Chapitre 7

Courbes et surfaces

7.1 Introduction

Les courbes et surfaces jouent un rôle important en infographie. Différents types de courbes peuvent être manipulées comme les courbes paramétriques, les courbes non paramétriques (implicites ou explicites) ou encore les courbes fractales. Elles permettent d'obtenir des représentations précises de formes douces contrairement à la représentation des formes par polygônes. De plus, elles nécessitent un espace mémoire moindre et il n'est pas nécessaire d'interpoler la forme entre les points.

Nous n'aborderons dans ce chapitre que l'étude de quelques courbes et surfaces paramétriques définies par des points de contrôle. Les courbes seront définies dans un espace à deux dimensions, la généralisation de celles-ci à un espace tridimensionnel étant aisée. De plus, nous ne décrirons que brièvement les surfaces, celles-ci se généralisant de manière naturelle des courbes.

Il existe deux types de courbes paramétriques : les courbes interpolées et les courbes approchées. Les courbes interpolées relient les points de contrôle entre eux alors que les courbes approchées ont une forme qui est influencée par la position des points de contrôle qui peuvent être extérieurs à la courbe.

7.2 Les courbes de Bézier

7.2.1 Définition

Les courbes de Bézier sont des courbes paramétriques approximant un réseau de points $P_i = (x_i, y_i), 0 \leqslant i \leqslant n$ (c'est-à-dire $n+1$ points de $\mathbb{R}^2$).

Soit $B(t)$ la courbe de Bézier de **degré** n (on dit aussi qu'elle est **d'ordre** $n+1$) approximant le réseau de points $P_i, 0 \leqslant i \leqslant n$:

$$B(t) = \left\{ \begin{array}{l} x(t) \\ y(t) \end{array} \right. , \forall t \in [0,1].$$

Les points d'une courbe de Bézier sont des barycentres des points du réseau, on a donc :

$$B(t) = \sum_{i=0}^{n} P_i \mathcal{B}_{i,n}(t), \forall t \in [0,1],$$

c'est-à-dire

$$\begin{cases} x(t) = \sum_{i=0}^{n} x_i \mathcal{B}_{i,n}(t) \\ y(t) = \sum_{i=0}^{n} y_i \mathcal{B}_{i,n}(t) \end{cases}, \forall t \in [0,1]$$

où les $\mathcal{B}_{i,n}(t)$ correspondent à des fonctions de pondération dans la formule de calcul des barycentre : les polynômes de Bernstein.

On a

$$\mathcal{B}_{i,n}(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}, \forall t \in [0,1],$$

(loi de distribution de la probabilité binomiale).

On peut aussi définir ces polynômes de façon récursive (Fig. 7.1) :

- Sur deux points P_0, P_1 : $B(t) = (1-t)P_0 + tP_1 (= \sum_{i=0}^{1} P_i \mathcal{B}_{i,1}(t))$
- Sur trois points P_0, P_1, P_2 :
  - Sur P_0, P_1 : $E(t) = (1-t)P_0 + tP_1$
  - Sur P_1, P_2 : $F(t) = (1-t)P_1 + tP_0$

  D'où

$$\begin{aligned} B(t) &= (1-t)E(t) + tF(t) \\ &= (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \\ \Big(&= \sum_{i=0}^{2} P_i \mathcal{B}_{i,2}(t) \Big) \end{aligned}$$

- Sur quatre points P_0, P_1, P_2, P_3 :
  - Sur P_0, P_1, P_2 : $G(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$
  - Sur P_1, P_2, P_3 : $H(t) = (1-t)^2 P_1 + 2t(1-t)P_2 + t^2 P_3$

  D'où

$$\begin{aligned} B(t) &= (1-t)G(t) + tH(t) \\ &= (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \\ \Big(&= \sum_{i=0}^{3} P_i \mathcal{B}_{i,3}(t) \Big) \end{aligned}$$

$$\vdots$$

7.2.2 Quelques propriétés

Nous avons affirmé que les polynômes de Bernstein étaient des fonctions de pondération, en voici la preuve :

Proposition 7.2.1 *Les fonctions $\mathcal{B}_{i,n}$ correspondent à des fonctions de pondération :*

$$\sum_{i=0}^{n} \mathcal{B}_{i,n}(t) = 1, \forall t \in [0,1]$$

Preuve. On a

$$\begin{aligned}(a+b)^n &= \sum_{p=0}^{n} C_n^p a^p b^{n-p} \\ &= \sum_{p=0}^{n} \frac{n!}{p!(n-p)!} a^p b^{n-p}\end{aligned}$$

donc

$$\begin{aligned}\sum_{i=0}^{n} \mathcal{B}_{i,n}(t) &= \sum_{i=0}^{n} \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i} \\ &= (t+(1-t))^n \\ &= 1\end{aligned}$$

■

Nous donnons maintenant deux propriétés relatives au comportement de la courbe à ses extrémités :

Propriété 7.2.2 *La courbe de Bézier passe par les deux extrémités du réseau de points à approcher.*

Preuve. Soit $P_i, i = 0 \dots n$ le réseau de points à approcher.

On veut montrer que $B(0) = P_0$ et $B(1) = P_n$.

– Vérifions que $B(0) = P_0$:

$$B(0) = \sum_{i=0}^{n} \mathcal{B}_{i,n}(0) P_0$$

et

$$\mathcal{B}_{i,n}(0) = \frac{n!}{i!(n-i)!} 0^i 1^{n-i}$$

on a donc

$$\mathcal{B}_{i,n}(0) = \begin{cases} 1 \text{ si } i = 0 \ (0^i = 1 \text{ et } 0! = 1) \\ 0 \text{ sinon} \end{cases}$$

d'où $B(0) = P_0$.

– On montre de même que $B(1) = P_n$.

■

Propriété 7.2.3 *Les tangentes aux extrémités de la courbe sont portées par le premier et le dernier segment du réseau de points :* $[P_0, P_1]$ *et* $[P_{n-1}, P_n]$.

Preuve. Soit $\overrightarrow{v}$ le vecteur directeur de la tangente à la courbe au point P_0. On suppose B prolongée par continuité en 0, on a $\overrightarrow{v} = (x'(0), y'(0))$, soit

$$\overrightarrow{v} = \begin{pmatrix} \sum_{i=0}^{n} x_i \mathcal{B}'_{i,n}(t) \\ \sum_{i=0}^{n} y_i \mathcal{B}'_{i,n}(t) \end{pmatrix}.$$

Calculons $\mathcal{B}'_{i,n}(t)$:

– Pour tout $i \in]0, n[$, on a

$$\mathcal{B}'_{i,n}(t) = \frac{n!}{i!(n-i)!}\left[it^{i-1}(1-t)^{n-i} - t^i(n-i)(1-t)^{n-i-1}\right].$$

– Pour $i = 0$, on a $\mathcal{B}'_{0,n}(t) = -n(1-t)^{n-1}$ car $\mathcal{B}_{0,n}(t) = (1-t)^n$.

– Pour $i = n$, on a $\mathcal{B}'_{n,n}(t) = nt^{n-1}$ car $\mathcal{B}_{n,n}(t) = t^n$.

Ainsi, pour $i \in]0, n[$, on a :

– Si $i = 1$, on obtient

$$\mathcal{B}'_{i,n}(t) = n\left[(1-t)^{n-1} - t(n-1)(1-t)^{n-2}\right]$$

d'où $\mathcal{B}'_{i,n}(0) = n(1-0)^{n-1} = n$.

– Si $i \neq 1$ alors $\mathcal{B}'_{i,n}(0) = 0$.

et $\mathcal{B}'_{0,n}(0) = -n$, $\mathcal{B}'_{n,n}(0) = 0$.

On obtient donc

$$\begin{aligned} x'(0) &= x_0(-n) + x_1 n + \sum_{i=2}^{n} x_i.0 \\ &= n(x_1 - x_0). \end{aligned}$$

De même, on montre que $y'(0) = n(y_1 - y_0)$.

Le vecteur directeur de la tangente en 0 s'écrit :

$$\overrightarrow{v} = n\begin{pmatrix} x_1 - x_0 \\ y_1 - y_0 \end{pmatrix} = n(\overrightarrow{P_0P_1})$$

On vérifie de même que le vecteur directeur de la tangente en 1 est un vecteur directeur de la droite $(P_{n-1}P_n)$. ∎

Corollaire 7.2.4 *Les courbes de Béziers peuvent s'enchaîner en conservant la continuité de leurs dérivées à leur point de jonction dès l'instant où les côtés adjacents des deux segments reliant les points extrêmes des réseaux servant à leur construction sont alignés (Fig. 7.2).*

Les courbes de Bézier ont un « bon » comportement vis-à-vis des transformations affines :

Proposition 7.2.5 *La transformée affine d'une courbe de Bézier est la courbe de Bézier approchant par la transformée des points du réseau.*

Preuve. Soit $\mathcal{A}$ la transformation affine et $\mathcal{L}$ la transformation linéaire associée. On a alors :

$$\mathcal{A}(M) = \mathcal{A}(O) + \mathcal{L}(M)$$

d'où

$$\begin{aligned} \mathcal{A}(B(t)) &= \mathcal{A}(O) + \mathcal{L}\left(\sum_{i=0}^{n} \mathcal{B}_{i,n}(t).P_i\right) \\ &= \mathcal{A}(O) + \sum_{i=0}^{n} \mathcal{B}_{i,n}(t).\mathcal{L}(P_i) \end{aligned}$$

Soient $O' = \mathcal{A}(O)$, $B'(t) = \mathcal{A}(B(t))$ et $P'_i = \mathcal{A}(P_i)$, on a alors

$$\begin{aligned}\overrightarrow{O'B'(t)} &= \sum_{i=0}^{n} \mathcal{B}_{i,n}(t).\mathcal{L}(P_i) \\ &= \sum_{i=0}^{n} \mathcal{B}_{i,n}(t).\left(\mathcal{A}(P_i) - \mathcal{A}(O)\right) \\ &= \sum_{i=0}^{n} \mathcal{B}_{i,n}(t).\overrightarrow{O'P'_i}\end{aligned}$$

■

Nous donnons à présent quelques propriétés permettant de se faire une idée plus précise de la forme d'une courbe de Bézier :

Proposition 7.2.6 *Une courbe de Bézier appartient à l'enveloppe convexe du réseau de points.*

Preuve. La preuve est laissée au lecteur.

■

Proposition 7.2.7 *La dérivée de B vaut :*

$$B'(t) = n \sum_{i=0}^{n-1} \mathcal{B}_{i,n-1}(t)\left(P_{i+1} - P_i\right).$$

Preuve. On a vu (preuve de la proposition 7.2.3) que

- $\mathcal{B}'_{0,n}(t) = -n(1-t)^{n-1}$
- $\mathcal{B}'_{n,n}(t) = nt^{n-1}$

Par définition de $\mathcal{B}$, on a donc :

- $\mathcal{B}'_{0,n}(t) = -n\mathcal{B}_{0,n-1}(t)$
- $\mathcal{B}'_{n,n}(t) = n\mathcal{B}_{n-1,n-1}(t)$

De même, on a vu que, pour $i \in]0, n[$, on a

$$\begin{aligned}\mathcal{B}'_{i,n}(t) &= \frac{n!}{i!(n-i)!}\left[it^{i-1}(1-t)^{n-i} - t^i(n-i)(1-t)^{n-i-1}\right] \\ &= n\left[\frac{(n-1)!}{(i-1)!(n-i)!}t^{i-1}(1-t)^{n-i} - \frac{(n-1)!}{(i)!(n-i-1)!}t^i(1-t)^{n-i-1}\right] \\ &= n\left[\mathcal{B}_{i-1,n-1}(t) - \mathcal{B}_{i,n-1}(t)\right]\end{aligned}$$

d'où

$$\begin{aligned}B'(t) &= \mathcal{B}'_{0,n}(t)P_0 + \sum_{i=1}^{n-1}\mathcal{B}'_{i,n}(t)P_i + \mathcal{B}'_{n,n}(t)P_n \\ &= -n\mathcal{B}_{0,n-1}(t)P_0 - n\sum_{i=1}^{n-1}\mathcal{B}_{i,n-1}(t)P_i + n\sum_{i=1}^{n-1}\mathcal{B}_{i-1,n-1}(t)P_i + n\mathcal{B}_{n-1,n-1}(t)P_n\end{aligned}$$

$$
\begin{aligned}
&= n\sum_{i=0}^{n-1}\mathcal{B}_{i,n-1}(t)P_{i+1} - n\sum_{i=0}^{n-1}\mathcal{B}_{i,n-1}(t)P_i \\
&= n\sum_{i=0}^{n-1}\mathcal{B}_{i,n-1}(t)\left(P_{i+1}-P_i\right)
\end{aligned}
$$

■

Nous finissons avec une remarque :

Remarque 7.2.1 Le déplacement d'un point de contrôle affecte toute la courbe ! (Fig. 7.3)

7.3 Les courbes B-splines

7.3.1 Définition

On définit les courbes B-splines de la même façon que les courbes de Bézier mais en utilisant d'autres fonctions de pondération que celles de Bernstein, on les nommera *fonctions de B-splines*. Ces fonctions ont la particularité de n'être non nulles que sur une partie d'un segment. On associe ainsi une fonction à chaque point de contrôle. Le déplacement d'un point de contrôle aura alors un effet local. De plus, les fonctions de B-splines permettent de contrôler le degré du polynôme obtenu indépendamment du nombre de points de contrôle (ce qui n'est pas le cas pour les courbes de Bézier).

Définition :
Soient $P_i, i = 0 \dots n$, $n+1$ points de contrôles dans $\mathbb{R}^2$.
La courbe B-spline de degré k approchant le réseau de points $\{P_i\}_{i\in\{0,n\}}$ est définie par :

$$BS_k(t) = \sum_{i=0}^{n}\mathcal{S}_{i,k}(t).P_i$$

où $\mathcal{S}_{i,k}(t)$ est la fonction de B-spline de degré k associée au point de contrôle i.

On définit la fonction de B-spline de façon récursive :

$$\mathcal{S}_{i,0}(t) = \begin{cases} 1 \text{ si } t \in [v_i, v_{i+1}[\\ 0 \text{ sinon} \end{cases}$$

et

$$\mathcal{S}_{i,k}(t) = \frac{(t-v_i)\mathcal{S}_{i,k-1}(t)}{v_{i+k}-v_i} + \frac{(v_{i+k+1}-t)\mathcal{S}_{i+1,k-1}(t)}{v_{i+k+1}-v_{i+1}}$$

où v_i est la i$^{\text{ème}}$ composante du vecteur nodal.

Le vecteur nodal permet de gérer, indépendamment du nombre de points de contrôle, le degré de la fonction B-spline. Il est composé de valeurs réelles croissantes. Des valeurs identiques dans le vecteur nodal peuvent être interprétées comme la présence de points de contrôle multiples. Nous considérerons, par la suite, une répartition équidistante des points de contrôle sur l'axe t, le vecteur sera composé de valeurs entières.

7.3.2 Propriétés

Nous présentons quelques propriétés des courbes B-spline, les démonstrations sont laissées au lecteur :

- Toute courbe B-spline commence (resp. termine) par un point sur la première (resp. dernière) arête de la ligne polygonale de contrôle et est tangente à celle-ci.
- Les points intermédiaires sont des points de contrôle externes à la courbe.
- La transformée affine d'une courbe B-spline est la courbe passant par la transformée des points.

Les propriétés suivantes donnent une idée de la forme de la B-spline en fonction de son degré (Fig. 7.4) :

- Le premier et le dernier point de contrôle doivent être de multiplicité égale à $k+1$ (où k est le degré) pour que la courbe commence et termine aux points P_0 et P_n.
- Si le degré est 0, on obtient $\cup_{i=0}^{n-1}\{P_i\}$.
- Si le degré est 1, on obtient la ligne polygonale.
- Si le degré de la B-spline est égal au nombre de côtés de la ligne polygonale et s'il n'y a pas de points doubles, la courbe est une courbe de Bézier.

7.4 Les courbes NURBS

Les NURBS sont une généralisation des B-splines (on ajoute des poids aux points de contrôle).

7.4.1 Définition

Définition
Une NURBS (Non Uniform Rational B-Spline) d'ordre k est définie par :
- Un vecteur nodal
- $n+1$ points de contrôle P_i
- $n+1$ poids w_i
- $n+1$ fonctions de pondération $\mathcal{N}_{i,k}$ déduites des fonctions de B-spline $\mathcal{S}_{i,k}$ au moyen des poids w_i :

$$\mathcal{N}_{i,k}(t) = \frac{w_i \mathcal{S}_{i,k}(t)}{\sum_{j=0}^{n} w_j \mathcal{S}_{j,k}(t)}$$

7.4.2 Propriétés

Les preuves sont laissées au lecteur.
Les propriétés suivantes rendent compte des possibilités qu'offrent les NURBS :
- En choisissant correctement les points de contrôle et les poids, toute conique peut-être exactement représentée par des NURBS.
- La transformée affine d'une NURBS est la courbe passant par la transformée des points.
- L'image d'une NURBS par une projection est la NURBS approchant la projection des points (les poids sont, bien sûr, différents de la NURBS d'origine).

7.4.3 De l'intérêt des coordonnées homogènes

On peut voir les NURBS comme étant des B-splines dans un système de coordonnées homogènes. Soit une B-spline de degré k

$$BS(t) = \sum_{i=0}^{n} \mathcal{S}_{i,k}(t) P_i$$

où les points de contrôle sont en coordonnées homogènes :

$$P_i = \begin{pmatrix} w_i x_i \\ w_i y_i \\ w_i z_i \\ w_i \end{pmatrix}$$

On a alors

$$BS(t) = \sum_{i=0}^{n} \mathcal{S}_{i,n}(t) \begin{pmatrix} w_i x_i \\ w_i y_i \\ w_i z_i \\ w_i \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n} \mathcal{S}_{i,n}(t) w_i x_i \\ \sum_{i=0}^{n} \mathcal{S}_{i,n}(t) w_i y_i \\ \sum_{i=0}^{n} \mathcal{S}_{i,n}(t) w_i z_i \\ \sum_{i=0}^{n} \mathcal{S}_{i,n}(t) w_i \end{pmatrix}$$

Ce qui nous donne en coordonnées cartésiennes (on divise par la dernière coordonnée) :

$$BS(t)_{cart} = \sum_{i=0}^{n} \frac{\mathcal{S}_{i,k}(t) w_i}{\sum_{j=0}^{n} \mathcal{S}_{j,k}(t) w_j} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \sum_{i=0}^{n} \frac{\mathcal{S}_{i,k}(t) w_i P_{i,cart}}{\sum_{j=0}^{n} \mathcal{S}_{j,k}(t) w_j}$$

7.4.4 Exemple

Nous avons vu que toute conique peut-être représenté par une NURBS. Nous donnons ici un moyen (parmi d'autres) de représenter le cercle unité par une NURBS.

Pour cela on utilise neuf points de contrôle $P_0, \ldots, P_9$ comme indiqué Fig. 7.5 de poids 1 pour les points d'indice pairs et $\frac{1}{\sqrt{2}}$ pour ceux d'indices impairs.

Afin que la courbe passe par les points d'indices pairs, il faut choisir comme vecteur nodal : $(0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4)$. Le degré de la NURBS devant être de deux.

7.5 Les courbes splines

7.5.1 Définition

Nous définissons dans dans cette section les **splines cubiques interpolées**. Ces courbes simulent le comportement de la "latte du jardinier" : on dispose de $n + 1$ piquets sur lesquels on appuie une mince planche de bois (Fig. 7.6).

La spline S est alors représentée par un polynôme de degré trois entre chaque segment $[P_i, P_{i+1}]$ (où $(P_i)_{i \in [0,n]}$ est le réseau de points à interpoler) :

$$S(t) = \bigcup_{i=0}^{n-1} \left\{ B_{i,0} + B_{i,1} t + B_{i,2} t^2 + B_{i,3} t^3 \mid t \in [0, 1] \right\}$$

Les cœfficients $B_{i,j}$ sont tels que :
- La courbe passe par les points de contrôle.
- Elle est de classe $\mathcal{C}^1$ en P_i pour $i \in [0, n]$ et de classe $\mathcal{C}^2$ aux autres points.

7.5.2 Calcul des cœfficients

Montrons que si on se donne les tangentes en chaque point de contrôle ces conditions suffisent à définir entièrement la courbe.

Considérons la courbe entre les points P_k et P_{k+1}, $k \in [0, n-1]$. Soient T_k, T_{k+1} les tangentes souhaitées aux points P_k, P_{k+1}.

Les conditions impliquent alors :

$$\begin{cases} S_k(0) = B_{k,0} = P_k \\ S_k(1) = B_{k,0} + B_{k,1} + B_{k,2} + B_{k,3} = P_{k+1} \\ S'_k(0) = B_{k,1} = T_k \\ S'_k(1) = B_{k,1} + 2B_{k,2} + 3B_{k,3} = T_{k+1} \end{cases}$$

On en déduit, après résolution du système :

$$\begin{cases} B_{k,0} = P_k \\ B_{k,1} = T_k \\ B_{k,2} = 3\left(P_{k+1} - P_k\right) - 2T_k - T_{k+1} \\ B_{k,3} = 2\left(P_k - P_{k+1}\right) + T_k + T_{k+1} \end{cases}$$

7.5.3 Un cas particulier : les courbes de Catmull-Rom

Nous avons vu que pour définir une spline il suffit de se fixer les tangentes en chaque point de contrôle. Dans le cas des courbes de Catmull-Rom on se donne des conditions supplémentaires permettant de connaître ces tangentes aux points intérieurs à la courbes :

La tangente T_k au point de contrôle P_k, $k \in]0, n[$ est parallèle au segment $[P_{k-1}, P_{k+1}]$:

$$T_k = m\left(P_{k+1} - P_{k-1}\right).$$

On choisit souvent $m = \frac{1}{2}$.

Ainsi, il ne reste plus qu'à fixer T_0 et T_n pour pouvoir calculer la spline. En pratique, deux cas sont généralement privilégiés : l'encastrement et la relaxation (Fig 7.7).
- L'encastrement : on fixe T_0 et T_n indépendamment des points de contrôle.
- La relaxation : on ajoute comme condition $S''_0(0) = 0$ et $S''_{n-1}(1) = 0$.

7.6 Les surfaces

Nous ne présentons dans cette section que quelques définitions, la généralisation des propriétés vu précédemment est laissée au lecteur. La généralisation des courbes interpolées aux surfaces ne donne généralement pas de résultats satisfaisants (problème sur les côtés, difficultés d'obtenir la surface désirée en procédant par étapes successives). Aussi nous nous restreindrons dans ce cours à la généralisation des courbes approchées.

7.6.1 Surfaces de Bézier et surface de B-splines

La généralisation des courbes aux surfaces se fait de façon naturelle :
Soit $P_{i,j}$, $i \in [0,n]$, $j \in [0,m]$ un ensemble de points de $\mathbb{R}^3$. On cherche alors la surface S approximant ce réseau de point.

Soient $\mathcal{M}_{i,k}(t)$ et $\mathcal{M}_{j,l}(s)$ les fonctions de pondérations (Bézier ou B-spline), on définit alors la surface par :

$$S(t,s) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathcal{M}_{i,k}(t)\mathcal{M}_{j,l}(s)P_{i,j},$$

7.6.2 Surfaces NURBS

Pour les surfaces, on définit les fonctions de pondérations de la manière suivante :

$$\mathcal{N}_{i,j,k,l}(t,s) = \frac{w_{i,j}\mathcal{S}_{i,k}(t)\mathcal{S}_{j,l}(s)}{\sum_{u,v} w_{u,v}\mathcal{S}_{u,k}(t)\mathcal{S}_{v,l}(s).}$$

où $w_{i,j}$ est le poids associé au point $P_{i,j}$.

La surface NURBS est alors :

$$N(t,s) = \sum_{i,j} \mathcal{N}_{i,j,k,l}(t,s)P_{i,j}.$$

La figure 7.8 représente une surface NURBS dont les paramètres sont les suivants :

- Points de contrôles :
  $P_{0,0} = (-1,1,1), P_{0,1} = (-1,0,1), P_{0,2} = (0,0,1), P_{0,3} = (1,0,1), P_{0,4} = (1,1,1)$
  $P_{1,0} = (-1,0.5,0.5), P_{1,1} = (-1,0,0.5), P_{1,2} = (0,0,0.5), P_{1,3} = (1,0,0.5), P_{1,4} = (1,0.5,0.5)$
  $P_{2,0} = (-1,0,0), P_{2,1} = (-1,0,0), P_{2,2} = (0,0,0), P_{2,3} = (1,0,0), P_{2,4} = (1,0,0)$
  $P_{3,0} = (-1,-0.5,-0.5), P_{3,1} = (-1,0,-0.5), P_{3,2} = (0,0,-0.5), P_{3,3} = (1,0,-0.5), P_{3,4} = (1,-0.5,-0.5)$
  $P_{4,0} = (-1,-1,-1), P_{4,1} = (-1,0,-1), P_{4,2} = (0,0,-1), P_{4,3} = (1,0,-1), P_{4,4} = (1,-1,-1)$
- Poids :
  $w_{i,j} = 1$ pour j pair et
  $w_{i,j} = \frac{1}{\sqrt{2}}$ sinon.
- Vecteurs nodaux selon t et s :

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \end{pmatrix}$$

7.7 Où l'on parle d'OpenGL

OpenGL permet le rendu de courbes et les surfaces de Bézier, la GLU, quant-à elle, permet de gérer les NURBS (et donc les B-splines).

7.7.1 Les courbes/surfaces de Bézier

Les courbes de Bézier

Définition

Pour spécifier quel sera la courbe à dessiner on utilise la fonction :

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

où

- `target` : évaluateur. Indique le type des points de contrôle (par exemple : coordonnées dans $\mathbb{R}^3$ ou couleur RGBA du point) contenu dans le tableau points. Les valeurs possibles sont
  - `GL_MAP1_VERTEX_3`
  - `GL_MAP1_VERTEX_4`
  - `GL_MAP1_INDEX`
  - `GL_MAP1_COLOR_4`
  - `GL_MAP1_NORMAL`
  - `GL_MAP1_TEXTURE_COORD_1`
  - `GL_MAP1_TEXTURE_COORD_2`
  - `GL_MAP1_TEXTURE_COORD_3`
  - `GL_MAP1_TEXTURE_COORD_4`
- `u1`, `u2` : intervalle dans lequel le paramètre de la fonction varie (dans le cours on a choisi $[0, 1]$).
- `stride` : nombre de valeurs séparant chaque bloc de données dans `points` (par exemple, si `points` ne contient que des coordonnées cartésiennes, `stride`=3).
- `order` : ordre de la courbe (= degré+1)
- `points` : pointeur sur un tableau de points de contrôle.

Il faut de plus activer l'évaluateur correspondant par :

```
void glEnable(GLenum target);
```

Calcul d'un point sur la courbe

```
void glEvalCoord1{fd}(TYPE u);
```

```
void glEvalCoord1{fd}v(TYPE *u);
```

évalue la fonction en `u` et génère les `glColor`, ... `glVertex`, correspondants.

Calcul d'un ensemble de points sur la courbe

On doit d'abord définir une grille d'évaluation :

```
void glMapGrid1{fd}(GLint un, TYPE u1, TYPE u2);
```

Définit une grille d'évaluation allant de `u1` à `u2` en `un` étapes.

On peut ensuite appliquer cette grille à la fonction :

```
void glEvalMesh1(GLenum mode, GLint i1, GLint i2);
```

applique la grille à tous les évaluateurs actifs. `mode` vaut `GL_POINT` ou `GL_LINE`, indique de quelle façon doivent-être reliés les points calculés. `i1` et `i2` indiquent les étapes pour lesquelles on évalue la fonction.

Ceci correspond *grosso modo* à :

```
glBegin(mode);
  for (i=i1;i<=i2;i++) {
    glEvalCoord1(u1+i*(u2-u1)/un);
glEnd();
```

Surfaces de Bézier

Le principe est le même, il « suffit » de remplacer `MAP1` par `MAP2`.

```
void glMap2{fd}(xGLenum target,
                GLdouble u1, GLdouble u2, GLint ustride, GLint uorder,
                GLdouble v1, GLdouble v2, GLint vstride, GLint vorder,
                const TYPE *points);
```

Ici on a deux paramètres `u` et `v` à configurer. Le reste est identique à `glMap1`.

```
void glEvalCoord2{fd}(TYPE u, TYPE v);
```

```
void glEvalCoord2{fd}v(TYPE *values);
```

```
void glMapGrid2{fd}(GLint un, TYPE u1, TYPE u2, GLint vn, TYPE v1,
TYPE v2);
```

```
void glEvalMesh2(GLenum mode, GLint i1, GLint i2, GLint j1, GLint j2);
```

`mode` peut ici prendre la valeur `GL_FILL`.

7.7.2 Les courbes/surfaces NURBS

Création

On utilise tout d'abord la fonction `gluNewNurbsRenderer` pour créer un objet permettant d'afficher une (ou plusieurs) NURBS.

```
GLUnurbsObj* gluNewNurbsRenderer(void);
```

Pour le détruire on utilisera :

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

Il faut ensuite fixer les propriétés d'affichage :

```
void gluNurbsProperty(GLUnurbsObj *nobj, GLenum property, GLfloat value
);
```

`value` est la valeur qu'on souhaite donner à `property`. `property` peut prendre les valeurs suivantes (se reporter à la spécification pour plus de détails) :

- `GLU_SAMPLING_METHOD` : indique les propriétés du maillage, les valeurs possibles sont alors :
  `GLU_PATH_LENGTH`, `GLU_PARAMETRIC_ERROR` ou `GLU_DOMAIN_DISTANCE`.
  - `GLU_PATH_LENGTH` : (par défaut) la longueur maximale (en pixels) des cotés des polygônes n'est pas plus grande que la valeur de `GLU_SAMPLING_TOLERANCE`.
  - `GLU_PARAMETRIC_ERROR` : la distance (en pixel) entre la surface et les polygônes l'approximant n'est pas plus grande que `GLU_PARAMETRIC_TOLERANCE`.
  - `GLU_DOMAIN_DISTANCE` : permet de spécifier le nombre de points selon les axes u et v qui sont pris par unité de longueur.
- `GLU_SAMPLING_TOLERANCE` : voir `GLU_PATH_LENGTH`. Valeur par défaut : 50.
- `GLU_PARAMETRIC_TOLERANCE` : voir `GLU_PARAMETRIC_ERROR`. Valeur par défaut : 0.5.
- `GLU_U_STEP` : voir `GLU_DOMAIN_DISTANCE`. Valeur par défaut : 100.
- `GLU_V_STEP` : voir `GLU_DOMAIN_DISTANCE`. Valeur par défaut : 100.
- `GLU_DISPLAY_MODE` : valeurs possibles :
  `GLU_FILL (par défaut)`, `GLU_OUTLINE_POLYGON` ou `GLU_OUTLINE_PATCH`.
- `GLU_CULLING` : `GL_TRUE` indique que la NURBS n'est pas calculée si ses points de contrôles sont en dehors du viewport. Par défaut : `GL_FALSE` puisqu'une NURBS n'est pas nécessairement contenue dans son enveloppe convexe.
- `GLU_AUTO_LOAD_MATRIX` : `GL_TRUE` (par défaut) ou `GL_FALSE`. Permet de spécifier quelles matrices utiliser (celle du serveur ou celles spécifiées par `gluLoadSamplingMatrices(GL_FALSE)`

Affichage

On inclut les commandes d'affichages entre

```
void gluBeginSurface(GLUnurbsObj *nobj);
void gluEndSurface(GLUnurbsObj *nobj);
```

ou

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

Pour afficher une NURBS on utilise :

```
void gluNurbsCurve(GLUnurbsObj *nobj, GLint uknots_count, GLfloat *uknot,
GLint u_stride, GLfloat *ctlarray, GLint uorder, GLenum type );
```

ou

```
void gluNurbsSurface(GLUnurbsObj *nobj, GLint uknot_count, GLfloat
*uknot, GLint vknot_count, GLfloat *vknot, GLint u_stride, GLint v_stride,
GLfloat *ctlarray, GLint uorder, GLint vorder, GLenum type);
```

- `nobj` : indique l'objet NURBS.
- `uknot_count` : indique le nombre de nœuds pour la direction u (t dans le cours).
- `uknot` : indique un tableau de `uknot_count` nœuds.
- `vknot_count` : idem que `uknot_count` pour la direction v (s dans le cours).
- `vknot` : idem que `uknot` pour la direction v.
- `u_stride` : nombre de valeurs séparant chaque points dans le tableau `ctlarray` pour la direction u.
- `v_stride` : idem que `ustride` pour la direction v.
- `ctlarray` : tableau contenant les points de contrôle de la NURBS .
- `uorder` : ordre (degré+1) de la NURBS dans la direction u.
- `vorder` : idem que `uorder` pour la direction v.
- `type` : par exemple `GL_MAP2_VERTEX_3` ou `GL_MAP2_COLOR_4` (voir courbes et surfaces de Bézier).

Remarque 7.7.1 Le nombre de points dans le tableau `ctlarray` n'est pas indiqué. Il est calculé par la formule suivante : (nombre de nœuds dans la direction u - ordre dans la direction u)*(nombre de nœuds dans la direction v - ordre dans la direction v). Les NURBS gérées par la GLU sont donc un cas particulier de celles vues dans le cours.

Gestion des erreurs

Il est possible de définir la fonction que sera appelée lorsqu'une erreur se produira dans la gestion des NURBS. Nous invitons le lecteur à se reporter à la spécification pour plus de détails, en attendant il pourra utiliser :

```
gluNurbsCallBack(theNurb, GLU_ERROR, nurbsError);
```

où la fonction `nurbsError` sera :

```
void CALBBACK nurbsError(GLenum codeErreur) {
    const GLubyte *string;

    string=gluErrorString(codeErreur);
    fprintf(stderr, ``Erreur dans une Nurbs : %s\n``, string);
    exit(0);
}
```

7.7.3 Un exemple de NURBS avec OpenGL

La surface de la figure 7.8 a été réalisée avec le programme suivant :

```
#include<GL/glut.h>
#include<stdlib.h>
#include<math.h>
#include<stdio.h>

#ifndef CALLBACK
#define CALLBACK
#endif
#define RAC2 1.41421356237309504880168872420969
#define PI 3.14159265358979323846264338327950
#define POIDS1 1
#define POIDS2 (PI/2)

/* Lumière */
float blanc[4]={.7,.7,.7,1};
float gris[4]={0.,.7,0.,1.};
float noir[4]={0.,.3,0.,1.};
float nulle[4]={0.,0.,0.,1.};
float pos_lum[4]={0.,10.,0,1.};
float dir_lum[3]={0.,-1.,0.};
/* Points de controles */
GLfloat pointControle[5][5][4]={
    {{-1*POIDS1,POIDS1,POIDS1,POIDS1},
     {-1*POIDS2,0*POIDS2,POIDS2,POIDS2},
     {0*POIDS1,0*POIDS1,POIDS1,POIDS1},
     {1*POIDS2,0*POIDS2,POIDS2,POIDS2},
     {1*POIDS1,1*POIDS1,POIDS1,POIDS1}},
    {{-1*POIDS1,.5*POIDS1,.5*POIDS1,POIDS1},
     {-1*POIDS2,0*POIDS2,0.5*POIDS2,POIDS2},
     {0*POIDS1,0*POIDS1,0.5*POIDS1,POIDS1},
     {1*POIDS2,0*POIDS2,0.5*POIDS2,POIDS2},
     {1*POIDS1,.5*POIDS1,0.5*POIDS1,POIDS1}},
    {{-1*POIDS1,0*POIDS1,0,POIDS1},
     {-1*POIDS2,0*POIDS2,0,POIDS2},
     {0*POIDS1,0*POIDS1,0,POIDS1},
     {1*POIDS2,0*POIDS2,0,POIDS2},
     {1*POIDS1,0*POIDS1,0,POIDS1}},
    {{-1*POIDS1,-.5*POIDS1,-0.5*POIDS1,POIDS1},
     {-1*POIDS2,0*POIDS2,-0.5*POIDS2,POIDS2},
     {0*POIDS1,0*POIDS1,-0.5*POIDS1,POIDS1},
     {1*POIDS2,0*POIDS2,-0.5*POIDS2,POIDS2},
     {1*POIDS1,-.5*POIDS1,-0.5*POIDS1,POIDS1}},
    {{-1*POIDS1,-POIDS1,-POIDS1,POIDS1},
     {-1*POIDS2,0*POIDS2,-POIDS2,POIDS2},
```

```
     {0*POIDS1,0*POIDS1,-POIDS1,POIDS1},
     {1*POIDS2,0*POIDS2,-POIDS2,POIDS2},
     {1*POIDS1,-1*POIDS1,-POIDS1,POIDS1}};
/* NURBS */
GLUnurbsObj *theNurb;
/* Vecteurs nodaux */
GLfloat noeudu[8]={0,0,0,1,1,2,2,2};
GLfloat noeudv[8]={0,0,0,1,1,2,2,2};

double degRad(double angle) {
  return angle*PI/180;
}

void CALLBACK nurbsError(GLenum errorCode) {
const GLubyte *estring;

estring = gluErrorString(errorCode);
fprintf(stderr, "Nurbs Error: %s\n", estring);
exit(0);
}

void display(void) {
  int i,j;

  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glEnable(GL_LIGHTING);
  glLightfv(GL_LIGHT0, GL_POSITION, pos_lum);
  glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir_lum);
  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, blanc);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, blanc);
  glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, 100);
  /* Surface */
  gluBeginSurface(theNurb);
  gluNurbsSurface(theNurb, 8, noeudu, 8, noeudv, 4, 5*4,
                  &pointControle[0][0][0], 3, 3,
                  GL_MAP2_VERTEX_4);
  gluEndSurface(theNurb);
  glutSwapBuffers();
}

void myInit(void) {

  glClearColor(1.,1.,1.,1.0);  /* Couleur du fond (RGBA) */
  glEnable(GL_DEPTH_TEST);
  /* On génère les normales automatiquement */
  /* (on n'a pas vu comment faire autrement ...) */
  glEnable(GL_AUTO_NORMAL);
```

```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum(-1.1,1.1,-1.1,1.1,90,140);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(0,0,100,0,0,0,0,1,0);
  /* Définition du NurbsRenderer */
  theNurb=gluNewNurbsRenderer();
  gluNurbsProperty(theNurb, GLU_SAMPLING_TOLERANCE, 5.0);
  gluNurbsProperty(theNurb, GLU_DISPLAY_MODE, GLU_OUTLINE_POLYGON);
  gluNurbsCallback(theNurb, GLU_ERROR, nurbsError);
  /* Définition de la lumière */
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_AMBIENT, gris);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, blanc);
  glLightfv(GL_LIGHT0, GL_SPECULAR, gris);
  glLighti(GL_LIGHT0, GL_SPOT_CUTOFF, 160);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, nulle);
  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
}

int main(int argc, char *argv[])
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGBA|GLUT_DEPTH);
  glutInitWindowSize(400,400);
  glutInitWindowPosition(0,0);
  glutCreateWindow("Surface NURBS");
  myInit();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

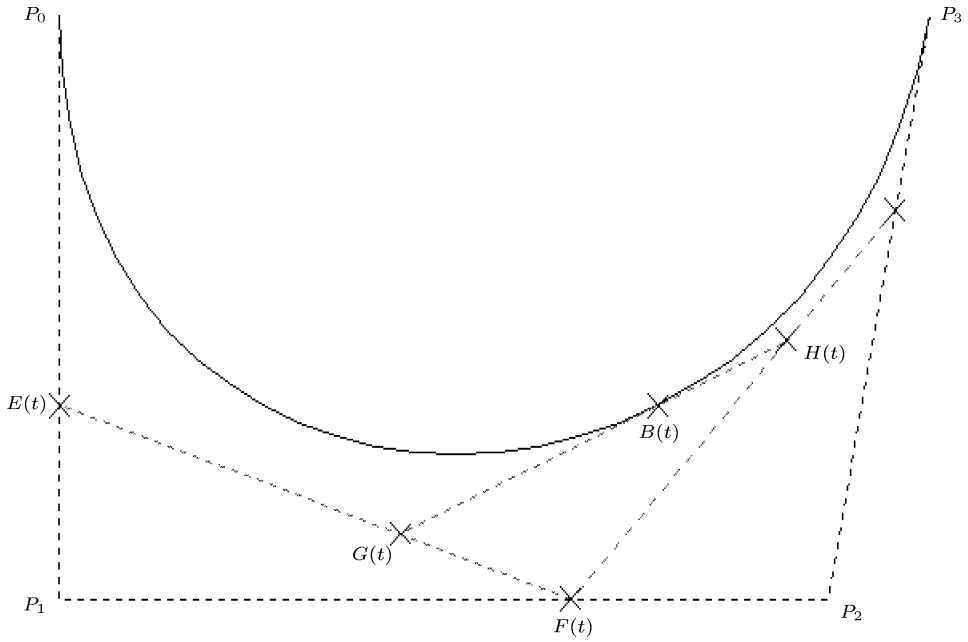

FIG. 7.1 – *Courbe de Bézier d'ordre 3 (4 points)*

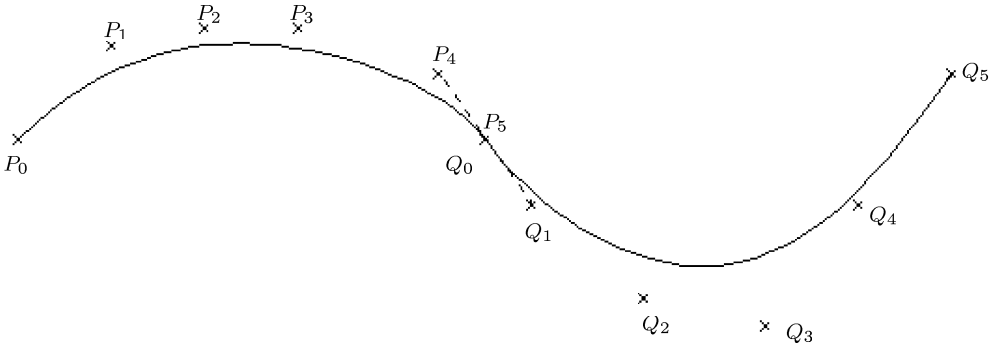

FIG. 7.2 – *Composition de deux courbes de Bézier*

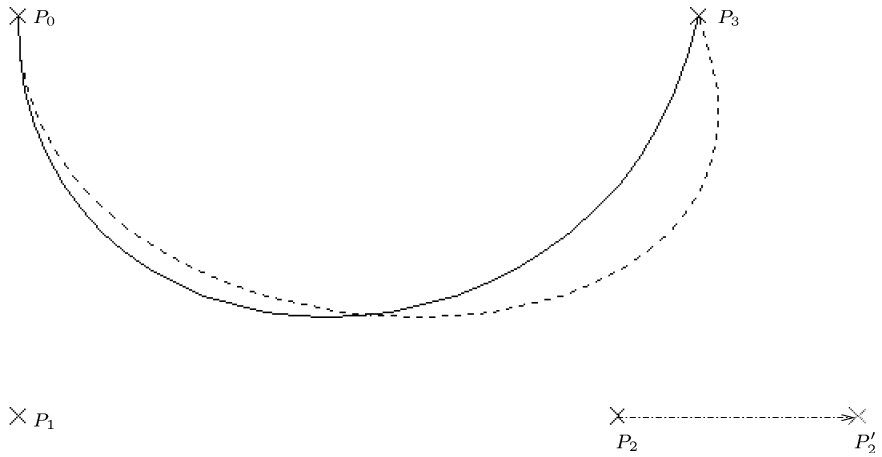

FIG. 7.3 – *Effet du déplacement d'un point de contrôle*

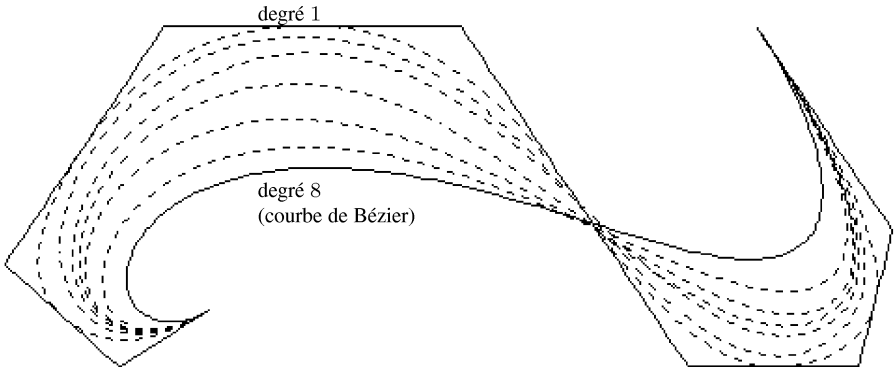

FIG. 7.4 – *Influence du degré sur la forme d'une B-spline*

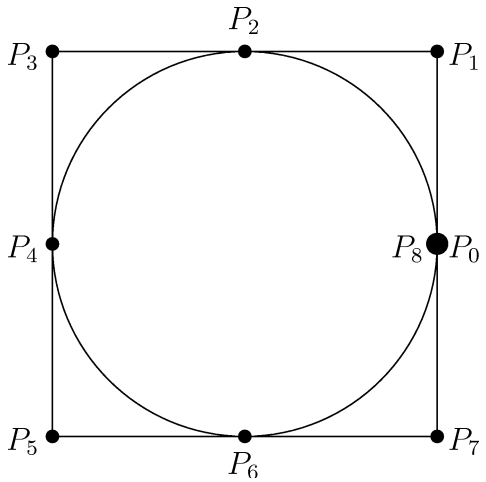

FIG. 7.5 – *représentation d'un cercle par une NURBS*

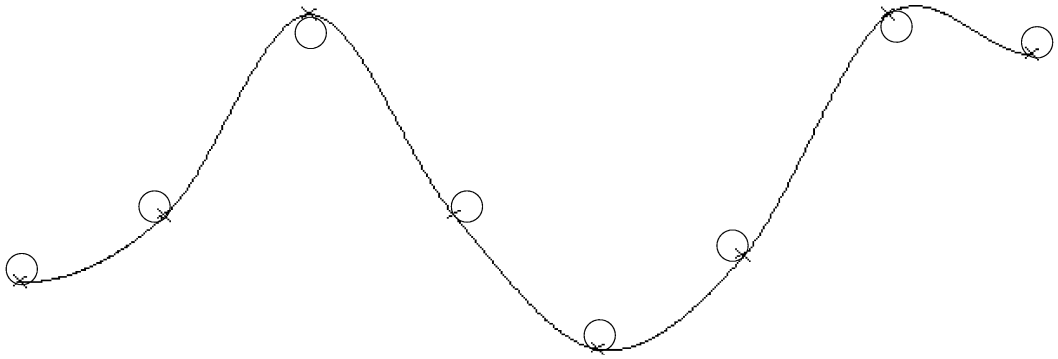

FIG. 7.6 – *La latte du jardinier*

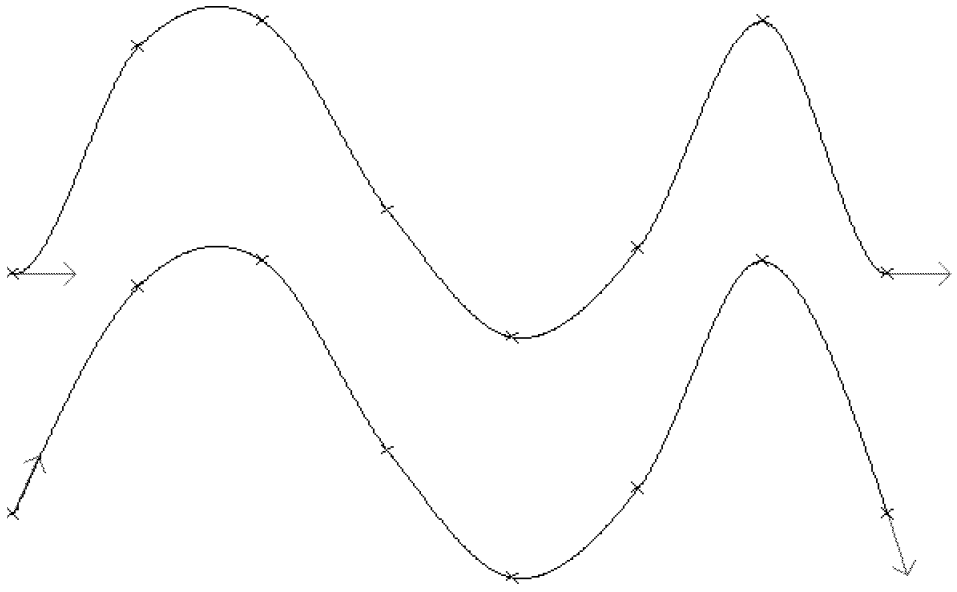

FIG. 7.7 – *Spline encastrée et spline relaxée*

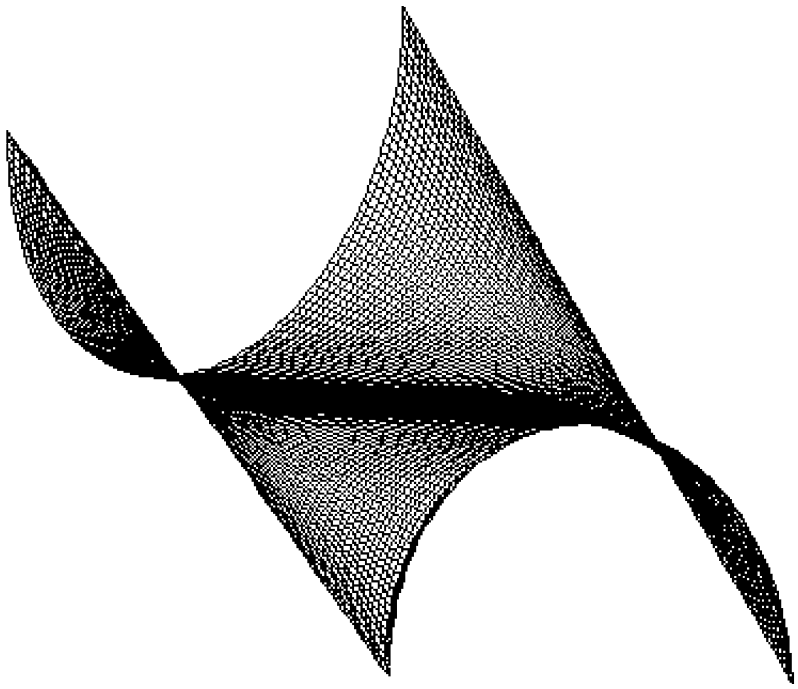

FIG. 7.8 – *Un exemple de surface NURBS*