

Courbes et surfaces

Représentations de courbes et de surfaces

- Représentation explicite/implicite
 - ▣ Utilisation de surfaces algébriques: les quadriques
 - ▣ Approximation polygonale
 - ▣ Contrôle plus complexe pour la modélisation de grande précision
- Représentation paramétrique
 - ▣ Les splines cubiques
 - ▣ Courbes et surfaces de Bézier
 - ▣ Courbes et surfaces B-Splinaires
 - ▣ Courbes NURBS

Représentation explicite/implicite

● Représentation explicite

- Forme explicite d'une courbe en 2D: $y = f(x)$
- Forme explicite d'une surface: $z = f(x, y)$
- Il n'existe pas toujours une représentation explicite pour toutes les courbes et surfaces d'intérêt

● Représentation implicite

- Forme implicite d'une courbe en 2D: $f(x, y) = 0$
- Forme implicite d'une surface en 3D: $f(x, y, z) = 0$
 - ▶ $ax + by + cz + d = 0$ est l'équation d'un plan
 - ▶ $x^2 + y^2 + z^2 - r^2 = 0$ est l'équation d'une sphère de rayon r centrée à l'origine
- Il peut être difficile de représenter une courbe en 3D sous une forme implicite
 - ▶ Intersection de deux surfaces: $f(x, y, z) = 0$ et $g(x, y, z) = 0$

Les surfaces quadriques

- ➊ Équation algébrique implicite d'une quadrique

$$Q(x, y, z) = a_1x^2 + a_2y^2 + a_3z^2 + a_4xy + a_5xz + a_6yz + a_7x + a_8y + a_9z + a_{10}$$

- ➋ Manipulation des quadriques dans OpenGL

- ➌ Créer un objet de type quadrique à l'aide de `gluNewQuadric()`

- ➌ Spécifier les attributs de l'objet créé à l'aide de:

- ▶ `gluQuadricOrientation()`, `gluQuadricDrawStyle()`, `gluQuadricNormals()`, `gluQuadricTexture()`

- ➌ Définir une fonction de gestion des erreurs à l'aide de `gluQuadricCallback()`

- ➌ Spécifier le type de quadrique désiré à l'aide des primitives:

- ▶ `gluSphere()`, `gluCylinder()`, `gluDisk()` ou `gluPartialDisk()`

- ➌ Les normales et les coordonnées de texture peuvent être générées automatiquement

- ➌ OpenGL implémente ces objets à l'aide d'une approximation polygonale

Approximation polygonale des surfaces

- ➊ Augmenter le nombre de polygones (maillage plus fin) pour mieux approximer la surface:
 - ▣ diminue l'erreur de représentation (+)
 - ▣ matériel déjà disponible pour des polygones (+)
 - ▣ augmente l'espace mémoire (-)
 - ▣ augmente le temps requis pour le rendu (-)
 - ▣ augmente le nombre de points à manipuler pour modifier la surface (-)

Historique des courbes et surfaces paramétriques polynomiales

- Dans des applications réelles:
 - Une trajectoire est définie par une courbe
 - Un objet réel est défini comme une surface lisse
- Solution paramétrique polynomiale:
 - Courbe (cubique): $x(t)$, $y(t)$, $z(t)$
 - Surface ou facette (bic cubique): $x(s, t)$, $y(s, t)$, $z(s, t)$
- Splines cubiques développées chez GM dans les années 50.
- Courbes de Bézier développées chez Renault et Peugeot dans les années 60 et 70.
- Courbes B-Splinaires et NURBS développées chez Boeing dans les années 70 et 80.

Courbes d'interpolation et d'approximation

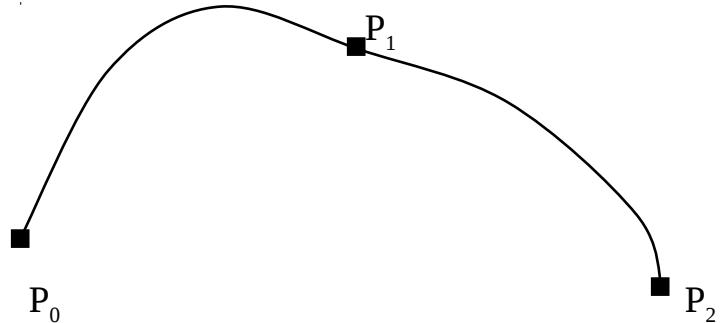
Deux catégories:

- Les courbes d'interpolation (collocation) passent, par définition, par les points de contrôle.
exemples: splines linéaires (polylignes), splines cubiques.
- Les courbes d'approximation peuvent passer par certains points de contrôle, mais en général, ne passent pas par tous les points.
exemples: courbes de Bézier, B-Splines, NURBS.

Courbes splinaires

Notion de spline

- ➊ Courbe polynomiale par morceaux
- ➋ Pour chaque morceau, on définit :
 - ▣ Ordre ou degré des polynômes
(ordre = degré + 1)
 - ▣ Plage du paramètre
 - ▣ Ordre de continuité aux jonctions



- ➌ Chaque morceau peut s'écrire sous la forme:

$$f(t) = \sum_i N_i G_i$$

avec:

- t le paramètre de la courbe,
- $f(t)$ la fonction polynomiale,
- G_i la $i^{\text{ème}}$ information géométrique
- $N_i(t)$ la $i^{\text{ème}}$ fonction de base ou de pondération

Fonction polynomiale par intervalles

- Une **fondtion spline** de degré m et d'ordre de continuité r est une fonction polynomiale par intervalles de degré m , définie par exemple sur n intervalles de la manière suivante:

$$f(t) = f_i(t) \quad \text{si} \quad t_i \leq t \leq t_{i+1} \quad \text{pour } i = 0, \dots, n-1$$

et satisfaisant aux conditions suivantes:

$$f_{i-1}^{(j)}(t_i) = f_i^{(j)}(t_i) \quad \text{si } i = 1, 2, \dots, n-1 \quad \text{et } j = 0, 1, \dots, r-1$$

- La continuité de la $j^{\text{ième}}$ dérivée de f aux valeurs nodales
⇒ la continuité de f et de ses $r-1$ premières dérivées aux frontières des intervalles.

Splines linéaires

- Représentation d'un segment de droite:

- Forme explicite:

$$y = mx + b$$

- Forme paramétrique:

$$x = (1 - t) \cdot P_{1x} + t \cdot P_{2x}$$

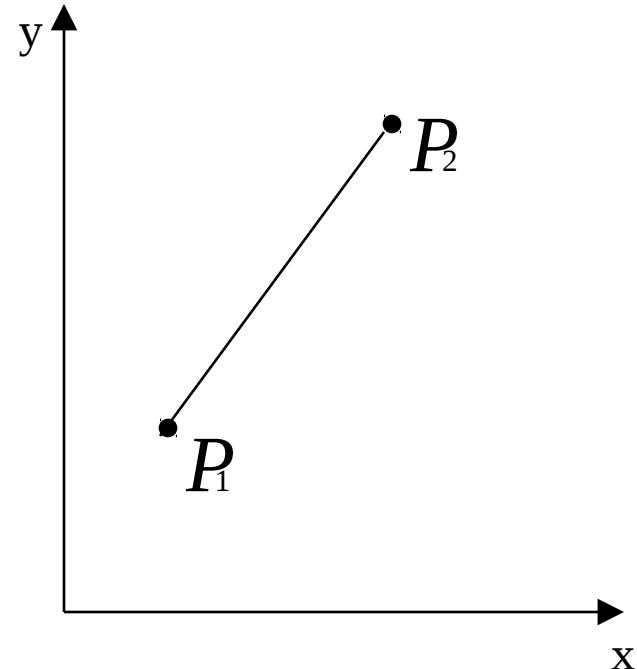
$$y = (1 - t) \cdot P_{1y} + t \cdot P_{2y}$$

- Fonctions de base:

$$N_1 = (1 - t) \quad N_2 = t \quad 0 \leq t \leq 1$$

- Information géométrique:

$$P_1 \text{ et } P_2$$

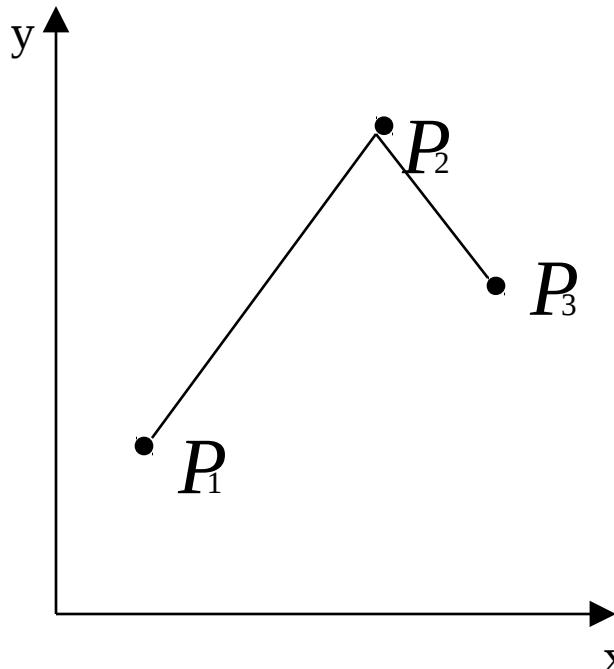


Splines linéaires

- Représentation d'une série de segments de droite en forme paramétrique (degré 1, ordre 2):

$$P(t) = (1-t) \cdot P_1 + t \cdot P_2 + 0 \cdot P_3 \quad \text{si } 0 \leq t \leq 1$$

$$P(t) = 0 \cdot P_1 + (1-(t-1)) \cdot P_2 + (t-1) \cdot P_3 \quad \text{si } 1 \leq t \leq 2$$



Splines linéaires

- ➊ La paramétrisation de la courbe influence la qualité de la représentation de la courbe:

- ▣ Si l'on fixe un Δt constant, la paramétrisation influence la «vitesse de parcours» de la courbe

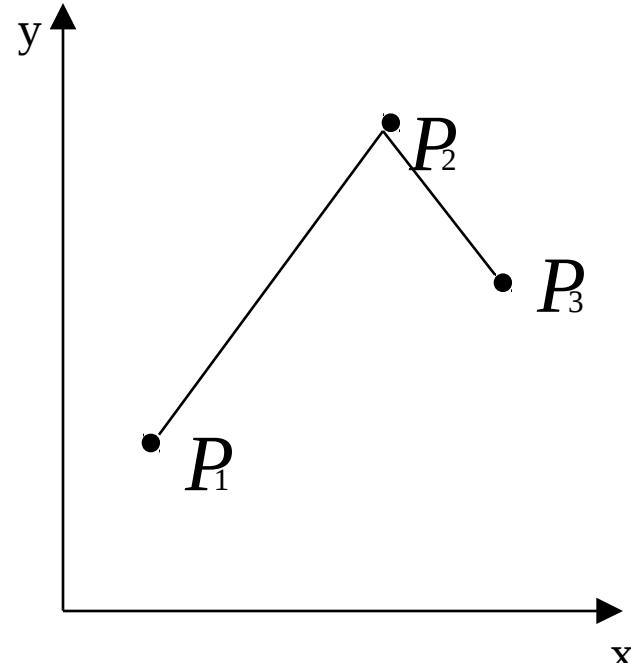
- ➋ Pour obtenir une «vitesse» constante, on reparamétrise souvent la courbe en fonction de la longueur de chaque segment:

- ▣ Pour le segment 1 (de longueur L_1):

$$0 \leq t \leq \frac{L_1}{L_{total}}$$

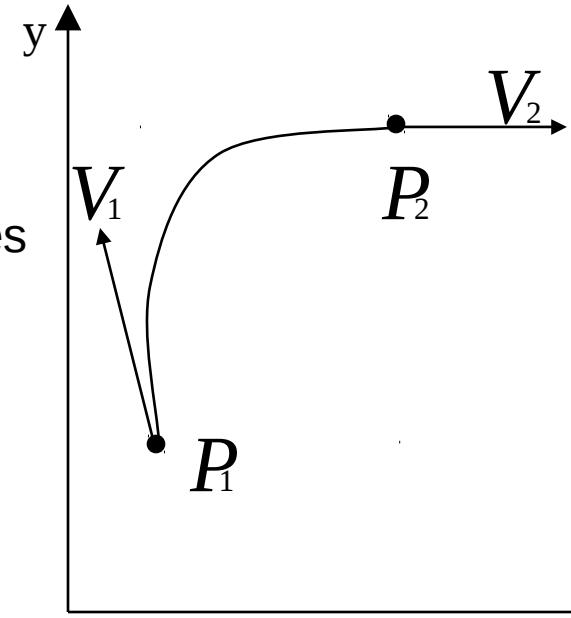
- ▣ Pour le segment 2 (de longueur L_2):

$$\frac{L_1}{L_{total}} \leq t \leq 1$$



Splines cubiques

- ➊ Les splines cubiques sont des courbes polynomiales par morceaux, dont chaque segment est représenté par un polynôme de degré 3 (ordre 4).
- ➋ Une façon naturelle de représenter un segment de spline cubique est d'utiliser comme information géométrique :
 - ▣ la position des points aux deux extrémités
 - ▣ le vecteur tangent à chaque extrémité



Splines cubiques

- Pour un segment de spline cubique,

$$f(t) = \sum_i N_i G_i$$

devient:

$$P(t) = [H_1(t), H_2(t), H_3(t), H_4(t)] \begin{bmatrix} P_1 \\ P_2 \\ V_1 \\ V_2 \end{bmatrix}$$

avec:

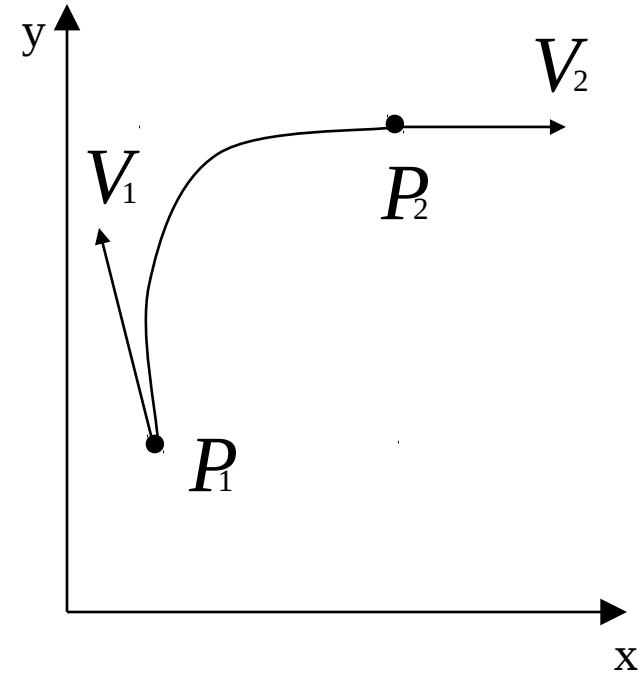
$$H_1(t) = 2t^3 - 3t^2 + 1$$

$$H_2(t) = -2t^3 + 3t^2$$

$$H_3(t) = t^3 - 2t^2 + t$$

$$H_4(t) = t^3 - t^2$$

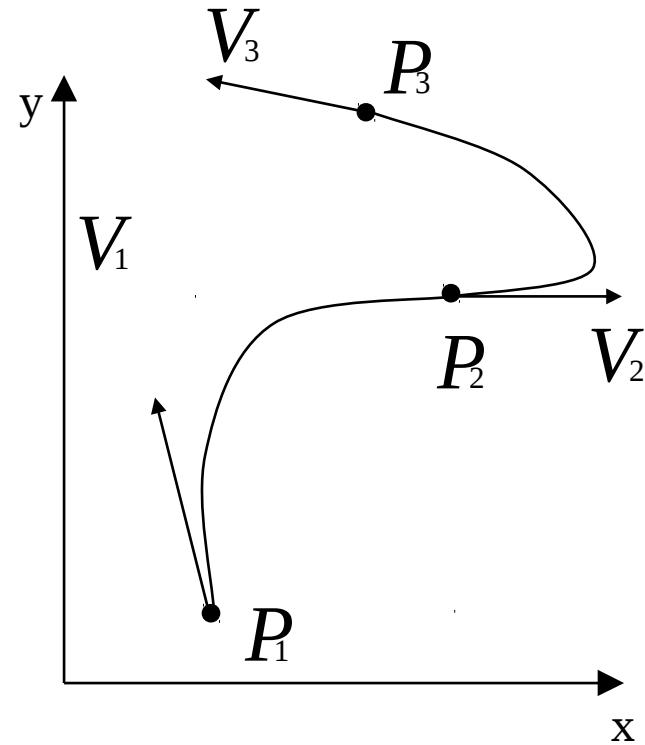
Les $H_i(t)$ sont les polynômes d'Hermite de degré 3.



Splines cubiques

- ➊ Une courbe est obtenue en assemblant plusieurs segments:

- ▣ Les vecteurs tangents associés aux nœuds internes peuvent être calculés automatiquement en résolvant un système linéaire.
 - ▣ Pour construire le système linéaire, on impose la continuité des tangentes et des courbures.
 - ▣ Il faut imposer deux conditions limites en plus de la position des points pour obtenir un système ayant autant d'équations que d'inconnues.

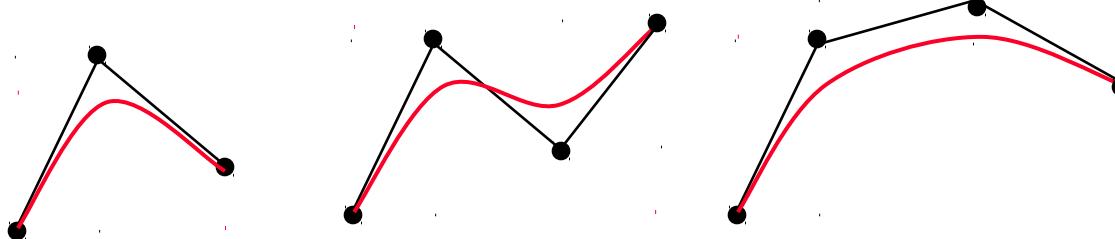


Courbes de Bézier

Courbes de Bézier

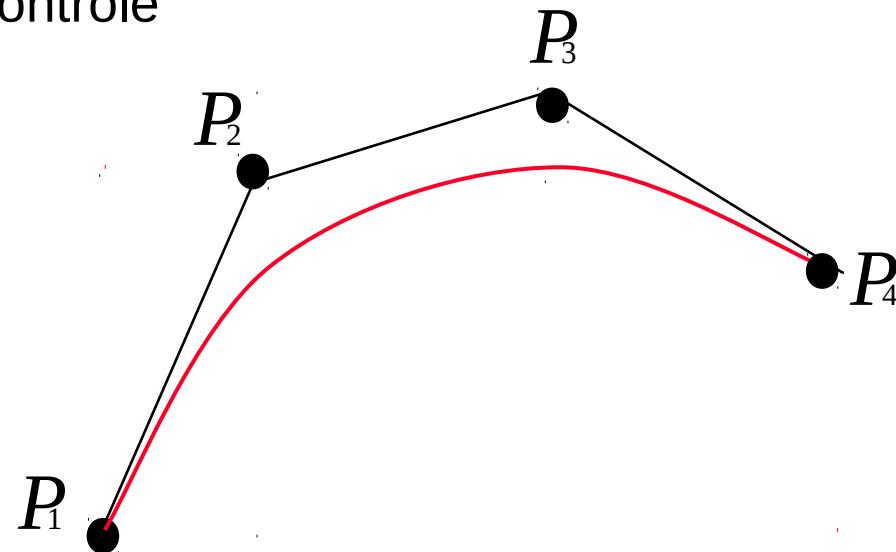
Caractéristiques

- le premier et le dernier nœud sont colloqués,
- la courbe est tangente au premier segment et au dernier du polygone de contrôle,
- l'information pour la représentation de la courbe est constituée de points seulement et ne comprend pas les vecteurs tangents de façon explicite.
- Le degré des polynômes est toujours le nombre de points (n) moins 1, (L'ordre de la courbe est n).



Courbes de Bézier

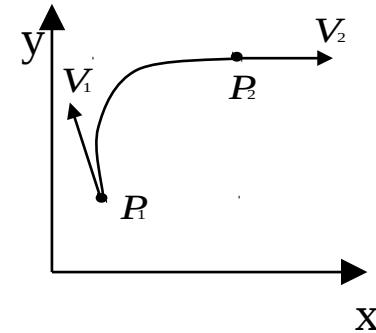
- Idée:
 - Remplacer les vecteurs tangents par des points de contrôle internes.
 - L'ensemble ordonné des points de contrôle forment le polygone de contrôle



Courbes de Bézier

- ➊ Développement mathématique à partir des formes d'Hermite (pour 4 points):

$$P(t) = [H_1(t), H_2(t), H_3(t), H_4(t)] \begin{bmatrix} P_1 \\ P_2 \\ V_1 \\ V_2 \end{bmatrix}$$

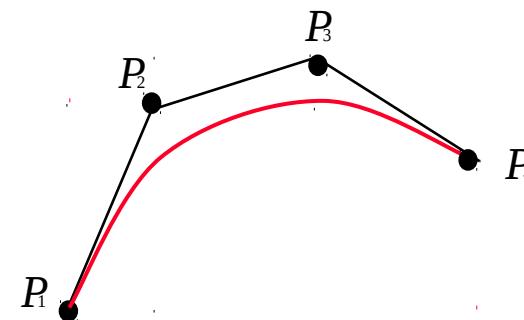


- ➋ On remplace les V par

$\rightarrow \quad \rightarrow \quad \rightarrow$

$$V_1 \propto P_2 - P_1 = K(P_2 - P_1)$$

$$V_2 \propto P_4 - P_3 = K(P_4 - P_3)$$



Courbes de Bézier

Ce qui donne:

$$\vec{P} = (H_1 \ H_2 \ H_3 \ H_4) \begin{bmatrix} P_1 \\ P_4 \\ K(P_2 - P_1) \\ K(P_4 - P_3) \end{bmatrix}$$

où la notation

$$P_1 \leftarrow P(0)$$

$$P_4 \leftarrow P(1)$$

où P_2 et P_3 sont deux nouveaux points complétant le polygone de contrôle

Courbes de Bézier

- En explicitant on trouve:

$$\begin{aligned}P(t) = & \left[(2t^3 - 3t^2 + 1) - K(t^3 - 2t^2 + t) \right] P_1 \\& + Kt(t-1)^2 P_2 - Kt^2(t-1) P_3 \\& + \left[(-2t^3 + 3t^2) - K(t^3 - t^2) \right] P_4\end{aligned}$$

Courbes de Bézier

- Symétrie entre t et $(t - 1)$ dans les coefficients de P_2 et P_3 que l'on peut étendre à P_1 et P_4 par un choix judicieux de K égal à 3 alors,

$$\vec{P}(t) = \begin{bmatrix} (1-t)^3, 3t(1-t)^2, 3t^2(1-t), t^3 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$
$$= [J_{0,3}, J_{1,3}, J_{2,3}, J_{3,3}] \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

Courbes de Bézier

- Pour 3 points

$$\vec{P}(t) = [2t^3 - 3t^2 + 1, -2t^3 + 3t^2, t^3 - 2t^2 + t, t^3 - t^2] \begin{bmatrix} P_1 \\ P_3 \\ K(P_2 - P_1) \\ K(P_3 - P_2) \end{bmatrix}$$

- En prenant $K=2$, on obtient

$$\vec{P}(t) = [(1-t)^2, 2t(1-t), t^2] \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$= [J_{0,2}, J_{1,2}, J_{2,2}] \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Courbes de Bézier

- En généralisant et en numérotant les points P_i de 0 à n , on obtient

$$P(t) = \sum_{i=0}^n J_{i,p}(t) P_i$$

où les points $P_i = (x_i, y_i, z_i)$ sont les $(n+1)$ points de contrôle et les polynômes $J_{i,p}$ sont définis par

$$J_{i,p}(t) = \binom{p}{i} t^i (1-t)^{p-i} \text{ où } \binom{p}{i} = \frac{p!}{i!(p-i)!}$$

Courbes de Bézier

- Pour 5 points, $p=4$

$$P(t) = [(1-t)^4, 4t(1-t)^3, 6t^2(1-t)^2, 4t^3(1-t), t^4] \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{bmatrix}$$

$$= [J_{0,4}, J_{1,4}, J_{2,4}, J_{3,4}, J_{4,4}] \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{bmatrix}$$

Courbes de Bézier

- ➊ Critique des courbes de Bézier
 - ▣ Les courbes de Bézier utilisent de l'information issue uniquement des points de contrôle (+)
 - ▣ Introduisent le concept de polygone de contrôle et d'enveloppe convexe (+)
 - ▣ Peuvent être tracées efficacement à l'aide d'un algorithme de subdivision (algorithme de Casteljau) (+)
 - ▣ L'ordre des polynômes formant les fonctions de base est égal au nombre de points de contrôle, et donc croît rapidement (-)
 - ▣ Toutes les fonctions de base influencent la forme de la courbe sur toute la plage du paramètre: pas de contrôle local (-)

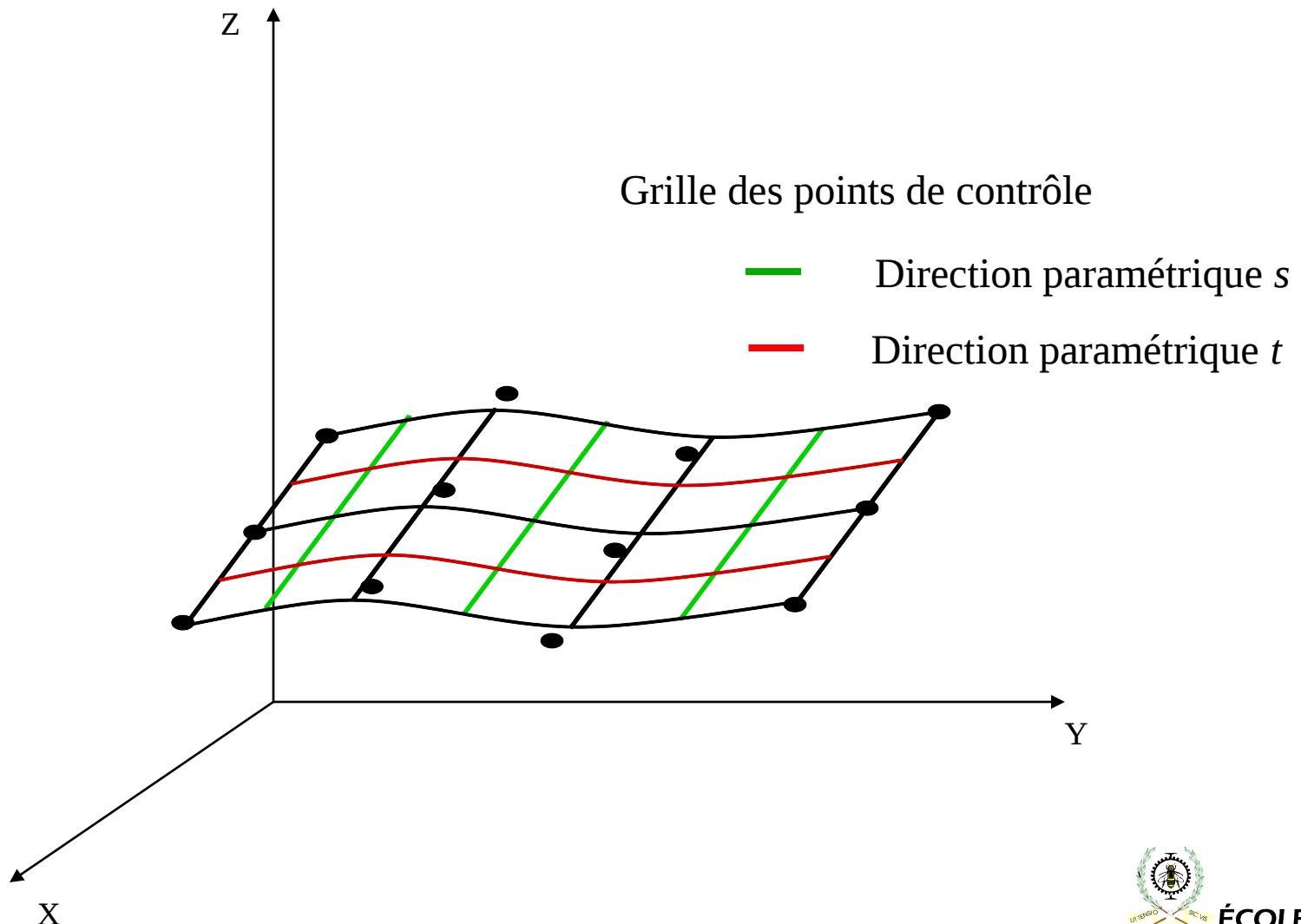
Surfaces de Bézier

- Les facettes de surfaces de Bézier sont générées à partir de la formule suivante:

$$P(t, s) = \sum_{i=0}^n \sum_{j=0}^m J_{i,p}(t) J_{j,q}(s) P_{ij}$$

- P_{ij} constitue l'ensemble des $(m+1) \times (n+1)$ points de contrôle et $J_{i,p}$ et $J_{j,q}$ sont les polynômes de Bernstein définis cette fois ci pour les deux directions s et t.
- Une facette bic cubique de Bézier a pour enveloppe convexe le polygone définis par les 16 points de contrôle P_{ij} pour $i=0, 3$ et $j=0, 3$ et passe par les points P_{00}, P_{03}, P_{30} et P_{33} .
- Les autres conditions d'approximations sont décrites par les dérivées aux coins de la facette

Surfaces de Bézier



Manipulation des courbes et des surfaces de Bézier avec OpenGL

- Le polynôme de Bernstein de degré n (ordre $n+1$) est donné par:

$$J_{i,p}(t) = \binom{p}{i} t^i (1-t)^{p-i} \text{ où } \binom{p}{i} = \frac{p!}{i!(p-i)!}$$

- Si P_i représente l'ensemble des points de contrôle pour $i = 0$ à n l'équation de la courbe de Bézier pour t variant de 0.0 à 1.0 est donnée par:

$$c(t) = \sum_{i=0}^n J_{i,p}(t) P_i$$

- Avec OpenGL, la commande `glMap1()` permet d'évaluer l'équation $c(u-u_1/u_2-u_1)$ pour permettre au paramètre u de varier entre u_1 et u_2

Évaluation d'une courbe de Bézier

- Une courbe de Bézier est évaluée dans OpenGL à l'aide de la commande:

```
glMap1{fd}( GLenum target, TYPE u1, TYPE u2,  
            GLint stride, GLint order,  
            const TYPE *points );
```

- Exemple:
 - GLdouble data[] = {....};
 - glMap1d(GL_MAP_VERTEX_3, 0.0, 1.0, 3, 4, data);
 - glEnable(GL_MAP_VERTEX_3); doit être appelé pour activer l'évaluateur
 - glEvalCoord1d(u); doit être utilisé à la place de glVertex pour générer les différents points de la courbe selon une résolution donnée
 - Autres valeurs du premier argument:
 - GL_MAP1_Normals pour générer les normales aux sommets
 - GL_MAP1_TEXTURE_COORD_1 pour générer de la texture
 - GL_MAP1_COLOR_4 pour générer de la couleur en RGBA
 - etc.

Évaluation d'une courbe de Bézier

- Exemple:

- Générer une courbe de Bézier sur l'intervalle [0, 10] à l'aide de 100 points équidistants:

```
glBegin( GL_LINE_STRIP );
for ( int i=0; i<= 100; i++)
    glEvalCoord1f( float) i/100.0 );
glEnd( );
```

- OpenGL offre une autre alternative lorsque les valeurs de u sont réparties à une distance uniforme:

```
glMapGrid1f( un, u1, u2 );  glEvalMesh1( mode, i1, i2 );
ex. : glMapGrid1f( 100, 0.0, 10.0 ); glEvalMesh1( GL_LINE, 0, 100 );
```

équivalent à :

```
glBegin( mode );
for ( int i = i1; i <= i2; ++i )
    glEvalCoord1f( u1 + i * (u2-u1)/un );
glEnd();
```

Évaluation d'une courbe de Bézier

● Définir un évaluateur

```
void glMap1{fd} ( GLenum target,  
                  TYPE u1,  
                  TYPE u2,  
                  GLint stride,  
                  GLint order,  
                  const TYPE * points );  
  
GL_MAP1_VERTEX_3           x,y,z  
GL_MAP1_VERTEX_4           x,y,z,h  
GL_MAP1_INDEX               r,g,b,a  
GL_MAP1_NORMAL              Normale  
GL_MAP1_TEXTURE_COORD_1     Coordonnées de texture s  
GL_MAP1_TEXTURE_COORD_2     Coordonnées de texture s,t  
GL_MAP1_TEXTURE_COORD_3     Coordonnées de texture s,t,r  
GL_MAP1_TEXTURE_COORD_4     Coordonnées de texture s,t,r,q
```

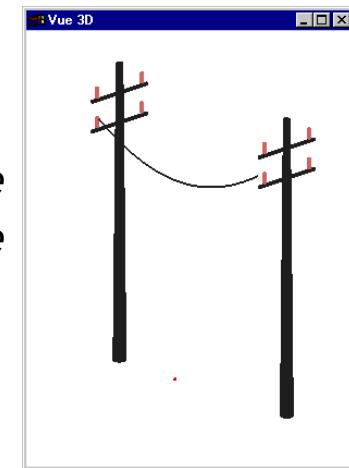
Évaluation d'une courbe de Bézier

- Évaluer les points de la courbe

```
void glEvalCoord1{fd} (TYPE u);           // valeur du paramètre  
void glEvalCoord1{fd}v (TYPE *u);         // valeur du paramètre
```

- Exemple

```
void AfficherFils( Point3D_T *pts )  
{  
    const int Ordre = 4;  
    const int n = 30;  
    glMap1f( GL_MAP1_VERTEX_3, // on veut des vertex en 3D  
             0.0, 1.0,           // u varie de 0.0 à 1.0  
             3,                  // incrément entre les x  
             Ordre,              // ordre de la Bézier = nb pts  
             (GLfloat *) pts ); // points de contrôle  
    glEnable( GL_MAP1_VERTEX_3 );  
  
    glColor3f( 0.2, 0.2, 0.2 );  
    glBegin( GL_LINE_STRIP );  
    for ( int i = 0; i < n; i++ ) // Évaluation de n points intermédiaires  
        glEvalCoord1f( (GLfloat) i /n );  
    glEnd();  
}
```



Évaluation d'une surface de Bézier

- ➊ L'évaluation d'une surface de Bézier dans OpenGL est similaire à celle des courbes sachant que les commandes utilisées ont besoin de deux paramètres u et v .
- ➋ `glMap2{fd}(GLenum target, TYPE u1, TYPE u2, GLint ustride,
 GLint uorder, TYPE v1, TYPE v2, GLint vstride,
 GLint vorder, TYPE *points);`
- ⌋ `glEvalCoord2{fd}(TYPE u, TYPE v);`
- ⌋ `glMapGrid2{fd}(GLint nu, TYPE u1, TYPE u2,
 GLint nv, TYPE v1, TYPE v2);`
- ⌋ `glEvalMesh2(GLenum mode, GLint i1, GLint i2, GLint j1, GLint j2);`

Courbes B-Splines

Courbes B-Splines

- ➊ Idée: fournir un contrôle local sur la courbe en utilisant des fonctions de base à support compact.
 - ▣ Soit une courbe $C(t)$ dont la plage de paramètre va de $t=0$ à $t=1$,

▣ Soit:

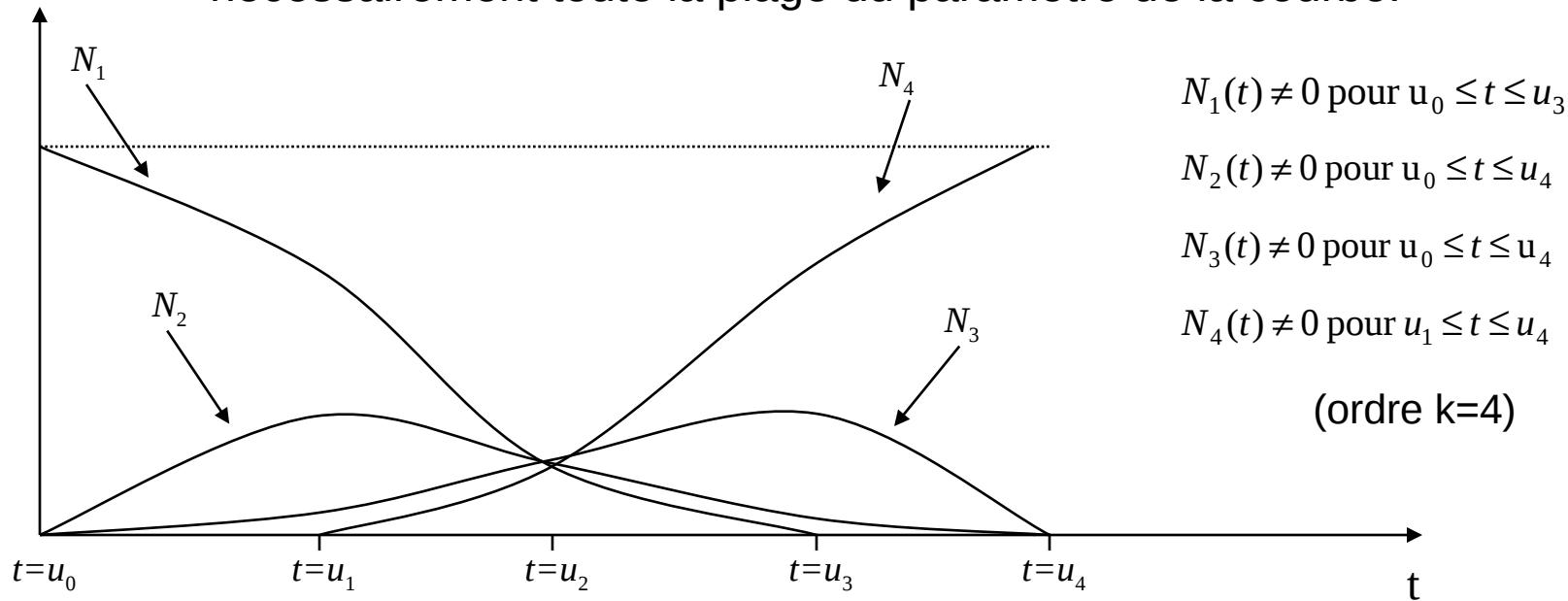
$$C(t) = \sum_{i=1}^n N_{i,k}(t) P_i$$

avec: P_i la position des points de contrôle
 $N_{i,k}(t)$ les fonctions des base d'ordre k

- ▣ Si la fonction de base i à un support compris entre $t = u_j$ et $t = u_{j+k}$, le point de contrôle P_i n'a de l'influence sur la courbe qu'à l'intérieur de cette plage de paramètre.

Courbes B-Splines

- ➊ Définition de la plage de support pour chaque fonction de base:
 - La plage de paramètre doit être divisée en intervalles, et chaque fonction de base est non-nulle sur un ou plusieurs intervalles.
 - Ces intervalles, pour une fonction donnée, ne représentent pas nécessairement toute la plage du paramètre de la courbe.



Courbes B-Splines

- ➊ Définition de la plage de support pour chaque fonction de base:
 - ▣ On définit le **vecteur nodal** comme le vecteur des u_j , c'est-à-dire des positions de début ou de fin des supports compacts des fonctions de base.
 - ▣ La construction des fonctions de base pour les B-Splines repose sur la spécification de deux informations:
 - ▶ L'ordre k des fonctions à construire (degré des polynômes + 1)
 - ▶ Le vecteur nodal (les u_j)

Courbes B-Splines

- ➊ Définition des fonctions de base à partir du vecteur nodal:
 - ➋ Les fonctions de base sont définies de façon récursive à l'aide de l'algorithme suivant (algorithme de Cox-de Boor):

$$N_{i,1}(t) = \begin{cases} 1 & \text{si } u_i \leq t \leq u_{i+1} \\ 0 & \text{sinon} \end{cases}$$

$$N_{i,k}(t) = \frac{t - u_i}{u_{i+k-1} - u_i} N_{i,k-1}(t) + \frac{u_{i+k} - t}{u_{i+k} - u_{i+1}} N_{i+1,k-1}(t)$$

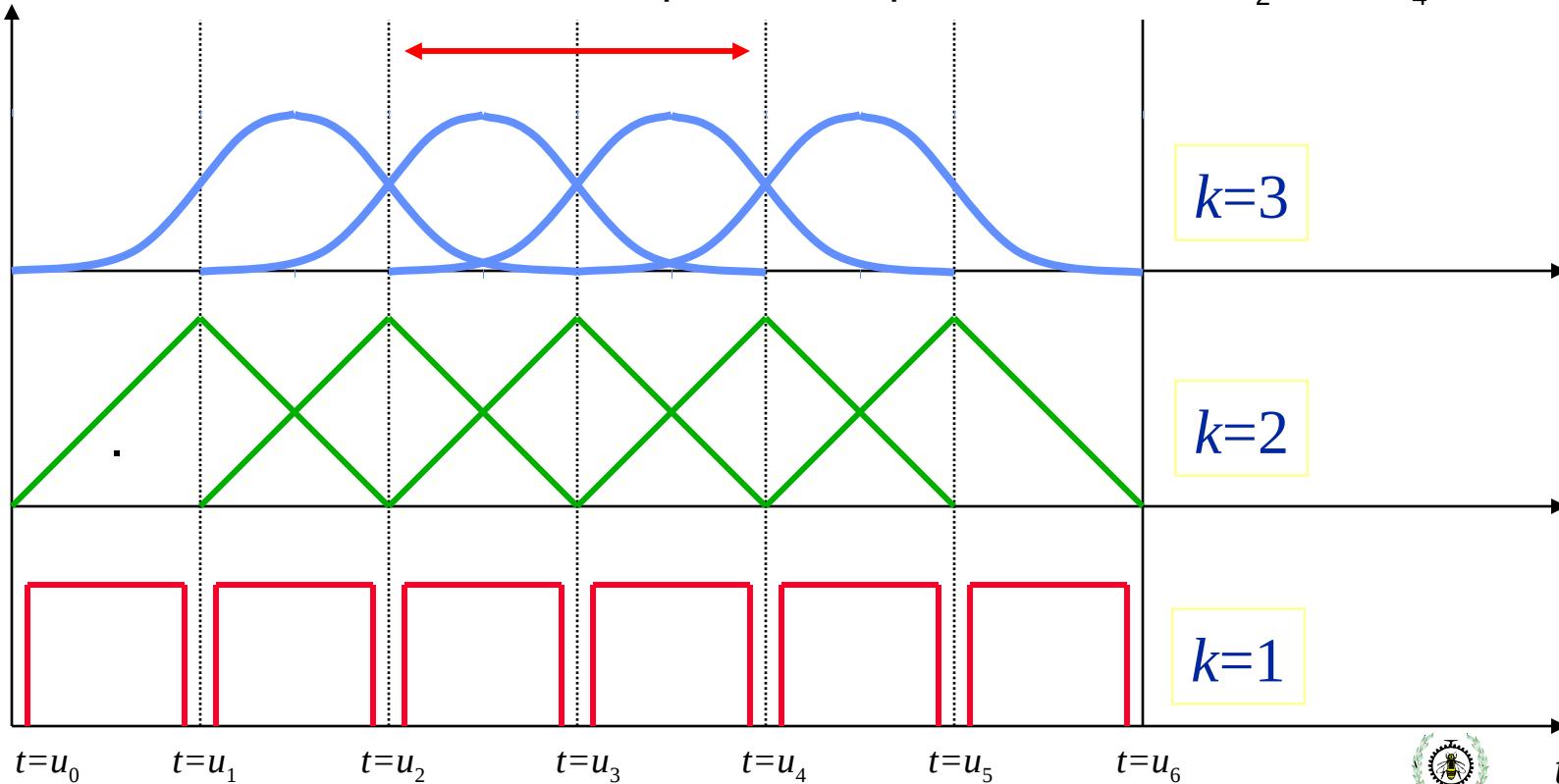
- ➋ Une fonction de l'ordre k est définie à partir de deux fonctions de l'ordre $k-1$.

Courbes B-Splines

- ➊ Définition des fonctions de base à partir du vecteur nodal pour n points de contrôle:
 - ▣ Puisque l'on perd 1 fonction chaque fois que l'on élève l'ordre de 1, il faut construire $n+k$ fonctions à l'ordre 1 pour obtenir n fonctions à l'ordre k . Donc, *il faut au départ $n+k$ nœuds dans le vecteur nodal.*
 - ▣ La somme des fonctions de base doit être 1 pour toute valeur valide du paramètre: $\sum N_i(t) = 1$
 - ▣ La *plage valide du paramètre* est ainsi du $k^{\text{ème}}$ nœud au $(n+1)^{\text{ème}}$ nœud, c'est-à-dire la plage pour laquelle k fonctions sont simultanément définies: les $k-1$ premiers intervalles et les $k-1$ derniers intervalles sont exclus.

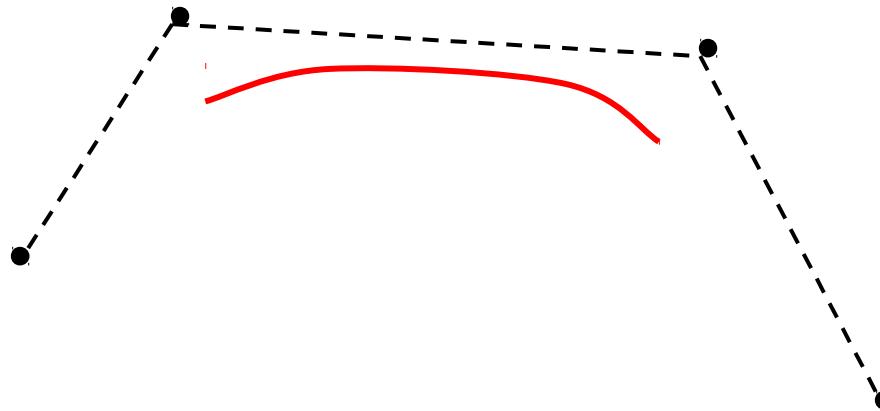
Courbes B-Splines

- Définition des fonctions de base à partir du vecteur nodal:
 - Ordre $k=3$ ($n=4$ et $n+k=7$)
 - Vecteur nodal = { $u_0, u_1, u_2, u_3, u_4, u_5, u_6$ }
 - Intervalle de validité du paramètre pour la courbe: $u_2 \leq t \leq u_4$



Courbes B-Splines

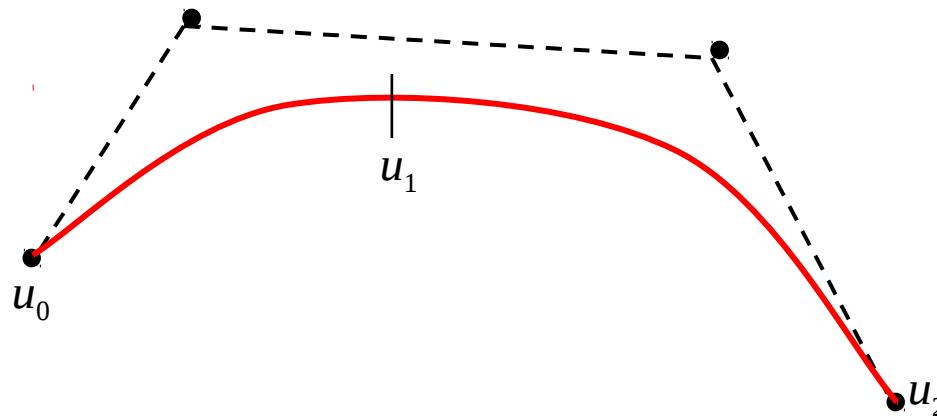
- ➊ Définition des fonctions de base à partir du vecteur nodal:
 - Ordre $k=3$ ($n=4$, 7 noeuds)
 - Vecteur nodal = { $u_0, u_1, u_2, u_3, u_4, u_5, u_6$ } (uniforme, périodique)
 - Intervalle de validité du paramètre pour la courbe: $u_2 \leq t \leq u_4$
 - *Périodique car l'intervalle de validité n'est pas de u_0 à u_6*
⇒ La courbe ne se rend pas aux deux extrémités



Courbes B-Splines

- ➊ Définition des fonctions de base à partir du vecteur nodal:

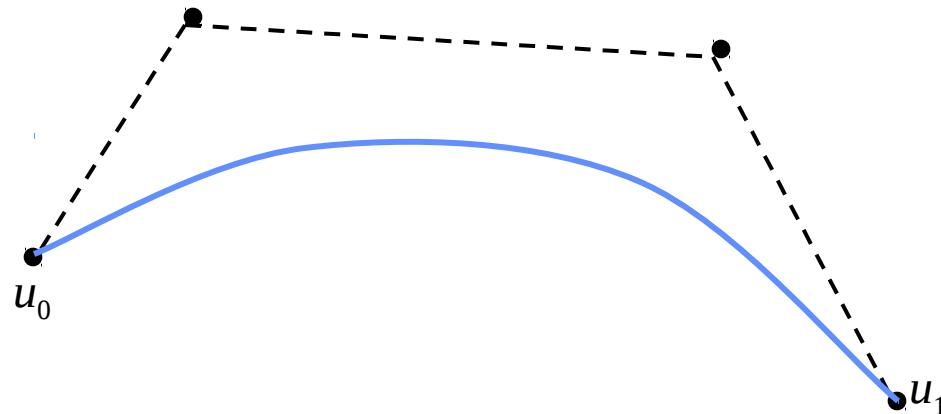
- Ordre $k=3$ ($n=4$, 7 noeuds)
- Vecteur nodal = { $u_0, u_0, u_0, u_1, u_2, u_2, u_2$ } (uniforme, ouvert)
- Intervalle de validité du paramètre pour la courbe: $u_0 \leq t \leq u_2$
- Ouvert car l'intervalle de validité est de u_0 à u_2
⇒ La courbe rend aux deux extrémités



- La courbe B-Splinaire est constituée de deux segments de parabole

Courbes B-Splines

- ➊ Définition des fonctions de base à partir du vecteur nodal:
 - Ordre $k=4$ ($n=4$, 8 noeuds)
 - Vecteur nodal = { $u_0, u_0, u_0, u_0, u_1, u_1, u_1, u_1$ } (uniforme, ouvert)
 - Intervalle de validité du paramètre pour la courbe: $u_0 \leq t \leq u_1$



- La courbe B-Splinaire est constituée d'un segment de courbe cubique
- Si $n = k$ et si le vecteur nodal est ouvert (le premier nœud et le dernier nœud sont répétés k fois), la B-Spline devient une courbe de Bézier!

Choix d'un vecteur de valeurs nodales

Vecteur nodal uniforme ouvert

- Les valeurs extrémales: degré de multiplicité égal à l'ordre k de la fonction de base B-spline.
- Les valeurs internes sont réparties également.

Voici quelques exemples:

$$k = 2, n = 4 \quad [0 \ 0 \ 1 \ 2 \ 3 \ 3]$$

$$k = 3, n = 5 \quad [0 \ 0 \ 0 \ 1 \ 2 \ 3 \ 3 \ 3]$$

et en mode normalisé

$$k = 2, n = 4 \quad [0 \ 0 \ 0.33 \ 0.66 \ 1 \ 1]$$

$$k = 3, n = 5 \quad [0 \ 0 \ 0 \ 0.33 \ 0.66 \ 1 \ 1 \ 1]$$

Choix d'un vecteur de valeurs nodales

Vecteur nodal non-uniforme

- ➊ Un vecteur nodal non-uniforme: valeurs nodales réparties non-uniformément et les valeurs internes peuvent posséder un degré de multiplicité plus grand ou égal à 1.
- ➋ Le vecteur peut être ouvert ou périodique. Voici quelques exemples:

[0 0 1 1 2 3 3] vecteur non-uniforme ouvert avec un degré de multiplicité 2 pour le premier noeud interne ($k=2, n=5$)

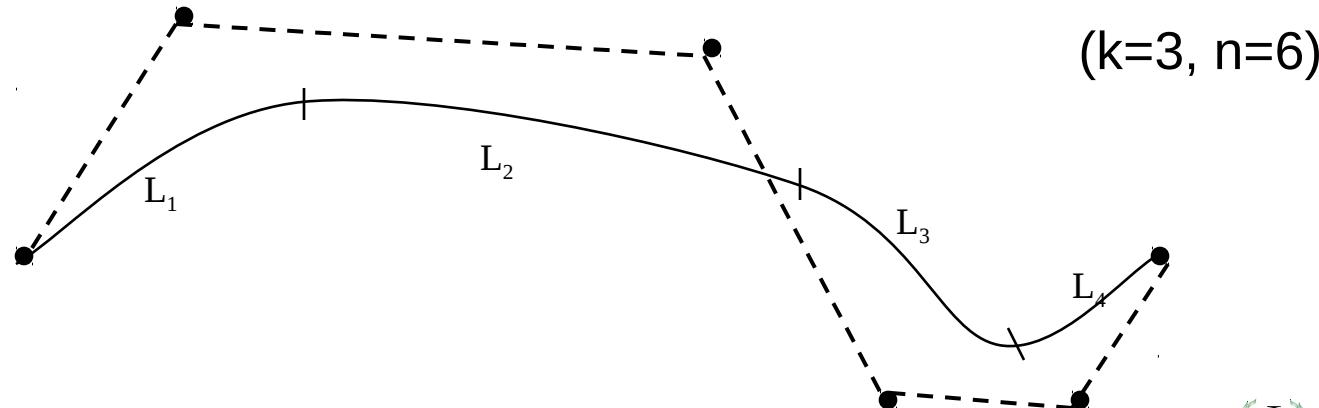
[0 1 1 2 2 3] vecteur non-uniforme périodique avec un degré de multiplicité 2 pour les noeuds internes ($k=2, n=4$)

Choix d'un vecteur de valeurs nodales

Vecteur non-uniforme normalisé

- Les valeurs internes du vecteur nodal peuvent être générées proportionnellement à la longueur de la corde entre les points P_i .

$$[0 \ 0 \ 0 \ (L_1/L_{tot}) \ ((L_1+L_2)/L_{tot}) \ ((L_1+L_2+L_3)/L_{tot}) \ 1 \ 1 \ 1]$$



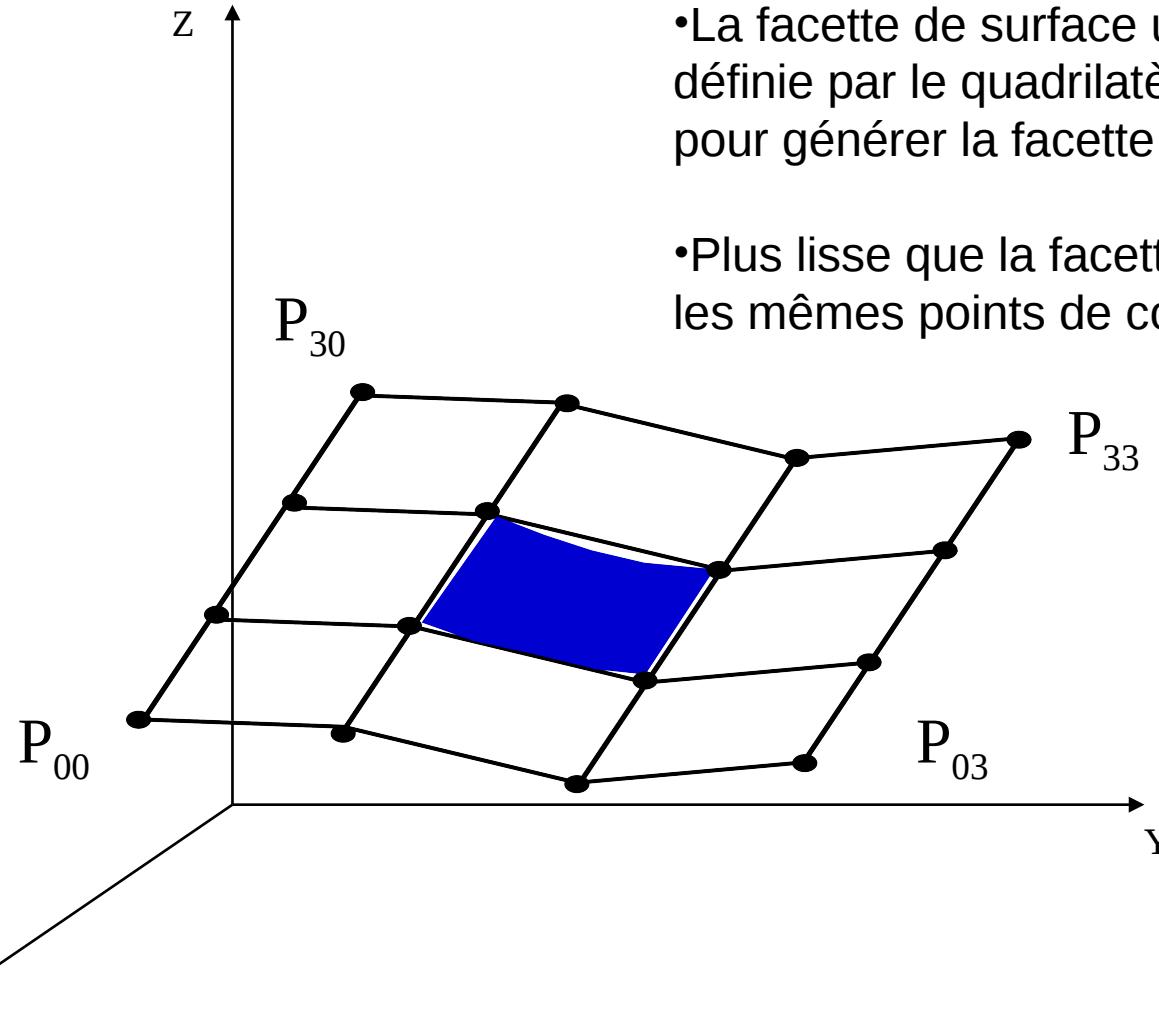
Surfaces d'approximation B-Splinaires

- ➊ Une surface paramétrique est une entité à deux paramètres, s et t définie à partir d'une grille de points de contrôle

$$S(s, t) = \sum_{i=1}^n \sum_{j=1}^m N_{i,k_s}(s) N_{j,k_t}(t) P_{i,j}$$

- ➋ Dans le cas d'une surface, on doit définir:
 - ▣ La grille des points de contrôle
 - ▣ L'ordre selon le premier paramètre (k_s)
 - ▣ L'ordre selon le deuxième paramètre (k_t)
 - ▣ Le vecteur nodal pour le premier paramètre
 - ▣ Le vecteur nodal pour le second paramètre

Surfaces d'approximation B-Splinaires



- La facette de surface utilisée est celle définie par le quadrilatère central (9 fois pour générer la facette complète)
- Plus lisse que la facette de Bézier même si les mêmes points de contrôle sont utilisés

Courbe B-Spline non-uniforme rationnelle : NURBS

Courbe B-Spline non-uniforme rationnelle : NURBS

- ➊ Les B-Splines sont des outils très puissants pour la modélisation de courbes quelconques, mais elles ne permettent pas de représenter exactement les coniques.
- ➋ En introduisant une coordonnée homogène supplémentaire pour chaque point de contrôle, on peut définir une nouvelle classe de courbes qui sont une généralisation des B-Splines, et qui permettent de représenter de façon exacte les coniques: les NURBS (**N**on-**U**niform **R**ational **B**-Splines).
- ➌ Les B-Splines sont ainsi des NURBS pour lesquelles la coordonnée homogène de chacun des points de contrôle est 1.

NURBS

- En introduisant une coordonnée homogène pour chaque point, il faut maintenant diviser par la coordonnée homogène avant de tracer (si $h \neq 1$)
- Les fonctions de base ne sont plus des polynômes, mais des fonctions rationnelles:

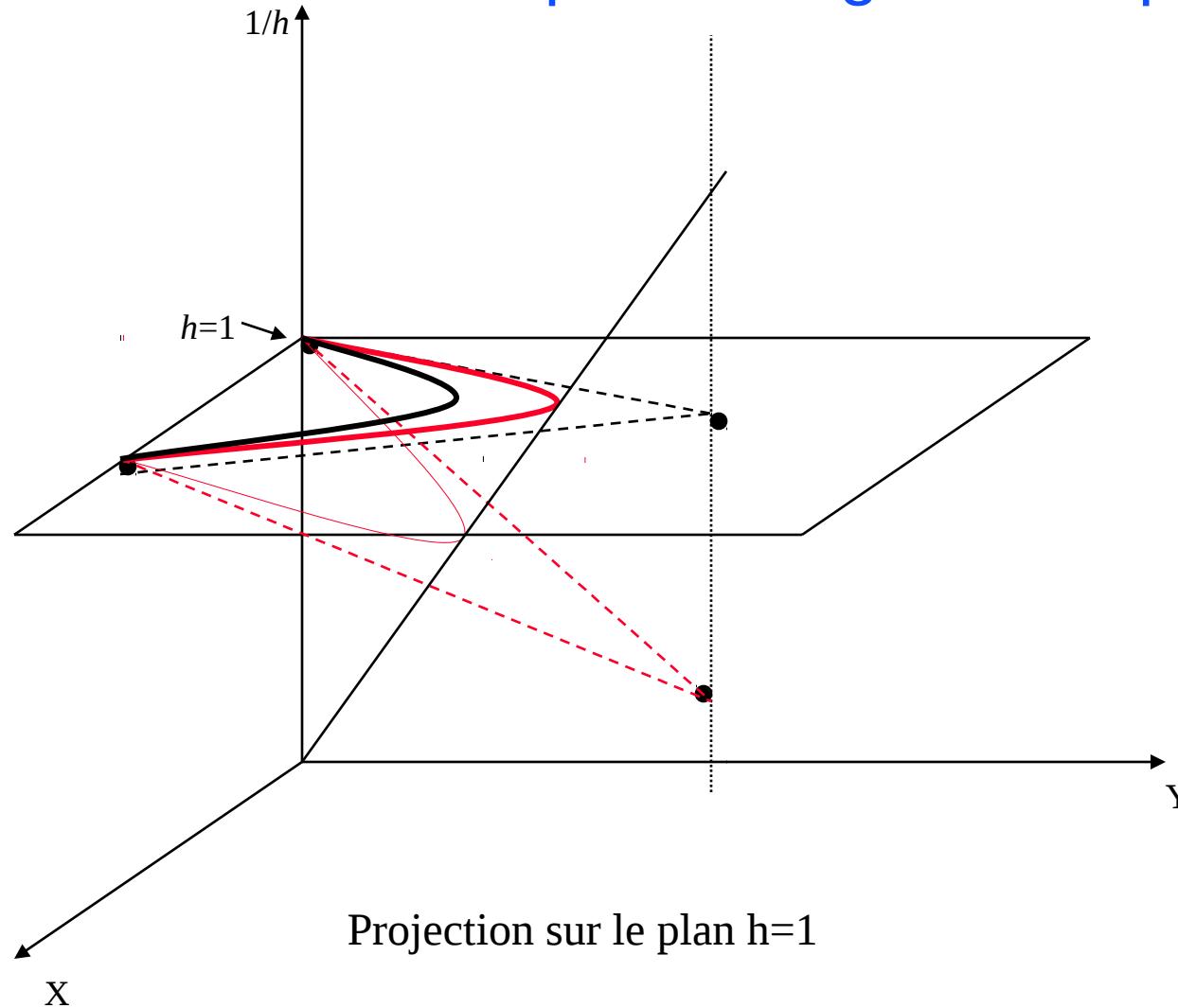
$$C(t) = \frac{\sum_{i=1}^n h_i N_{i,k}(t) P_i}{\sum_{i=1}^n h_i N_{i,k}(t)} = \sum_{i=1}^n \left(\frac{h_i N_{i,k}(t)}{\sum_{j=1}^n h_j N_{j,k}(t)} \right) P_i = \sum_{i=1}^n R_{i,k}(t) P_i$$

- avec: $N_{i,k}$ les fonctions de base B-Splinaires

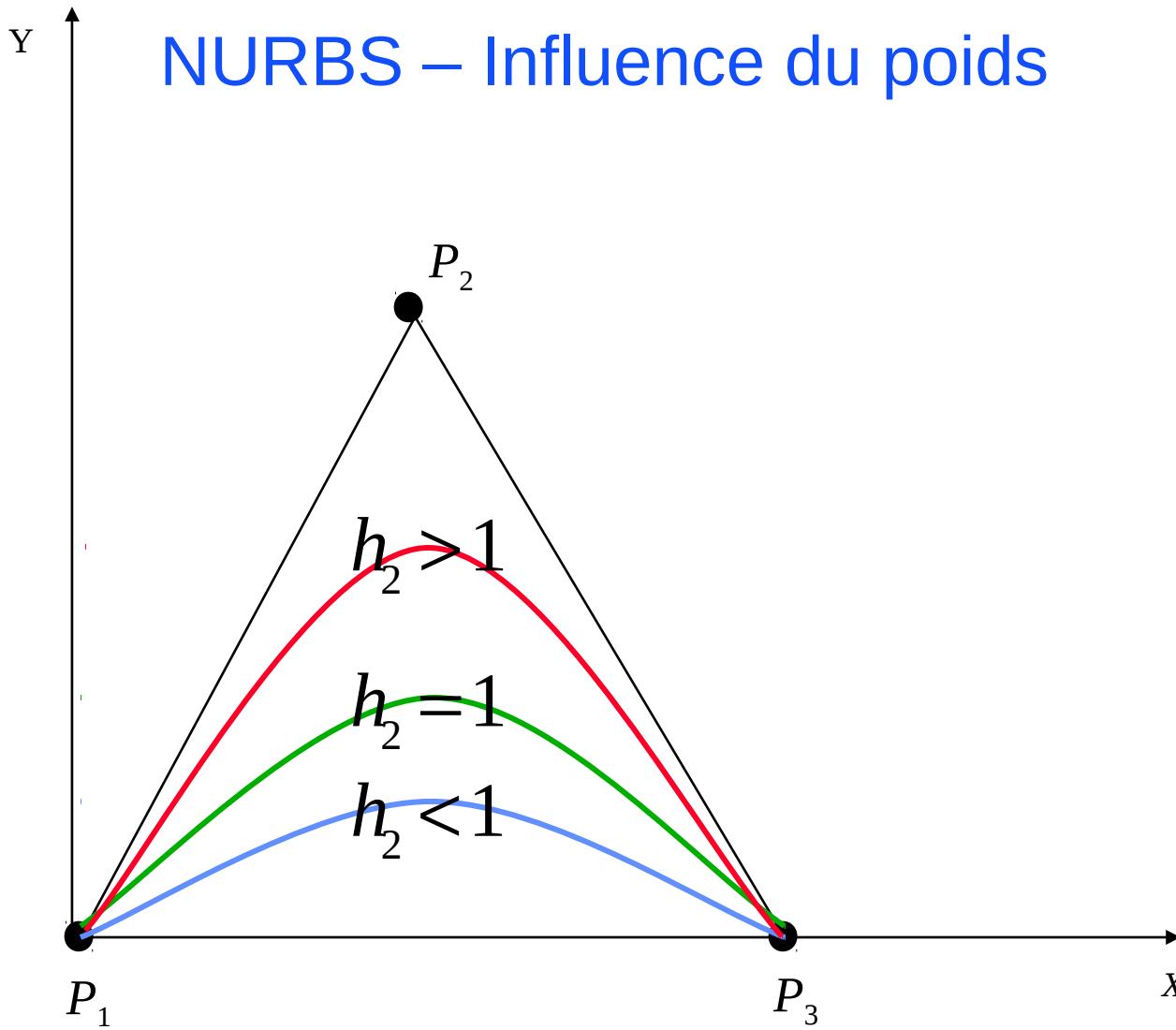
h_i le poids associé à chaque point de contrôle

$R_{i,k}$ les fonctions de base rationnelles

NURBS – Interprétation géométrique



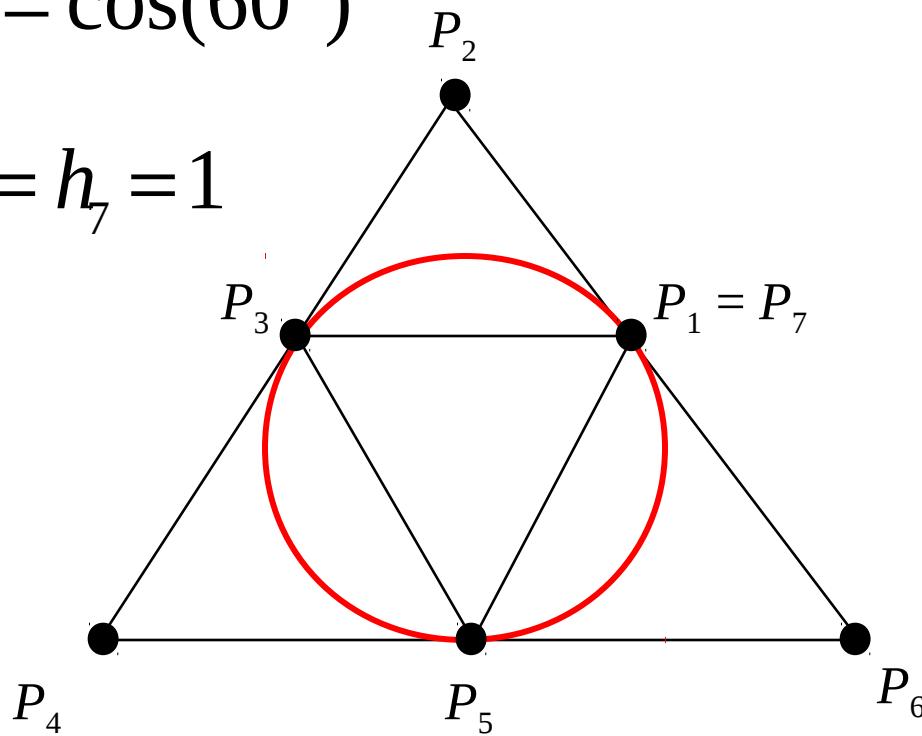
NURBS – Influence du poids



NURBS – Représentation d'un cercle

$$h_2 = h_4 = h_6 = \cos(60^\circ)$$

$$h_1 = h_3 = h_5 = h_7 = 1$$



Manipulation des NURBS dans OpenGL

- GLU offre une interface NURBS qui permet de manipuler des courbes et des surfaces NURBS dans OpenGL
- gluBeginCurve()* ou *gluBeginSurface()*
- gluNurbsCurve()* ou *gluNurbsSurface()* pour générer et faire le rendu d'une courbe ou d'une surface
- gluEndCurve()* ou *gluEndSurface()*
- glEnable(GL_AUTO_NORMAL)* permet de générer les normales aux sommets automatiquement.
- gluNewNurbsRenderer()* permet de créer un pointeur sur un objet de type NURBS.
- gluNurbsProperty()* permet de choisir la valeurs des paramètres de rendu tels que la taille maximale des lignes ou des polygones.
- gluNurbsCallback()* permet de gérer les erreurs.

Évaluation d'une NURBS

● Étapes

- Créer un objet Nurbs avec *gluNewNurbsRenderer()*;
- Si nécessaire, définir les caractéristiques de la courbe (épaisseur du trait,) avec *gluNurbsProperty()*;
- Enregistrer une fonction de rappel en cas d 'erreur avec *gluNurbsCallback()*;
- Commencer la courbe ou surface avec *gluBeginCurve()* ou *gluBeginSurface()*;
- Générer la courbe ou la surface avec *gluNurbsCurve()* ou *gluNurbsSurface()*;
- Terminer la courbe ou la surface avec *gluEndCurve()* ou *gluEndSurface()*;

Évaluation d'une NURBS

- Créer un objet Nurbs avec *gluNewNurbsRenderer()*;

```
GLUnurbsObj *Fils = gluNewNurbsRenderer();
```

- Enregistrer une fonction de rappel en cas d'erreur avec *gluNurbsCallback()*;

```
gluNurbsCallback( Fils, GLU_ERROR, NurbsErreur );
```

```
void NurbsErreur( GLenum CodeErreur )
{
    const GLubyte *Erreur = gluErrorString( CodeErreur );
    printf( "Erreur Nurbs : %s \n", Erreur );
}
```

Évaluation d'une NURBS

- Commencer la courbe ou surface avec *gluBeginCurve()*:

```
void gluBeginCurve ( GLUnurbsObj *nobj );
```

- Générer la courbe ou la surface avec *gluNurbsCurve()*:

```
gluNurbsCurve ( GLUnurbsObj * nobj,           // Objet Nurbs  
                 GLint uknot_count, // Taille du vecteur nodal  
                 GLfloat *uknot,    // Vecteur nodal  
                 GLint u_stride,   // Incrément entre les x  
                 GLfloat *ctlarray, // Points de contrôle  
                 GLint uorder,     // Ordre de la spline  
                 GLenum type);    // GL_MAP1_VERTEX_3 ou  
                           // GL_MAP1_VERTEX_4
```

- Terminer la courbe ou la surface avec *gluEndCurve()*:

```
void gluEndCurve( GLUnurbsObj *nobj );
```

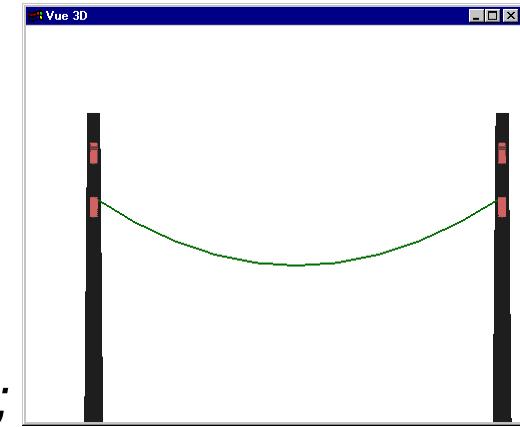
Évaluation d'une NURBS

- Exemple

```
void AfficherFils1( Point4D_T *pts, int n )
{
    const int Ordre = 4;
    GLfloat VecteurNodal[ ] = { 0.0, 0.0, 0.0, 0.0,
                                1.0, 1.0, 1.0, 1.0 };
    GLUnurbsObj *Fils = gluNewNurbsRenderer();

    gluNurbsCallback( Fils, GLU_ERROR, NurbsErreur );

    gluBeginCurve( Fils );
    gluNurbsCurve( Fils, Ordre + n,
                  VecteurNodal, 4,
                  (GLfloat *) &pts[0], Ordre,
                  GL_MAP1_VERTEX_3 );
    gluEndCurve( Fils );
}
```



Évaluation d'une NURBS

$$\rightarrow P(u) = \sum_{i=1}^{n+1} P_i R_{i,k}(u)$$

$$N_{i,1}(u) = \begin{cases} 1 & u_i \leq u \leq u_{i+1} \\ 0 & \text{sinon} \end{cases}$$

$$N_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} N_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} N_{i+1,k-1}(u)$$

$$0 \leq u \leq n-k+2$$

$$u_j = \begin{cases} 0 & \text{si } j \leq k \\ j-k+1 & \text{si } k < j \leq n+1 \\ n-k+2 & \text{si } j > n+1 \end{cases}$$

j varie de 1 à $n+k+1$.

Évaluation d'une NURBS

➊ Étapes

- Exactement les mêmes que pour une B-spline sauf que certains points P_i sont multipliés par la coordonnées h_i afin d'augmenter le poids de ces points.

Par exemple, $P_1 = (x_1, y_1, z_1, 1)$;

$$P_2 = (x_2', y_2', z_2', h_2) = (x_2 * h_2, y_2 * h_2, z_2 * h_2, h_2);$$

$$P_3 = (x_3', y_3', z_3', h_3) = (x_3 * h_3, y_3 * h_3, z_3 * h_3, h_3);$$

$$P_4 = (x_4, y_4, z_4, 1);$$

Évaluation d'une NURBS

- Exemple

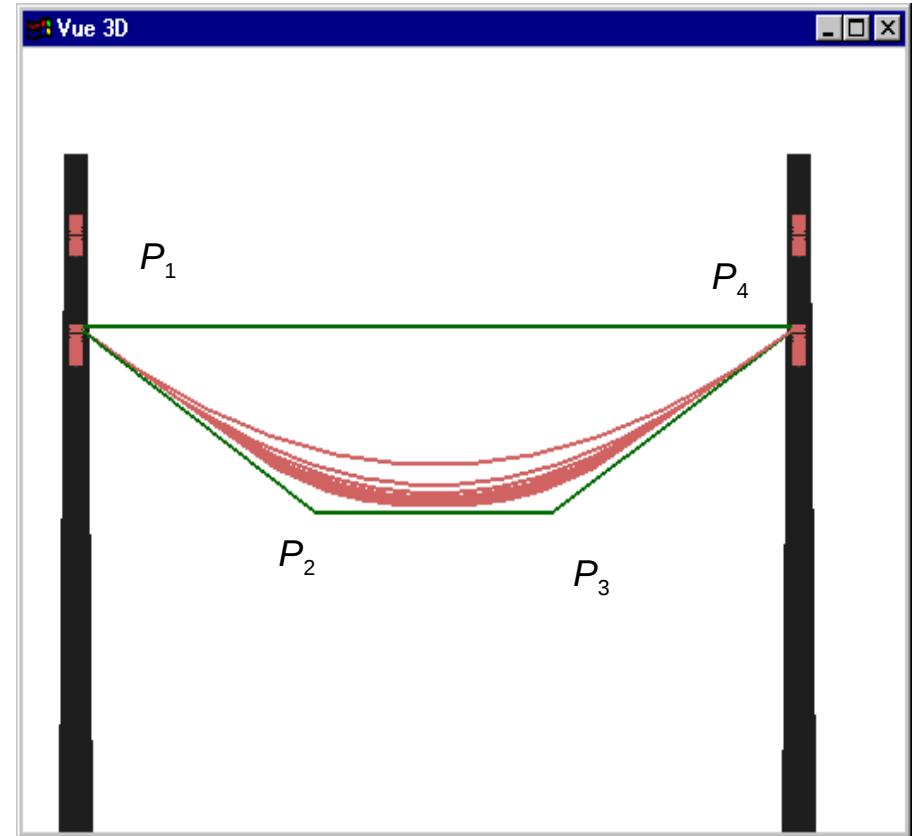
```
void AfficherFils2( Point4D_T *pts, int n)
{
    const int Ordre = 4;
    GLfloat VecteurNodal[ ] = { 0.0, 0.0, 0.0, 0.0,
                                1.0, 1.0, 1.0, 1.0 };
    GLUnurbsObj *Fils = gluNewNurbsRenderer();

    gluNurbsCallback( Fils, GLU_ERROR, NurbsErreur );

    gluBeginCurve( Fils );
    gluNurbsCurve( Fils, Ordre + n,
                  VecteurNodal, 4,
                  (GLfloat *) &pts[0], Ordre,
                  GL_MAP1_VERTEX_4 );
    gluEndCurve( Fils );
}
```

Évaluation d'une NURBS

- Exemple pour 4 points et pour P_2 et P_3 , $h = 1, 3, 5, 7, 9, 11, 13, 15, 17, 19$;
- Vecteur nodal
 $U = [0, 0, 0, 0, 1, 1, 1, 1]$



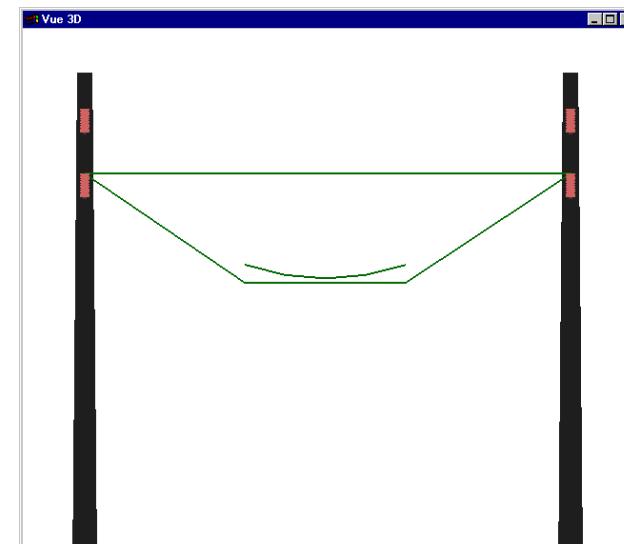
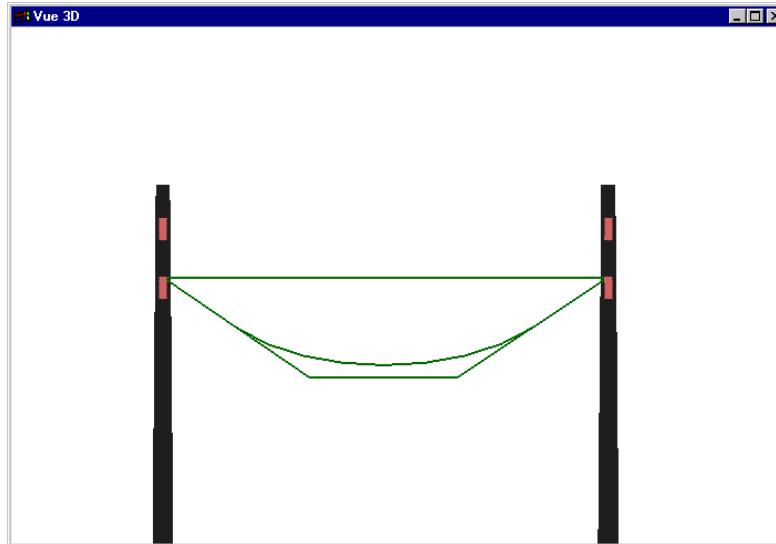
Évaluation d'une NURBS

- Vecteur nodal

$$U= [0, 0, 1, 1, 2, 2, 3, 3]$$

- Vecteur nodal

$$U= [0, 1, 2, 3, 4, 5, 6, 7]$$



Évaluation d'une NURBS

- Vecteur nodal

$$U = [0, 0, 0.1, 0.1, 0.9, 1, 1, 1]$$

