.6. Desenhando Curvas de Forma Eficiente

Vimos duas maneiras de se desenhar cúbicas paramétricas:

- Através do cálculo iterativo de x(t), y(t) e z(t) para valores de t incrementais, plotando-se os pontos calculados e ligando-se estes através de retas. Possui a desvantagem de se ter de calcular o valor das *blending functions* para cada passo.
- Através da subdivisão recursiva pelo método de divisão do ponto médio visto no início. As desvantagens deste método já foram discutidas.

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

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

$$\Delta f(t) = f(t+\delta) - f(t), \quad \delta > 0 \qquad \text{(EQ. 5.26)}$$

que pode ser reescrita da seguinte forma:

$$f(t+\delta) = f(t) + \Delta f(t) \qquad \text{(EQ. 5.27)}$$

se reescrevermos isto em termos iterativos, teremos:

$$f_{n+1} = f_n + \Delta f_n \qquad \text{(EQ. 5.28)}$$

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:

$$f(t) = at^3 + bt^2 + ct + d = T \cdot C \qquad \text{(EQ. 5.29)}$$

A *forward difference* fica então:

$$\begin{aligned} \Delta f(t) &= a(t+\delta)^3 + b(t+\delta)^2 + c(t+\delta) + d - (at^3 + bt^2 + ct + d) \\ &= 3at^2\delta + t(3a\delta^2 + 2b\delta) + a\delta^3 + b\delta^2 + c\delta \end{aligned} \qquad \text{(EQ. 5.30)}$$

Dessa forma, vemos que $\Delta f(t)$ é um polinômio de segundo grau. Isto é ruim, porque para calcular a Eq. 5.27, 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:

$$\Delta^2 f(t) = \Delta(\Delta f(t)) = \Delta f(t+\delta) - \Delta f(t) \qquad \text{(EQ. 5.31)}$$

Aplicando isto à Eq. 5.30, temos:

$$\Delta^2 f(t) = 6a\delta^2 t + 6a\delta^3 + 2b\delta^2 \qquad \text{(EQ. 5.32)}$$

Esta é uma equação de primeira ordem em t !

Reescrevendo 5.31 usando o índice n, obtemos:

$$\Delta^2 f_n = \Delta f_{n+1} - \Delta f_n \qquad \text{(EQ. 5.33)}$$

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

$$\Delta f_n = \Delta f_{n-1} + \Delta^2 f_{n-1} \qquad \text{(EQ. 5.34)}$$

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. 5.32). para acharmos $\Delta^2 f(t)$:

$$\Delta^3 f(t) = \Delta(\Delta^2 f(t)) = \Delta^2 f(t+\delta) - \Delta^2 f(t) = 6a\delta^3 \qquad \text{(EQ. 5.35)}$$

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. 5.35) usando o iterador n e $\Delta^3 f(t)$ como uma constante, temos:

$$\Delta^2 f_{n+1} = \Delta^2 f_n + \Delta^3 f_n = \Delta^2 f_n + 6a\delta^3 \qquad \text{(EQ. 5.36)}$$

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

$$\Delta^2 f_{n-1} = \Delta^2 f_{n-2} + 6a\delta^3 \qquad \text{(EQ. 5.37)}$$

Este resultado pode então ser usado na (EQ. 5.34) para calcular Δf_n, que por usa vez pode então ser usado na (EQ. 5.28) para calcular f_{n+1}.

Para se utilizar as forward differences em um algoritmo que itera de n=0 até $n\delta=1$, nós computamos as condições iniciais com (EQ. 5.29), (EQ. 5.30), (EQ. 5.32) e (EQ. 5.35) para t=0. Estas condições são:

$$\begin{aligned}
f_0 &= d \\
\Delta f_0 &= a\delta^3 + b\delta^2 + c\delta \\
\Delta^2 f_0 &= 6a\delta^3 + 2b\delta^2 \\
\Delta^3 f_0 &= 6a\delta^3
\end{aligned} \qquad \text{(EQ. 5.38)}$$

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

$$D = \begin{bmatrix} f_0 \\ \Delta f_0 \\ \Delta^2 f_0 \\ \Delta^3 f_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ \delta^3 & \delta^2 & \delta & 0 \\ 6\delta^3 & 2\delta^2 & 0 & 0 \\ 6\delta^3 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = E(\delta) \cdot C \qquad \text{(EQ. 5.39)}$$

Com base nessa representação, o lagoritmo para plotar uma curva é apresentado adiante:

```
algoritmo DesenhaCurvaFwdDiff( n, x, Δx, Δ^2x, Δ^3x,
          y, Δy, Δ^2y, Δ^3y,
          z, Δz, Δ^2z, Δ^3z)
início
   inteiro i = 1;
   real x_velho, y_velho, z_velho;
   x_velho <- x;      /* Guarda início do segmento */
   y_velho <- y;      /* de curva qdo. i = 1       */
   z_velho <- z;
   enquanto i < n faça
   i <- i + 1; /* Termina quando i = n      */
   x <- x + Δx;  Δx <- Δx + Δ^2x;  Δ^2x <- Δ^2x + Δ^3x;
   y <- y + Δy;  Δy <- Δy + Δ^2y;  Δ^2y <- Δ^2y + Δ^3y;
   z <- z + Δz;  Δz <- Δz + Δ^2z;  Δ^2z <- Δ^2z + Δ^3z;
   desenheLinha(x_velho, y_velho, z_velho, x, y, z);
   x_velho<- X; y_velho <- y; z_velho <- z;
   fim enquanto;
fim DesenhaCurvaFwdDiff;
```

Como eu calculo os coeficientes *a*,*b*,*c* e *d* ?

- Para isto basta recapitularmos o que vimos na aula passada...

Forma geral de representação de um ponto em uma curva paramétrica:

$$P(t) = [x(t) \ \ y(t) \ \ z(t)] \qquad \text{Eq. 5.1}$$

Fórmula geral da Curva Cúbica Paramétrica:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y \qquad 0 \leq t \leq 1 \qquad \text{Eq. 5.2}$$

$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

Derivadas com relação a t são todas iguais. Aqui x em relação a t:

$$\frac{dx}{dt} = a_x t^2 + b_x t + c_x \qquad \text{Eq. 5.3}$$

Usando Curvas de Hermite como exemplo...

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:

$$x(0) = P_{1_x},\ x(1) = P_{4_x},\ x'(0) = R_{1_x},\ x'(1) = R_{4_x}$$

$$y(0) = P_{1_y},\ y(1) = P_{4_y},\ y'(0) = R_{1_y},\ y'(1) = R_{4_y} \qquad \text{Eq. 5.5}$$

$$x(0) = P_{1_z},\ x(1) = P_{4_z},\ x'(0) = R_{1_z},\ x'(1) = R_{4_z}$$

Reescrevendo matricialmente x(t) temos:

$$x(t) = [\,t^3\ t^2\ t\ 1\,] \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}_x \equiv a_x t^3 + b_x t^2 + c_x t + d_x \qquad \text{Eq. 5.6}$$

Parte I: 5.3. Curvas de Hermite em 2D

Reescrevendo o vetor coluna dos coeficientes como C_x:

$$x(t) = [\, t^3 \; t^2 \; t \; 1 \,] \cdot C_x \qquad \text{Eq. 5.7}$$

Reescrevendo o vetor linha das potências de t como T:

$$x(t) = T \cdot C_x \qquad \text{Eq. 5.8}$$

Parte I: 5.3. Curvas de Hermite em 2D

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

$$x(0) = P_{1_x} = [\,0\,0\,0\,1\,] \cdot C_x$$
$$x(1) = P_{4_x} = [\,1\,1\,1\,1\,] \cdot C_x$$

Eq. 5.9

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

$$x'(t) = [\,3t^2\; 2t\;\; 1\;\; 0] \cdot C_x$$

Eq. 5.10

Assim:

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

Eq. 5.11

Parte I: 5.3. Curvas de Hermite em 2D

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

$$\begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot C_x \qquad \text{Eq. 5.12}$$

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

$$C_x = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x = M_H^{-1} \cdot G_{H_x} \qquad \text{Eq. 5.13}$$

Achei!!!

Generalizando, temos...

$$C_{x(hermite)} = M_H^{-1} . G_H$$

$$C_{x(bézier)} = M_B^{-1} . G_B$$

$$C_{x(b\text{-}spline)} = M_{BS}^{-1} . G_{BS}$$

Algoritmo de alto nível...

0. Definir um valor para δ (ex.: 0,001) e calcular δ^3 e δ^2 (constantes por toda a curva)

1. Calcular Coeficientes dependendo do método

   $C_{x(hermite)} = M_H^{-1} \cdot G_H$

   $C_{x(bézier)} = M_B^{-1} \cdot G_B$

   $C_{x(b\text{-}spline)} = M_{BS}^{-1} \cdot G_{BS}$

2. Calcular f_0, $\Delta(f_0)$, $\Delta^2(f_0)$ e $\Delta^3(f_0)$ para o primeiro ponto da curva (P_1)

3. Chamar **`DesenhaCurvaFwdDiff`** para os outros pontos...

Exercício

- Implemente no seu programa gráfico b-splines utilizando Forward Differences.
  - Deve ser possível ao usuário definir uma curva com quantos pontos de controle desejar.