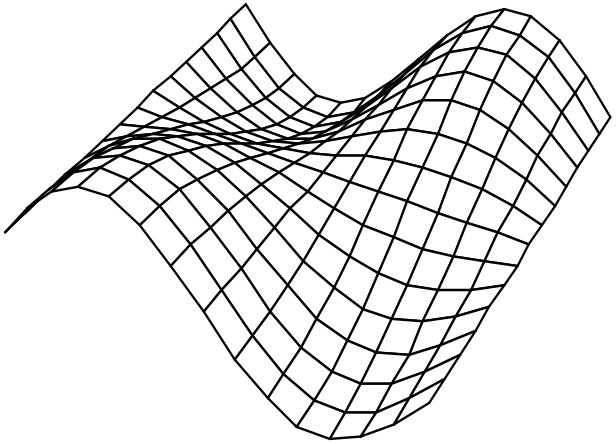
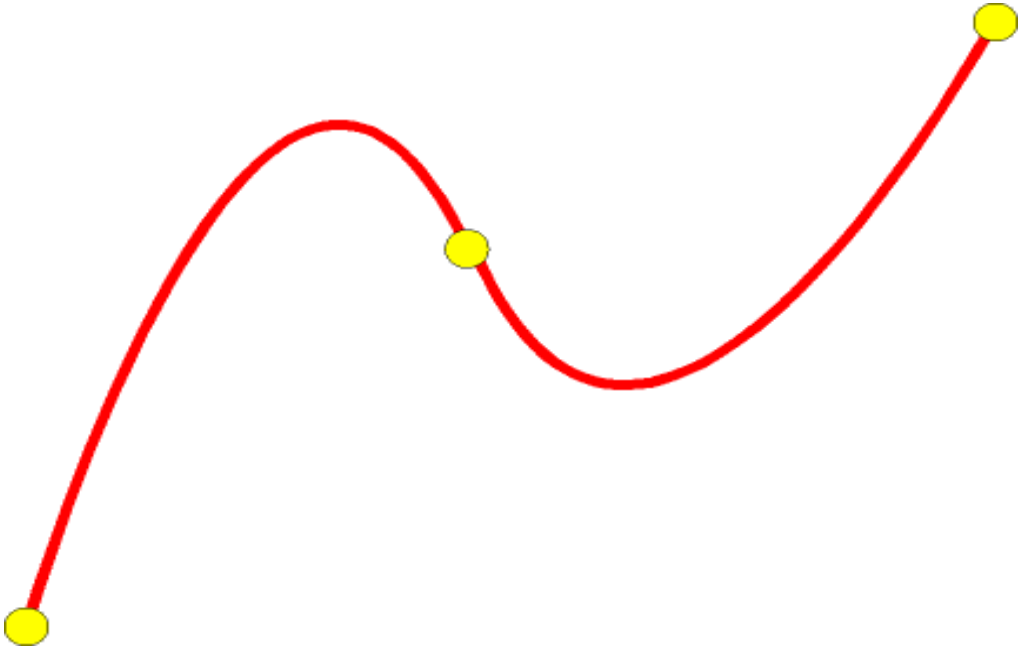


Curvas e Superfícies

Profª Soraia Raupp Musse

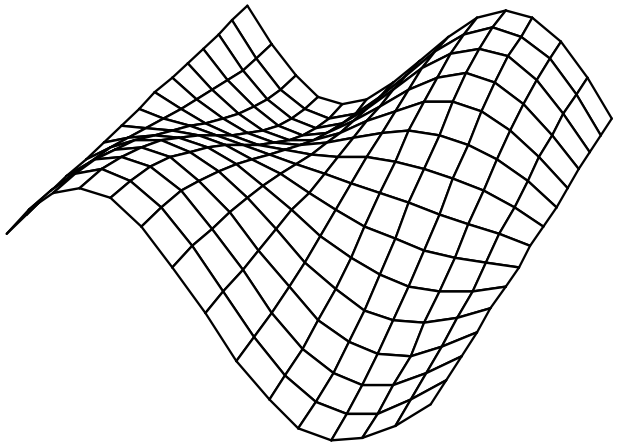
Classificação

- Curvas
  - Apenas comprimento



Classificação

- Superfícies
  - Apenas área
  - Cascas infinitesimalmente finas, ocas
  - Abertas ou fechadas

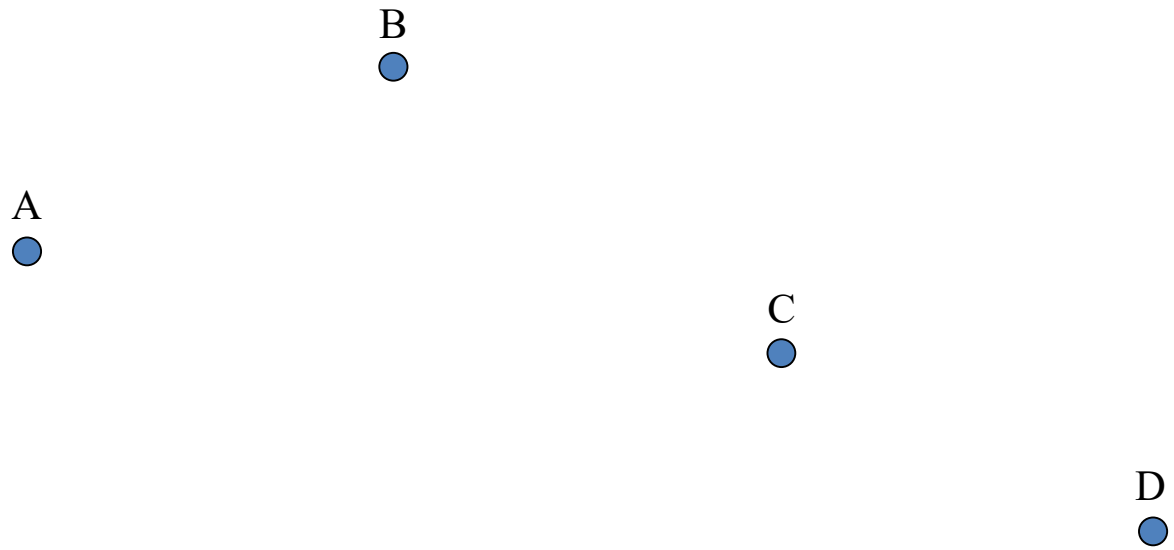


Classificação

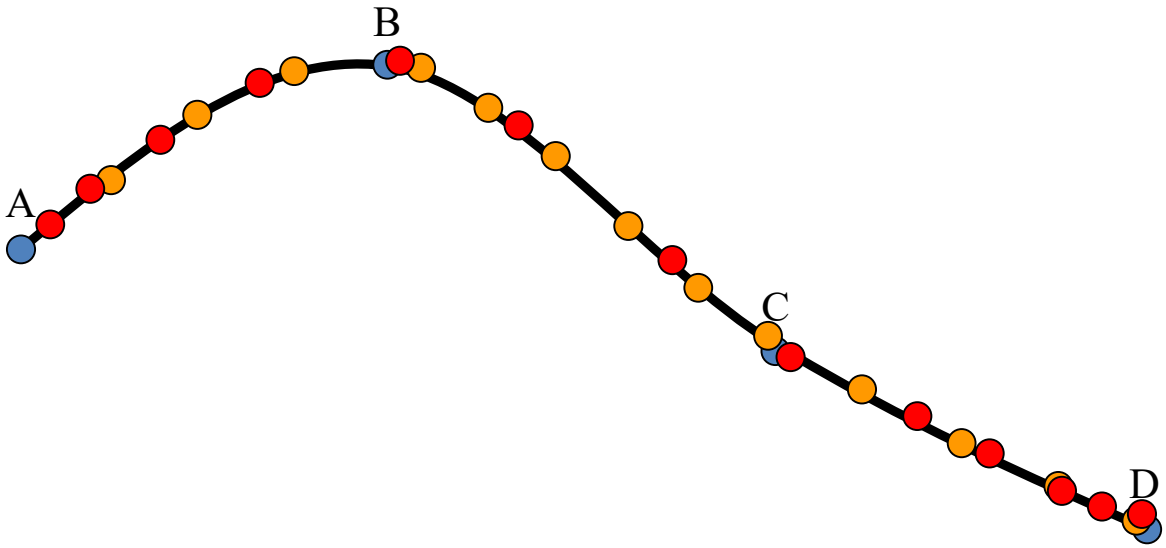
- Sólidos
  - O interior também interessa

Isto é um sólido?

Problema:
Gerar uma curva **suave** que passe por **pontos** específicos



Solução:
gerar uma curva no espaço, distribuindo **pontos** de maneira **suave**

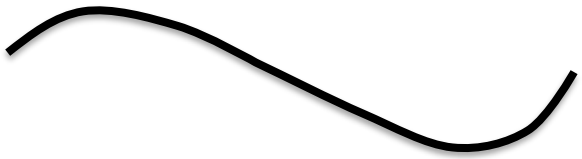


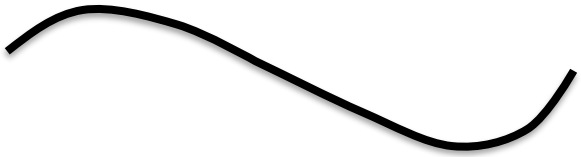
Interpolação

- Princípio básico
  - Alterar de forma incremental a posição de um ponto no espaço

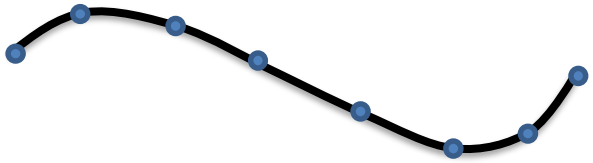
- A base para a maioria das técnicas de animação é algum tipo de interpolação de valores.

Interpolação X Aproximação



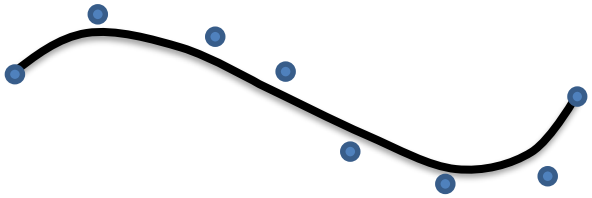


Interpolação X Aproximação



Na interpolação, a curva passa sobre todos os pontos definidos.

Hermite, Catmull-Rom spline



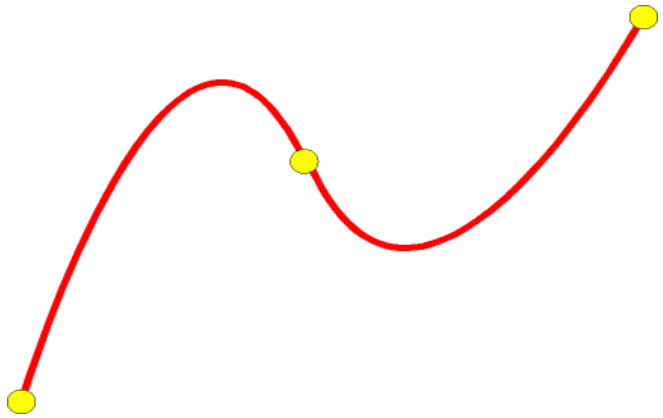
Na aproximação, a curva começa sobre o ponto inicial e termina sobre o final. Os demais pontos são aproximados.

Bézier, curvas B-spline

Como podemos representar uma curva?

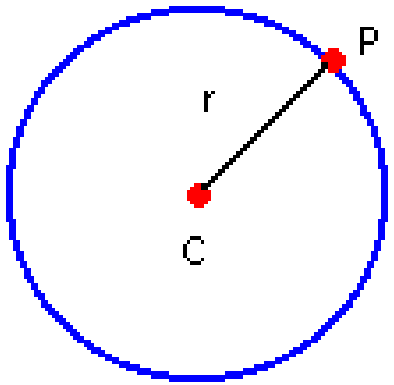
- Localização no espaço de um ponto que se move

- Como podemos descrever este conceito?



Possibilidades de Representações

- Alguns objetos podem ter mais de uma possibilidade para serem representados
- Exemplo: círculo centrado na origem com raio=1



$$d_{CP} = \sqrt{(X_P - X_C)^2 + (Y_P - Y_C)^2} \Rightarrow \sqrt{(x-a)^2 + (y-b)^2} = r \Rightarrow$$
$$\Rightarrow \boxed{(x-a)^2 + (y-b)^2 = r^2}$$

$$x^2 + y^2 = 1$$

Possibilidades de Representações

- Alguns objetos podem ter mais de uma possibilidade para serem representados
- Exemplo: círculo centrado na origem com raio=1

$$x^2 + y^2 = 1$$

Implícita

$$x(\theta) = \cos\theta$$
$$y(\theta) = \operatorname{sen}\theta$$

Paramétrica/explícita

Possibilidades de Representações

- Implícita: definida através de duas variáveis, mas não é dada uma maneira explícita de resolver y em função de x
- Função explícita de x e y em função de outra variável.

$$x^2 + y^2 = 1$$

Implícita

$$x(\theta) = \cos\theta$$
$$y(\theta) = \operatorname{sen}\theta$$

Paramétrica

Tipos de Representação

- Representação não-paramétrica
  - Representação por equações onde uma das coordenadas é determinada em função das outras
- Equações **explícitas**: y = f(x)
  - Adequadas para a geração de novos pontos
    - Exemplos

$$y = f(x)$$
$$y = mx + b$$
$$z = -(Ax + By + D) / C$$

- Equações **implícitas**: f(x,y) = 0
  - Boas para fazer consistência
    - Exemplos

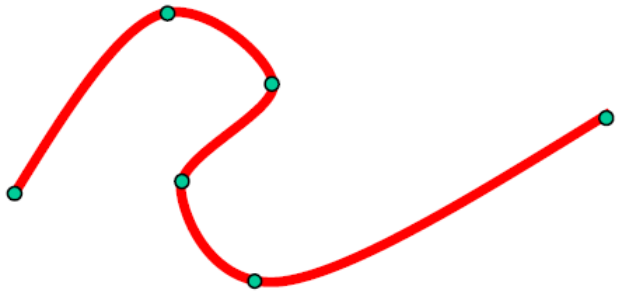
$$f(x, y) = 0$$
$$Ax + By + C = 0$$
$$(x - x_0)^2 + (y - y_0)^2 - r^2 = 0$$

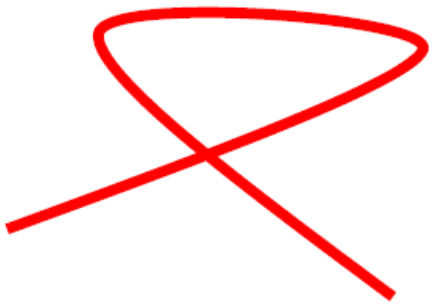
Tipos de Representação

Principais desvantagens das representações **não-paramétrica** em CG

- É difícil definir a equação não paramétrica de uma curva que passe por um conjunto de pontos pré-definidos.

- Não permite a representação de curvas com laços





Tipos de Representação

- Representação Paramétrica
  - Representação por equações onde as coordenadas são obtidas em função de um parâmetro
- Equações paramétricas: $x = f(t)$ e $y = g(t)$
  - Adequadas para gerar uma seqüência de pontos
  - Classificadas de acordo com seus termos: linear (grau 1), quadrática (grau 2), cúbica (grau 3), transcendental (sin, cos, log, ...)

Tipos de Representação

Forma paramétrica

Curvas paramétricas

$x = f(u)$

$y = g(u)$

$z = h(u)$

Superfícies Paramétricas

$x = f(u,v)$

$y = g(u,v)$

$z = h(u,v)$

Exemplo de equações paramétricas

$x = x0 + r \cos \alpha$

$y = y0 + r \sin \alpha$

$\alpha E[0,2\pi]$

Tipos de Representação

Principais vantagens das formas paramétricas em CG

- Resolve os problemas da forma não paramétrica.
- A curva pode ser definida a partir de pontos de controle (é fácil de manipular interativamente).
- A curva pode ou não passar por um conjunto de pontos pré-definidos.
- A curva é aproximada por polinômios que definem as suas várias partes.
- O comportamento da curva em relação a cada um dos eixos é definido por equações independentes
- As coordenadas são obtidas em função de um parâmetro

Representação Paramétrica

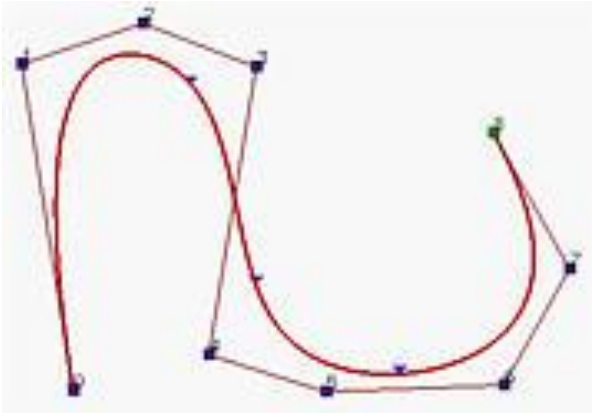
- Para CG, representações paramétricas costumam ser as mais convenientes

- Assim, genericamente, uma curva 3D é
  - $Q(t)=[x(t)\ \ y(t)\ \ z(t)]$

- x(t), y(t), z(t) são chamadas de funções-base (*blending functions*)

Representação Paramétrica

- A curva é definida através de um conjunto de **pontos de controle** que influenciam a forma da curva.
- Os nós são pontos de controle que pertencem à curva.
- A curva pode ser interpolada, passando nesse caso por todos os pontos de controle, ou pode ser aproximada, passando apenas em alguns pontos de controle ou mesmo nenhum.
- Os pontos de controle definem a fronteira de um polígono designado por *convex hull*.



Representação Paramétrica

- Parâmetro (t) é usado para percorrer a curva
  - Pode ser associado ao tempo

    $x(t) = fx(t)$

    $y(t) = fy(t)$

    $z(t) = fz(t)$

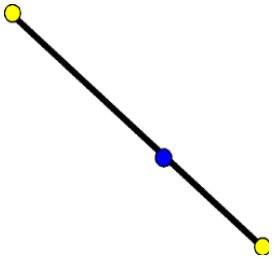
Reta Paramétrica

- $P(t) = P_0 + at$
  - $P_x = P_{x0} + at$
  - $P_y = P_{y0} + at$
  - $P_z = P_{z0} + at$

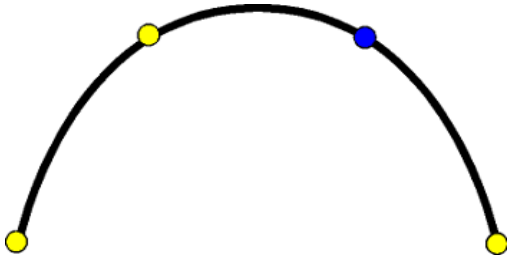
Complexidade

- Quanto mais simples a equação da função de interpolação, mais rápida sua avaliação.
- Polinômios são fáceis de avaliar
- Mas... polinômios de que grau?

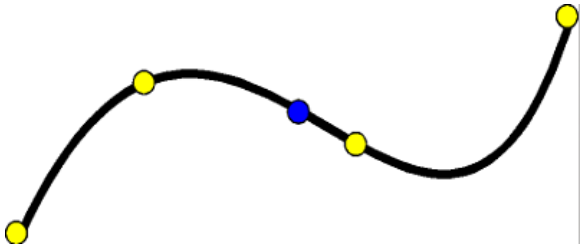
$f(t) = at + b$ $f(t) = at^2 + bt + c$ $f(t) = at^3 + bt^2 + ct + d$



Linear



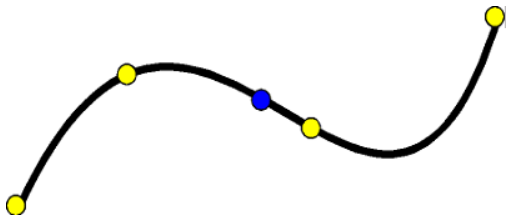
Quadrático



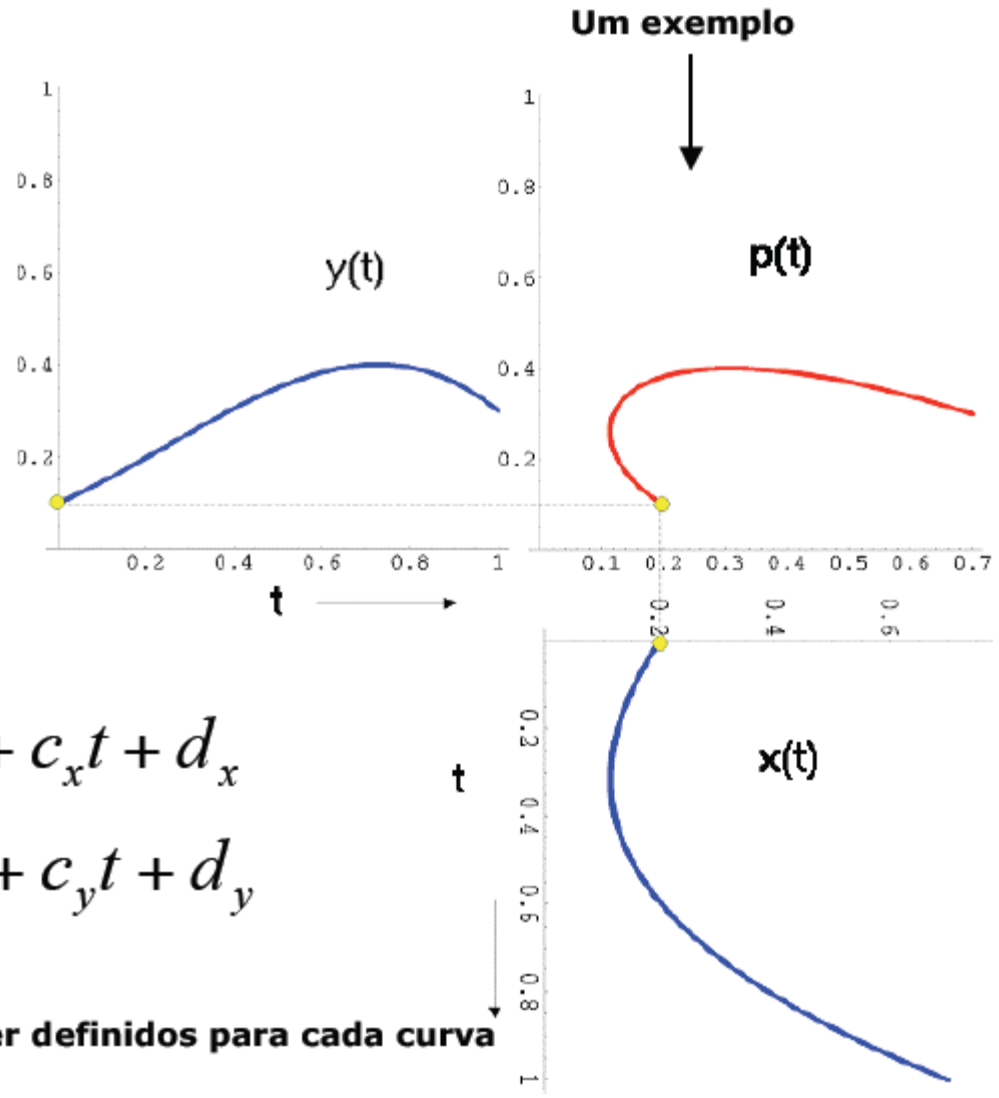
Cúbico

Complexidade

- Impacta na eficiência do algoritmo
- Grau menor que 3: Pouca flexibilidade.
- Grau maior que 3: Maior custo computacional com pouca vantagem prática.
- Por padrão, usa-se cúbicas. $f(t) = at^3 + bt^2 + ct + d$



Curvas Paramétricas Cúbicas 2D



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

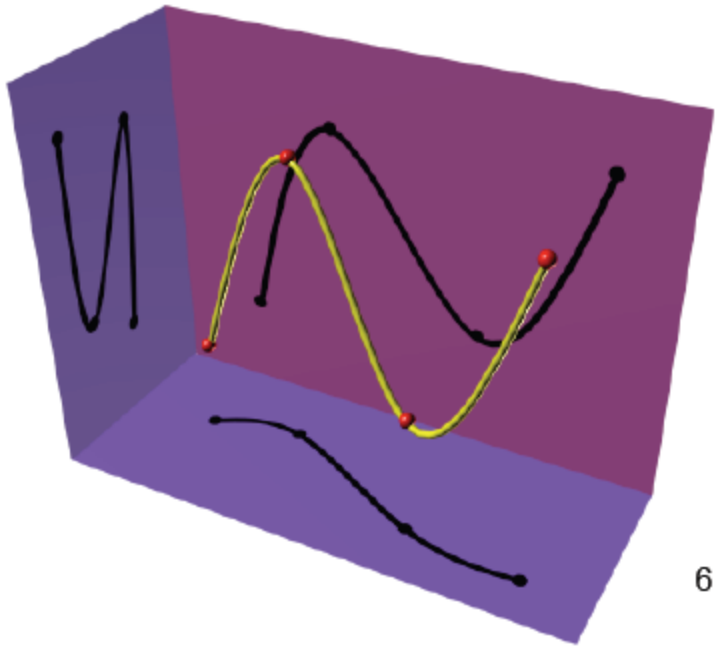
$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

Os valores de ax,bx, etc devem ser definidos para cada curva

Em 3D

$$\left.\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \\ z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z \end{aligned}\right\}$$

Em 3D 12 valores são necessários



Continuidade

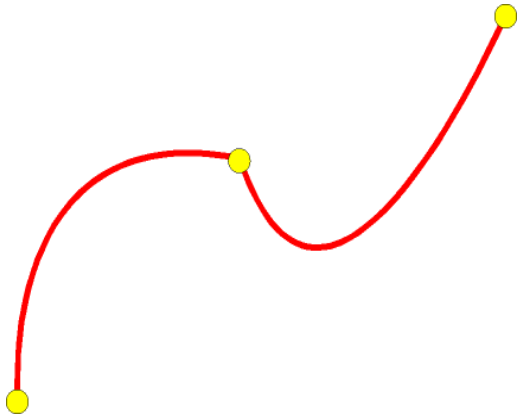
- Questão fundamental
- Métodos adequados:
  - Hermite
  - Catmull-Rom
  - Blending de parábolas
  - Curvas Bézier
  - ...

Continuidade

- Para assegurar a continuidade entre segmentos de curva, definem-se restrições adicionais de continuidade

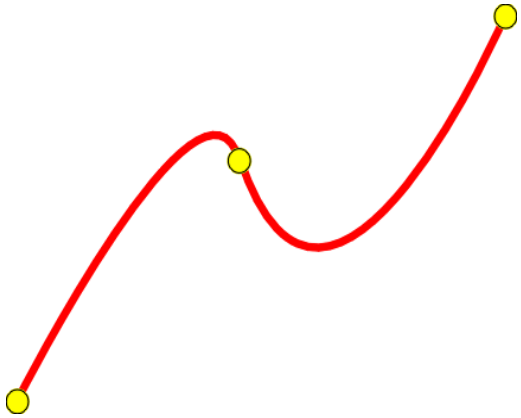
- 2 tipos de continuidade:
  - *Continuidade paramétrica*, denotada por C^n onde n = grau de continuidade
  - *Continuidade geométrica*, denotada por G^n

Continuidade



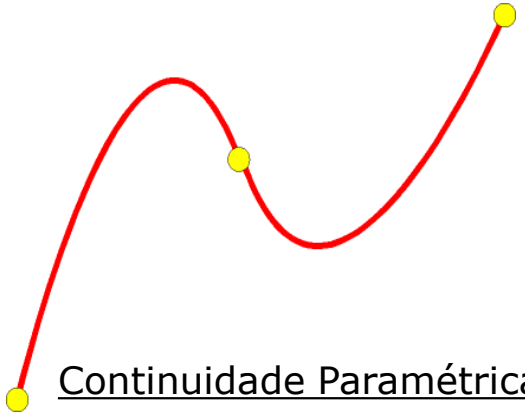
Continuidade Geométrica G0
Continuidade Paramétrica C0

Dois segmentos se encontram em um ponto



Continuidade Geométrica G1

Direção das tangentes dos segmentos são iguais no ponto de junção



Continuidade Paramétrica C1

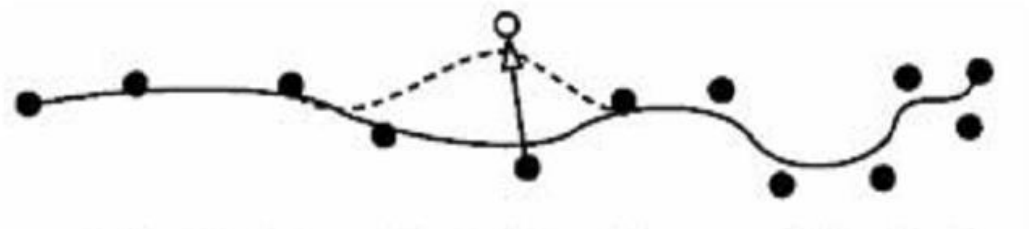
Direção e **magnitude** das tangentes dos segmentos são iguais no ponto de junção

Continuidade Paramétrica C2

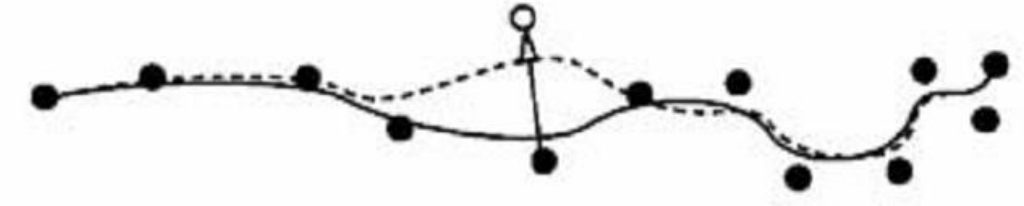
Segunda derivada é igual, ou seja, a mesma aceleração

Controle

- Local



- Global

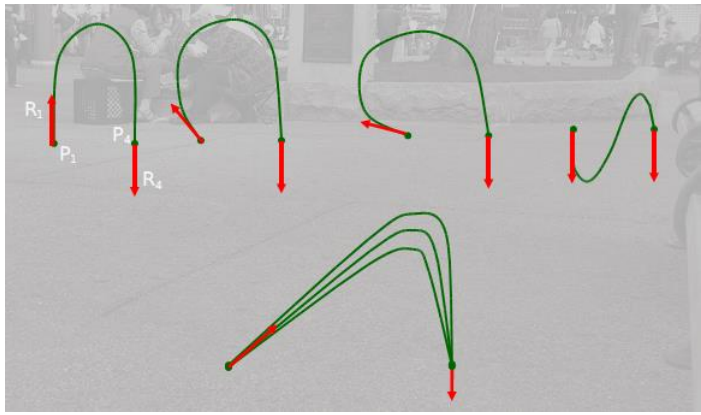


Métodos para representar curvas

- Hermite
- Bézier
- B-Spline

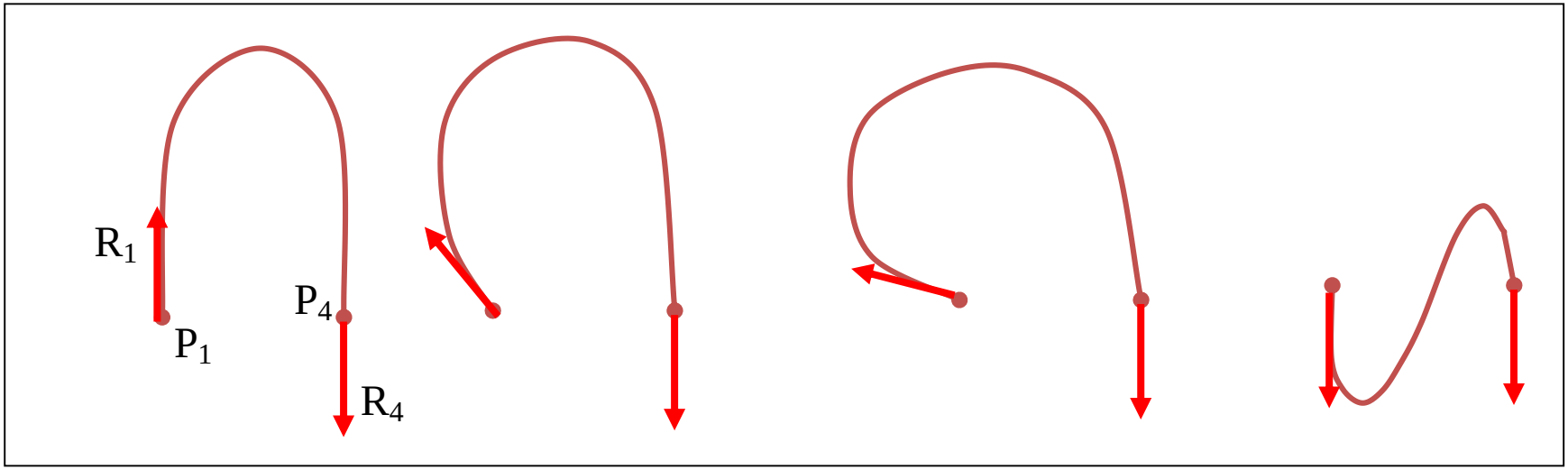
Hermite

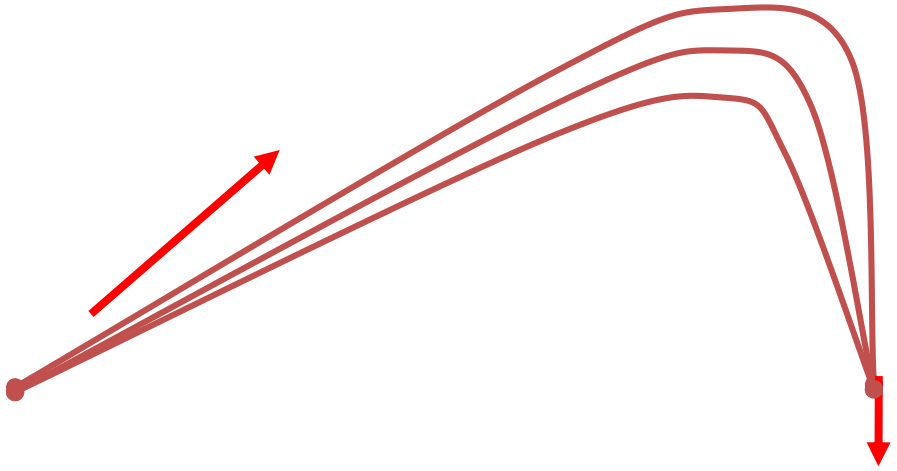
- Definida a partir de restrições no ponto inicial e no ponto final.
  - Os pontos propriamente ditos: P0 e P1
  - Vetores tangentes nestes pontos: m0 e m1



$$\boldsymbol{p}(t) = (2t^3 - 3t^2 + 1)\boldsymbol{p}_0 + (t^3 - 2t^2 + t)\boldsymbol{m}_0 + (-2t^3 + 3t^2)\boldsymbol{p}_1 + (t^3 - t^2)\boldsymbol{m}_1$$

Hermite





Hermite

- Vantagens
  - Bem fácil de implementar ☺
  - Adequada para aplicações onde seja útil definir a curva em função dos vetores tangentes
  - Passa nos pontos de controle (interpolação)

- Desvantagens
  - Não garante, de forma automática, a continuidade entre os segmentos de curva
    - É necessário os vetores tangentes terem a mesma direção e sentido
  - Não permite controle local
    - Alteração de um ponto de controle altera toda a curva

```
float Geometry[4][3] = {
                { 10,10,0 },   //
                {-10,5,-2 },   //
                { 5,-5,0  },   //
                { 5,10,0  }    //
};

unsigned int LOD=20;
```

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P(0) \\ P(1) \\ P'(0) \\ P'(1) \end{bmatrix}$$

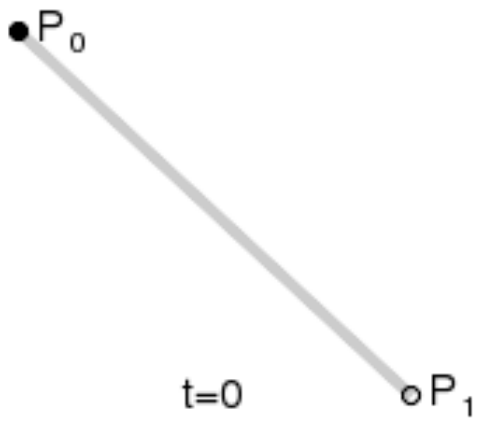
```
// use the parametric time value 0 to 1
for(int i=0;i!=LOD;++i) {

float t = (float)i/(LOD-1);

// calculate blending functions
float b0 =  2*t*t*t - 3*t*t + 1;
float b1 = -2*t*t*t + 3*t*t;
float b2 = t*t*t - 2*t*t + t;
float b3 = t*t*t - t*t;

// calculate the x,y and z of the curve point
float x = b0*Geometry[0][0] +
                        b1*Geometry[1][0] +
                        b2*Geometry[2][0] +
                        b3*Geometry[3][0] ;

float y = b0*Geometry[0][1] +
                        b1*Geometry[1][1] +
                        b2*Geometry[2][1] +
                        b3*Geometry[3][1] ;

float z = b0*Geometry[0][2] +
                        b1*Geometry[1][2] +
                        b2*Geometry[2][2] +
                        b3*Geometry[3][2] ;

// specify the point
glVertex3f( x,y,z );

}
```

Bézier

- Curva polinomial desenvolvida em 1962 por Pierre Bézier.
- Utilizada no projeto de automóveis (Renault).
- Baseada no algoritmo de De Casteljau em 1957.
- Curva de aproximação.
- Controle global.

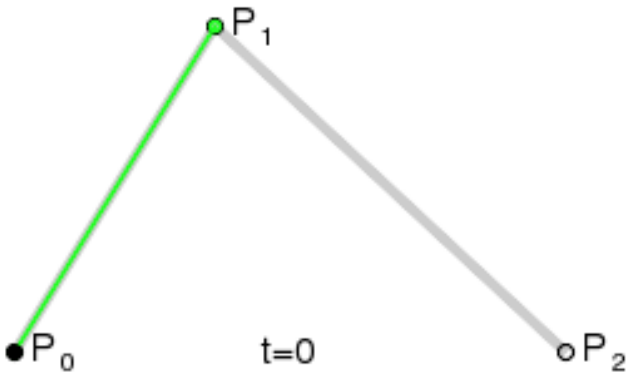
Bézier

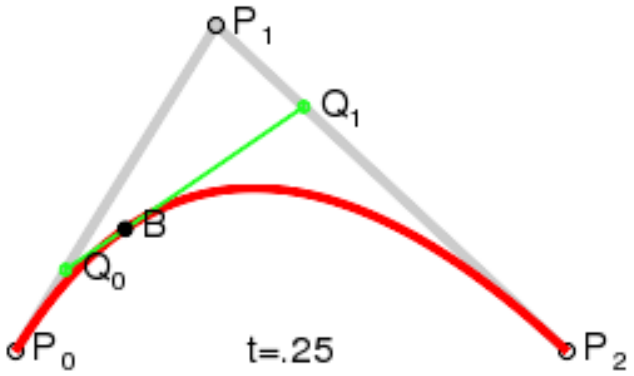
$$P(t) = (1-t) \cdot P_0 + t \cdot P_1$$



Bézier

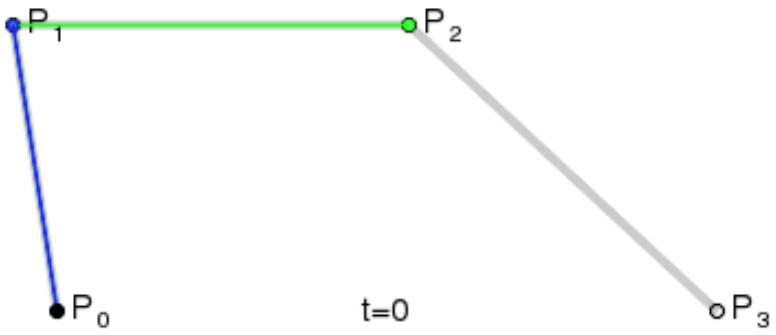
$$P(t) = (1 - t)^2 \mathbf{P}_0 + 2t(1 - t)\mathbf{P}_1 + t^2 \mathbf{P}_2$$

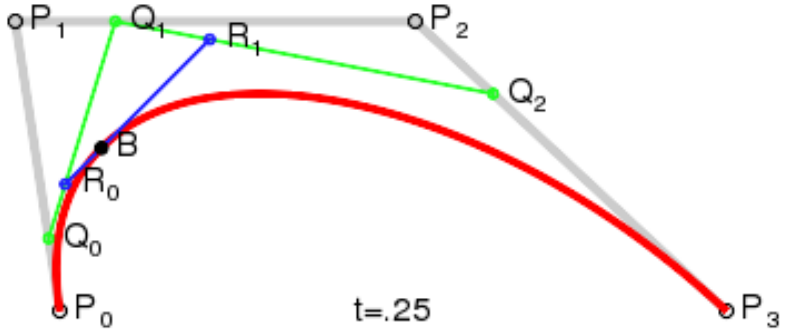




Bézier

$$P(t) = (1-t)^3 \mathbf{P}_0 + 3t(1-t)^2 \mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3 \mathbf{P}_3$$



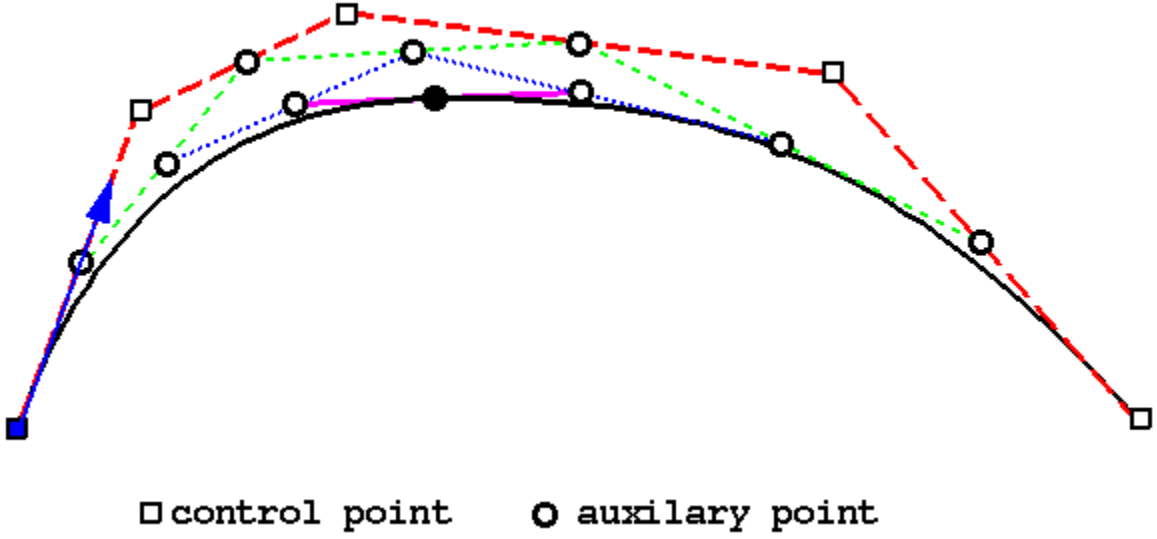


```
float Geometry[4][3] = {
                { 10,10,0 },   //          Point1
                {-10,5,-2 },   //          Point2
                { 5,-5,0  },   //          Tangent1
                { 5,10,0  }    //          Tangent2
};


unsigned int LOD=20;
```

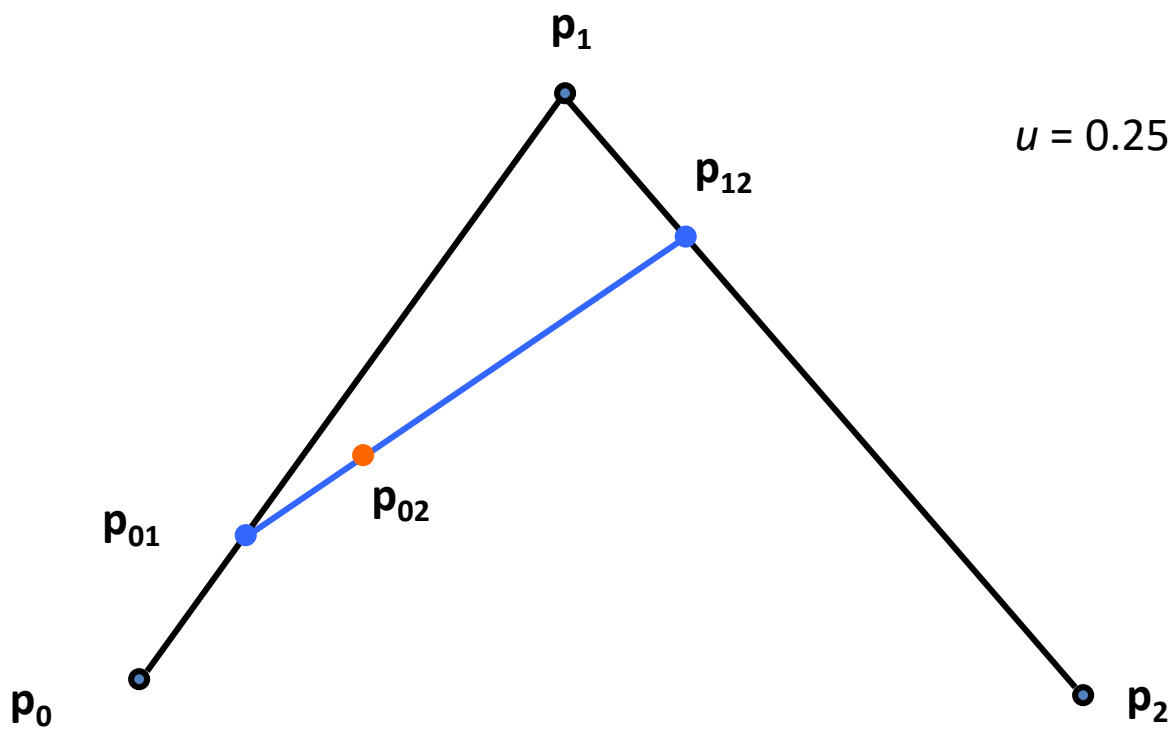
```
// use the parametric time value 0 to 1
                for(int i=0;i!=LOD;++i) {

                        float t = (float)i/(LOD-1);
                        float it = 1.0f-t;

                        // calculate blending functions
                        float b0 = t*t*t;
                        float b1 = 3*t*t*it;
                        float b2 = 3*t*it*it;
                        float b3 =  it*it*it;

                        // calculate the x,y and z of the curve point
                        float x = b0*Geometry[0][0] +
                                                b1*Geometry[1][0] +
                                                b2*Geometry[2][0] +
                                                b3*Geometry[3][0] ;

                        float y = b0*Geometry[0][1] +
                                                b1*Geometry[1][1] +
                                                b2*Geometry[2][1] +
                                                b3*Geometry[3][1] ;

                        float z = b0*Geometry[0][2] +
                                                b1*Geometry[1][2] +
                                                b2*Geometry[2][2] +
                                                b3*Geometry[3][2] ;

                        // specify the point
                        glVertex3f( x,y,z );
}
```

Bézier

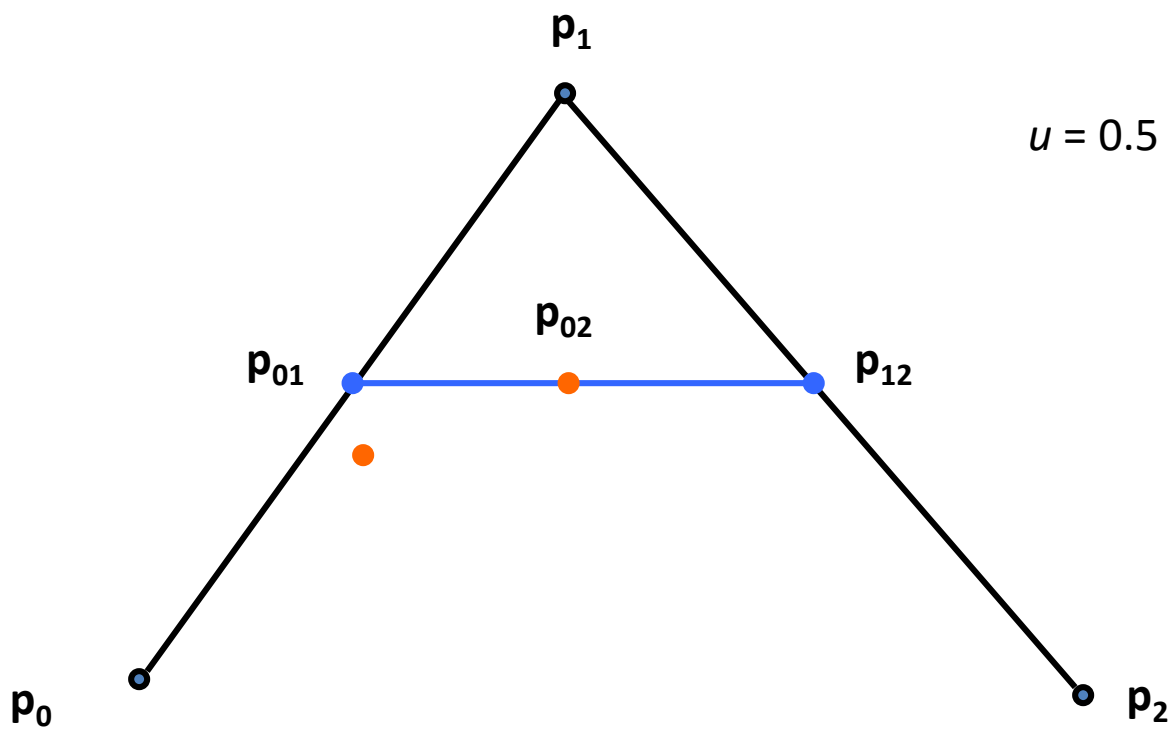
- De Casteljau: algoritmo geométrico para construção de curvas Bézier.



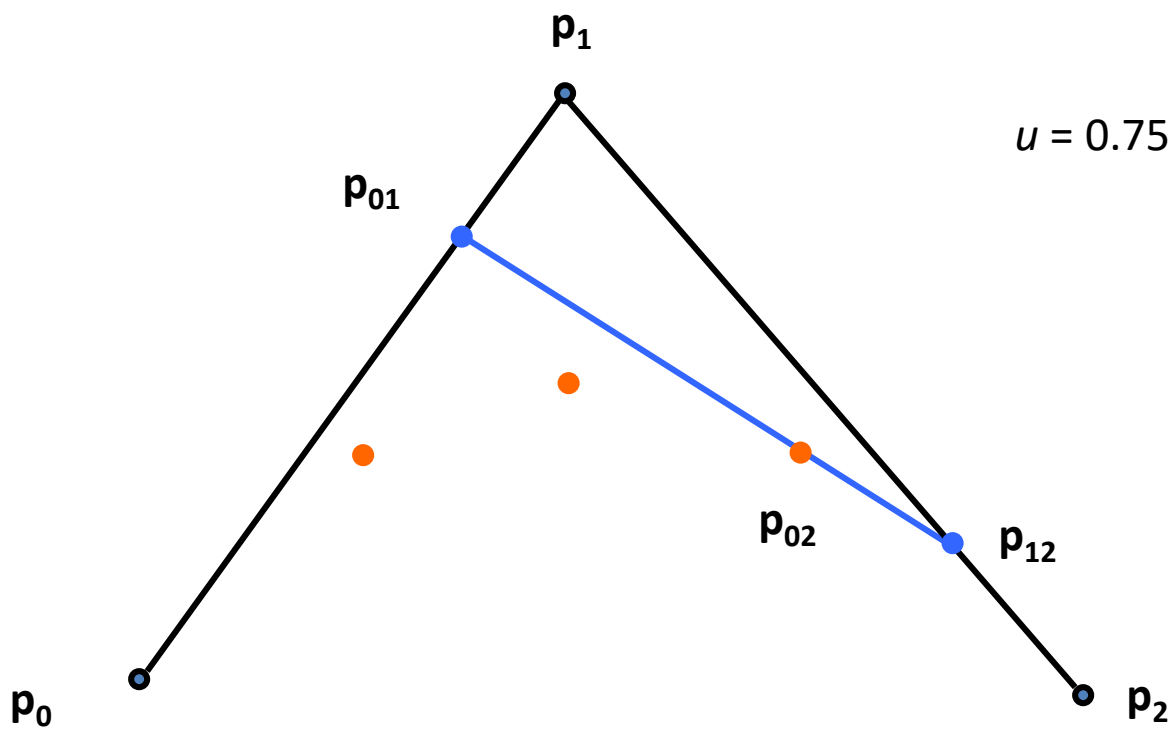
Algoritmo de De Casteljau



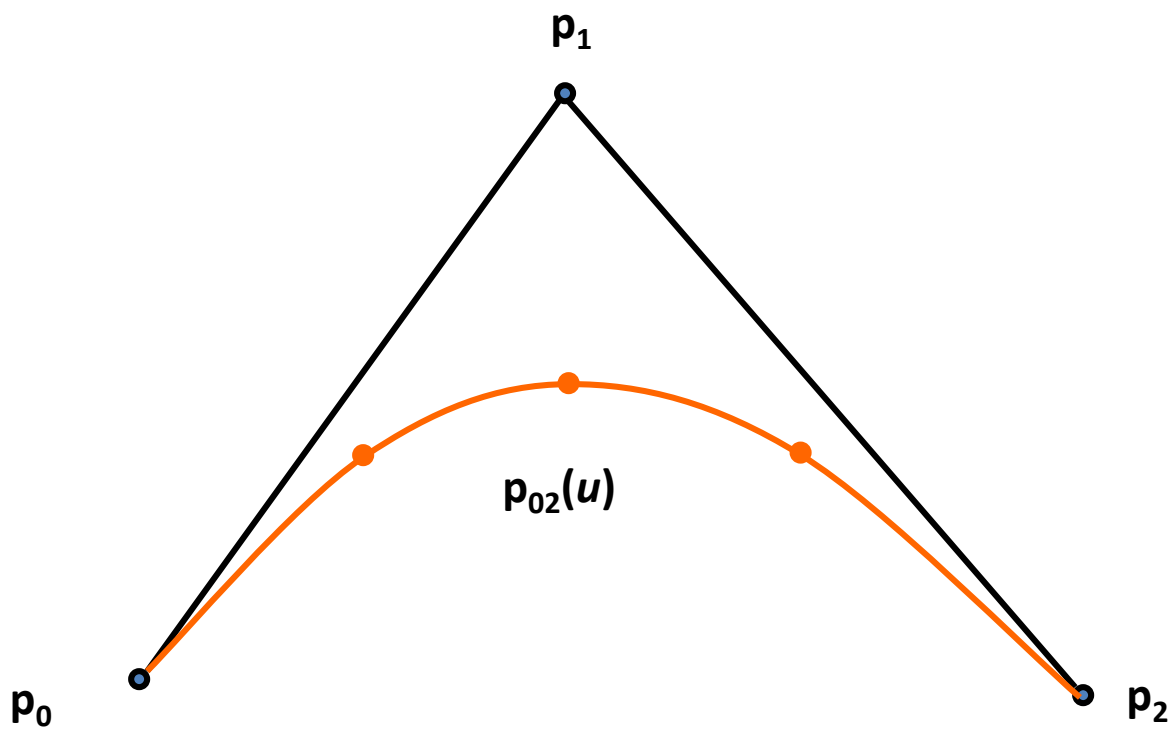
Algoritmo de De Casteljau



Algoritmo de De Casteljau

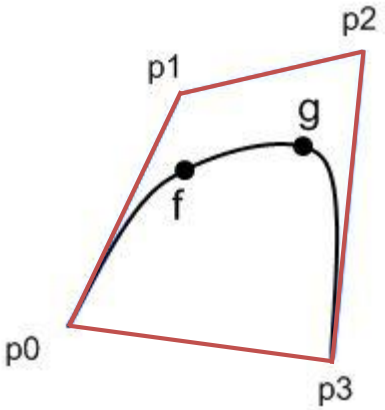


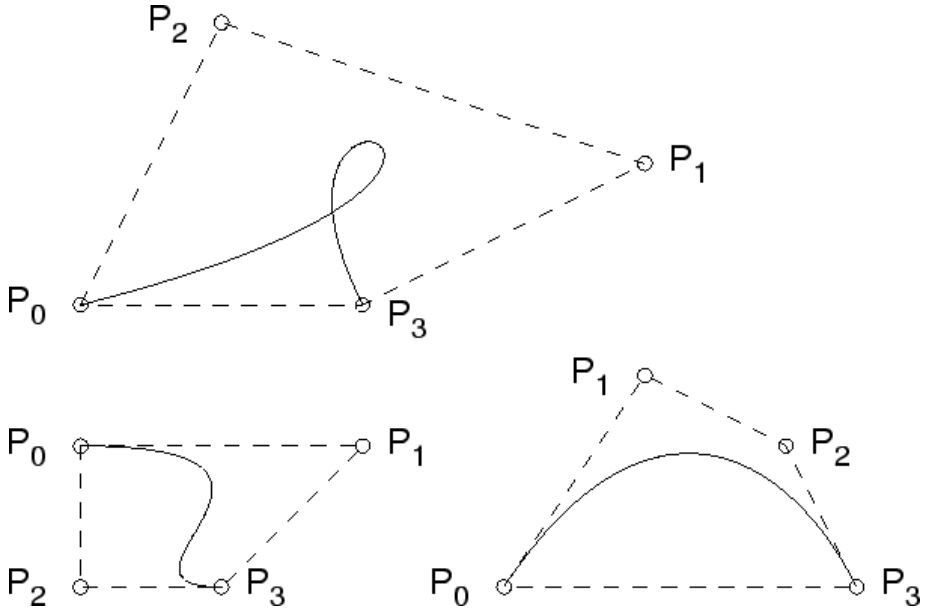
Algoritmo de De Casteljau



Bézier

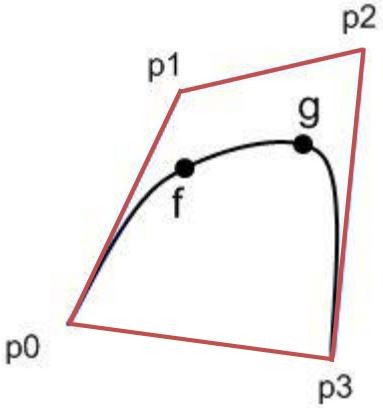
- Propriedade: *Convex Hull*
  - Uma curva de Bézier está completamente dentro do maior polígono convexo, formado pelos pontos de controle.

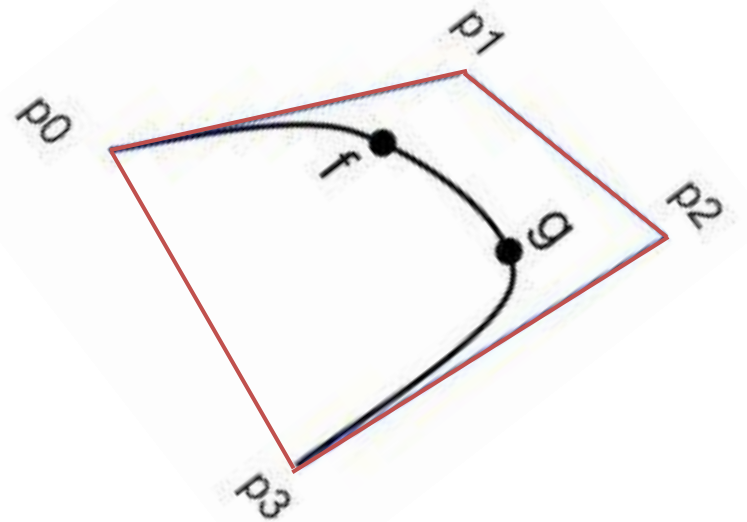




Bézier

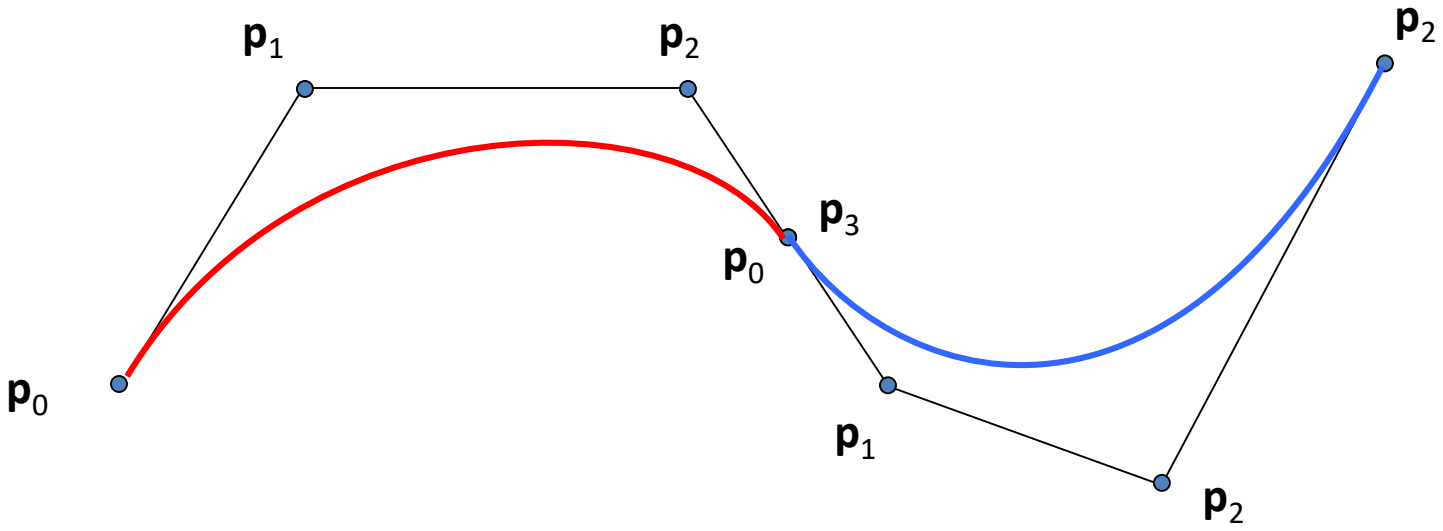
- Transformações
  - Executar as transformações (S,R,T) na curva é equivalente a realizar as transformações nos pontos de controle.





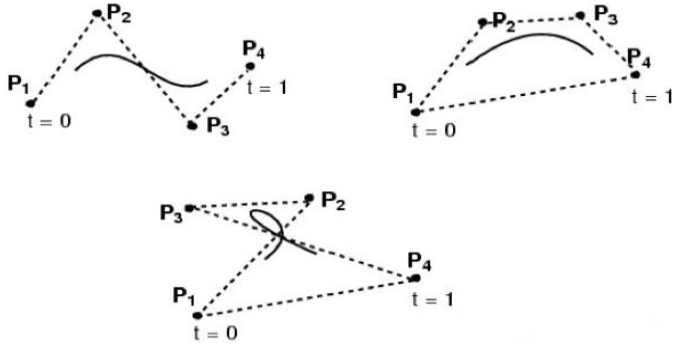
Emendando Curvas Bézier

- Continuidade C^0: Último ponto da primeira = primeiro ponto da segunda
- Continuidade C^1: C^0 e segmento $\mathbf{p}_2\mathbf{p}_3$ da primeira com mesma direção e comprimento que o segmento $\mathbf{p}_0\mathbf{p}_1$ da segunda
- Continuidade C^2: C^1 e + restrições sobre pontos $\mathbf{p}_1$ da primeira e $\mathbf{p}_2$ da segunda



B-Spline

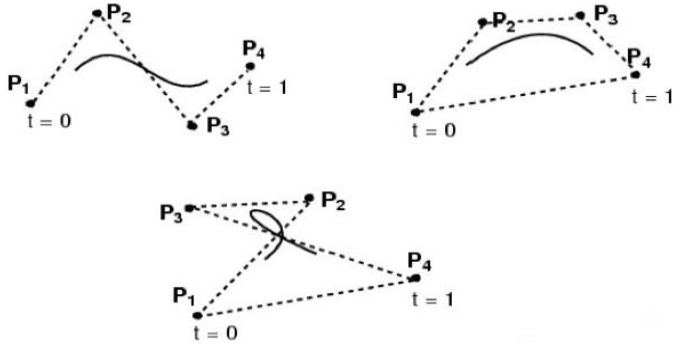
- Definida por quatro pontos de controle (P_1, P_2, P_3, P_4).
- Não passa por nenhum ponto de controle.
  - Curva de aproximação
- Mais suave que as anteriores
  - Mais fácil garantir continuidade paramétrica
- Controle local.



B-Spline

- Cada segmento é definido por 4 pontos, cada ponto influencia 4 segmentos de curva (exceto P0 e Pn)

- Knots são os pontos de junção entre os segmentos de curva



B-Spline

- Representação matricial

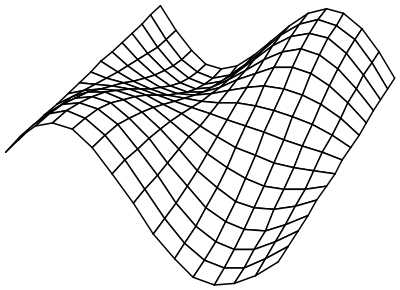
$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_2 \end{bmatrix}$$

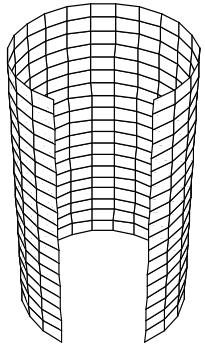
```
float Geometry[4][3] = {
        { 10,10,0 },   //      Point1
        {-10,5,-2 },   //      Point2
        { 5,-5,0  },   //      Tangent1
        { 5,10,0  }    //      Tangent2
};

unsigned int LOD=20;
```

```
// use the parametric time value 0 to 1
for(int i=0;i!=LOD;++i) {

    float t = (float)i/(LOD-1);
    float it = 1.0f-t;

    // calculate blending functions
    float b0 = it*it*it/6.0f;
    float b1 = (3*t*t*t - 6*t*t +4)/6.0f;
    float b2 = (-3*t*t*t +3*t*t + 3*t + 1)/6.0f;
    float b3 =  t*t*t/6.0f;

    // calculate the x,y and z of the curve point
    float x = b0*Geometry[0][0] +
                    b1*Geometry[1][0] +
                    b2*Geometry[2][0] +
                    b3*Geometry[3][0] ;

    float y = b0*Geometry[0][1] +
                    b1*Geometry[1][1] +
                    b2*Geometry[2][1] +
                    b3*Geometry[3][1] ;

    float z = b0*Geometry[0][2] +
                    b1*Geometry[1][2] +
                    b2*Geometry[2][2] +
                    b3*Geometry[3][2] ;

    // specify the point
    glVertex3f( x,y,z );
}
```

Superfícies Paramétricas

- Ideia de *multiplicação* de 2 curvas
- A informação geométrica que define uma curva passa a ser ela própria uma função de uma variável paramétrica





Superfícies Paramétricas

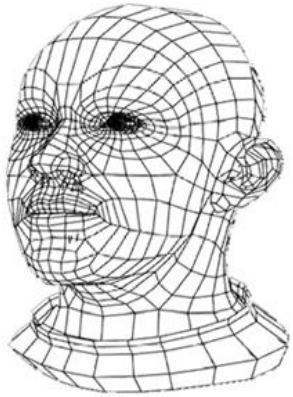
- A forma geral de uma superfície 3D na sua representação paramétrica é:

$$f(u,v) = (f_x(u,v), f_y(u,v), f_z(u,v))$$

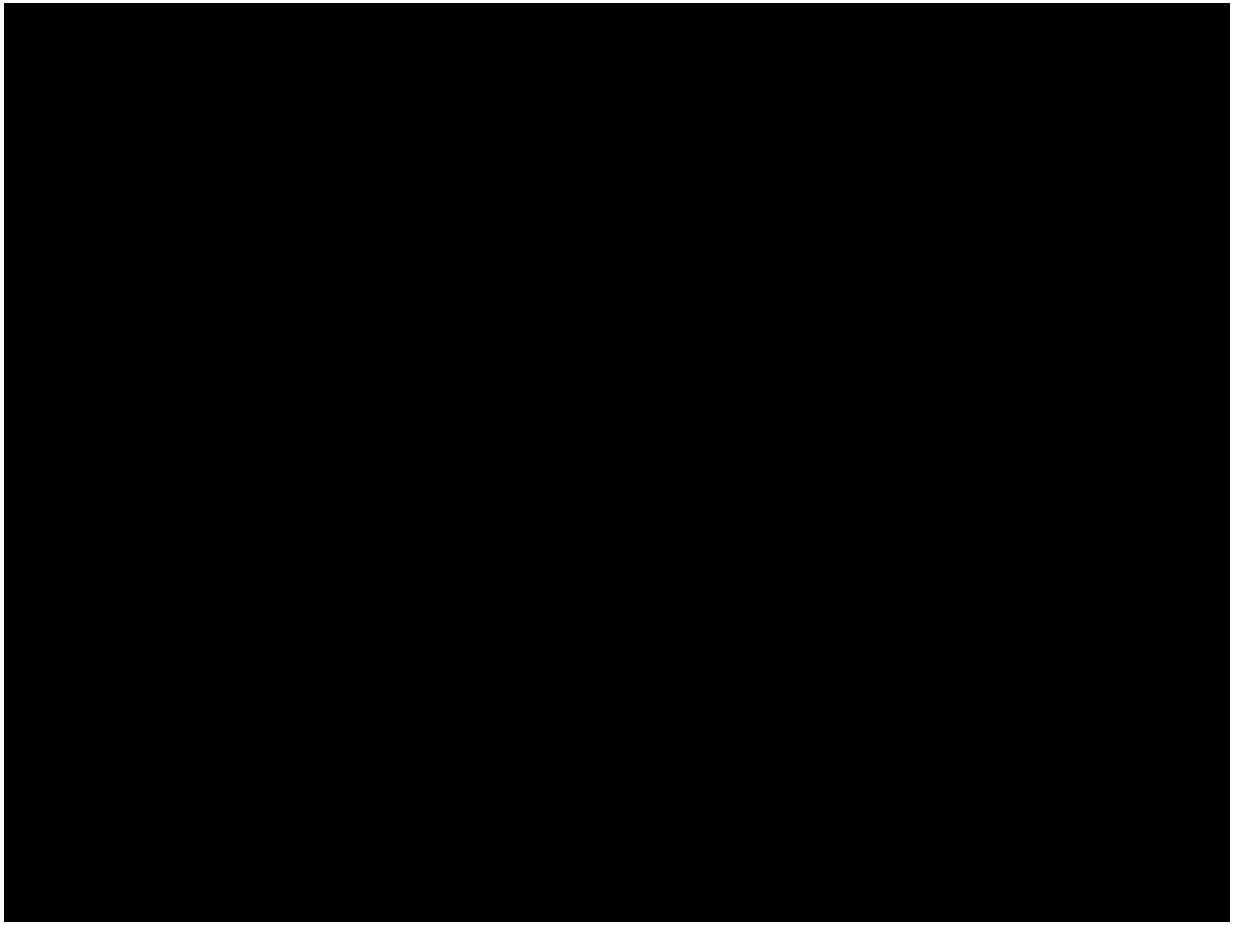
Malhas de Polígonos

- Coleção de arestas, vértices e polígonos conectados

- Diferentes formas de armazenar a estrutura do polígono
  - Vertex list + faces list
  - Vertex list + edges list + faces list



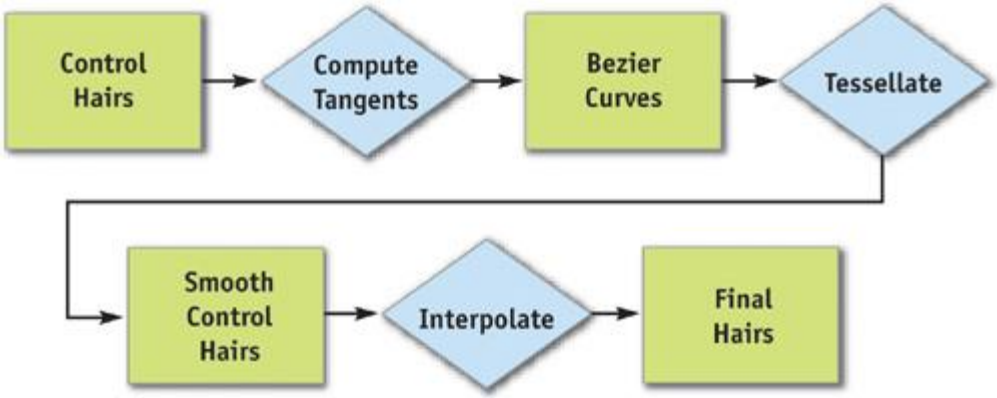
Animação de Cabelos

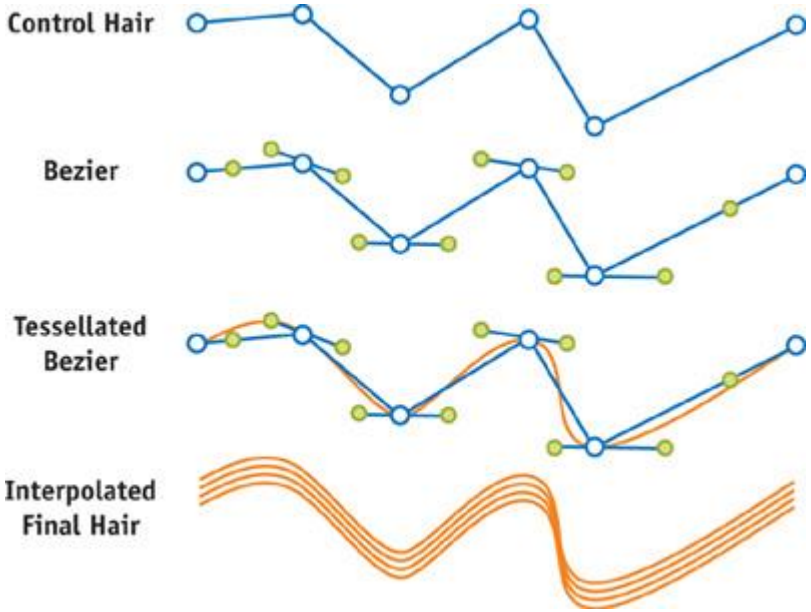


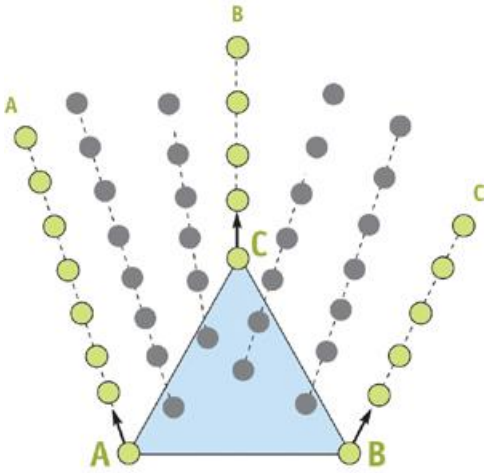
http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter23.html

Animação de Cabelos

- Modelo



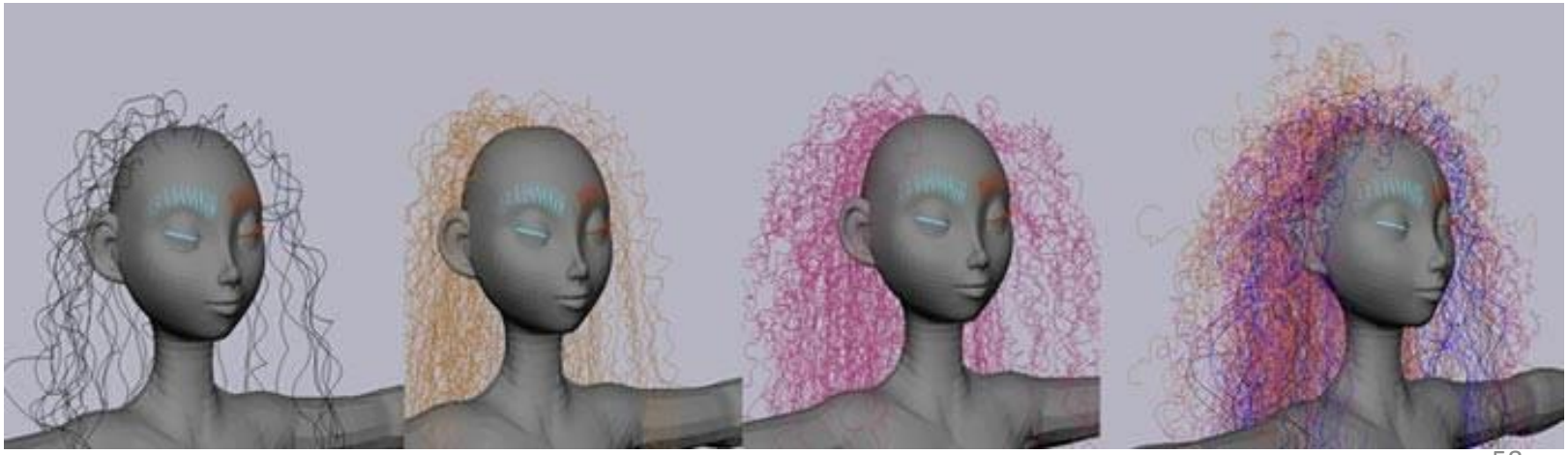





(a)


(b)

Mais exemplos…



Brave (Disney)

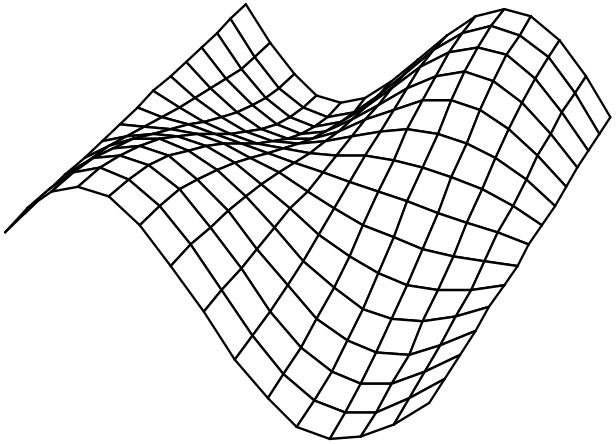
Desenvolveram um simulador chamado Taz. O cabelo foi modelado usando sistema Massa-mola. No entanto, os cachos são bem formados e deveriam ser rígidos, mas com movimento, sendo contraditório a resistência das molas. Deveriam voar ao vento, mas não muito. Além disto, tinham os problemas da intersecção e colisão que tornaram o projeto desafiador.

O simulador trata grupo de cabelos usando multi-threads. Os cabelos-chave são B-splines que foram usadas para ihnterpolar o resto dos cabelos.

Merida had 1500 hand placed curves which interpolate to some 111,000 curves at final render. Merida's hair was simulated at about 20 to 30 seconds a frame.

Brave (Disney)

http://www.thepixarpodcast.com/category/all-episodes
http://www.wired.com/underwire/2012/06/pl_bravehairtech/



Curvas e Superfícies

Profª Soraia Raupp Musse