Utilisation des déformations pour la modélisation des solides de forme libre en synthèse d'images

Zhigang Nie

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre : 59 ID

THESE

présentée par

Zhigang NIE

pour obtenir le titre de

DOCTEUR

DE L'UNIVERSITE DE SAINT-ETIENNE

ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Informatique)

Utilisation des déformations pour la modélisation des solides de forme libre en synthèse d'images

Soutenue à Saint-Etienne le 17 Octobre 1991

COMPOSITION DU JURY

Monsieur	B. PEROCHE	Président
Messieurs	R. CAUBET F. SCHMITT	Rapporteurs
Madame Mademoiselle Monsieur	Y. AHRONOVITZ S. COQUILLART J.C. LAFON	Examinateurs

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre : 59 ID

THESE

présentée par

Zhigang NIE

pour obtenir le titre de

DOCTEUR

DE L'UNIVERSITE DE SAINT-ETIENNE

ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Informatique)

Utilisation des déformations pour la modélisation des solides de forme libre en synthèse d'images

Soutenue à Saint-Etienne le 17 Octobre 1991

COMPOSITION DU JURY

Monsieur	B. PEROCHE	Président
Messieurs	R. CAUBET F. SCHMITT	Rapporteurs
Madame	Y. AHRONOVITZ	Examinateurs
Mademoiselle	S. COQUILLART	
Monsieur	J.C. LAFON	

Je tiens à remercier Monsieur le professeur Bernard PEROCHE, directeur du département informatique à l'Ecole des Mines de Saint Etienne, qui a consacré beaucoup de temps pour diriger ce travail, et qui me fait l'honneur de présider le jury.

Je voudrais remercier Monsieur le professeur René CAUBET de l'Université de Paul Sabatier à Toulouse, rapporteur, dont les remarques m'ont été précieuses.

Je tiens à remercier Monsieur le professeur Francis SCHMITT de l'Ecole de Télécommunication de Paris, qui a accepté d'être le rapporteur.

Je remercie Monsieur le professeur Jean Claude LAFON de CERISI à Sophia Antipolis, qui a accepté de faire partie du jury.

Je remercie Mademoiselle Sabine Coquillart, chargée de recherches à INRIA, qui a accepté de faire partie du jury.

Je remercie Madame Yolande AHRONOVITZ, maître de conférence à l'Université de Saint Etienne, qui a accepté de se joindre au jury.

Je remercie tous les membres passés et présents de l'équipe "communication visuelle", Sami AIT AOUDIA, Jacquline ARGENCE, Michel BEIGBEDER, Helmi BEN AMARA, Rabah BENOUAMER, Annie BOURGEAT, Gilles FERTEY, Gabriel HANOTEAUX, Ghassan JAHMI, François JAILLET, Philippe JAILLON, Roland JEGOU, Dominique MICHELUCCI, Jean Michel MOREAU, Marc ROELENS, Pascale ROUDIER.

Je voudrais aussi exprimer mon amitié à tous les membres du département. Je remercie l'ensemble du service reprographie de l'Ecole des Mines pour leur diligence qui m'a permis d'avoir ces exemplaires à temps.

Introduction

Du point de vue du traitement des informations, un système de synthèse d'images a pour but de transformer la modélisation d'un monde virtuel en images, pour aider l'utilisateur du système à mieux apprécier son monde virtuel.

Un système de synthèse d'images peut être caractérisé par le réalisme des images calculées, la rapidité de la production des images, la généralité de la modélisation supportée et la facilité de modélisation autorisée.

Depuis son apparition dans les années soixante, la synthèse d'images a conquis de nombreux domaines d'applications [PERO 88]. En mécanique, les images de synthèse permettent à un concepteur de voir une pièce avant sa fabrication. En architecture, un architecte peut utiliser les images de synthèse pour présenter son oeuvre, au lieu d'utiliser une maquette. La simulation de conduite utilise un système de synthèse d'images qui peut produire vingt-quatre images par seconde. En communication, un système de synthèse d'images peut produire des images qui seront difficiles à réaliser avec une caméra. De nouvelles applications apparaîtront vraisemblablement bientôt, suivant l'augmentation de performances et la baisse de prix des systèmes de synthèse d'images.

Pour la synthèse d'images, la modélisation consiste à décrire les propriétés optiques en chaque point du monde virtuel. Celles-ci déterminent en un point donné l'émission, la transmission et l'absorption de la lumière. Par exemple, pour modéliser un nuage, il faut donner en chaque point du ciel la densité d'humidité, qui est liée aux propriétés optiques en ce point. Pour la plupart des objets, la propriété optique est uniforme. Pour décrire un tel objet, on peut distinguer la modélisation géométrique qui décrit la géométrie de l'objet, et la modélisation optique qui décrit ses propriétés optiques.

En modélisation géométrique, la modélisation des solides et la modélisation des surfaces ont été développées plutôt indépendamment. La première consiste à modéliser des solides de forme régulière avec par exemple l'arbre de construction (Constructive Solid Geometry ou CSG en anglais). La deuxième est la modélisation des surfaces de forme libre en utilisant par exemple des surfaces paramétriques.

Il est important pour un système de synthèse d'images de permettre de modéliser des solides de formes différentes, du polyèdre au solide de forme libre. Pour modéliser un solide de forme libre, la méthode directe consiste à construire un modèle de courbes et interpoler les courbes avec des surfaces paramétriques. Le contrôle de la continuité entre les surfaces paramétriques pose des problèmes.

Certains solides de forme libre peuvent être considérés comme le résultat de déformation de solides de forme régulière. [BARR 84] a introduit pour la première fois la déformation dans la modélisation de solides. Quelques déformations globales comme la courbure, l'effilement et la torsion sont définis et permettent de créer beaucoup de solides de formes intéressantes.

[SEDE 86] a présenté une méthode pour déformer un objet, qui est baptisée déformation libre (Free Form Deformation en anglais). Autour de l'objet à déformer, on définit un repère local et un maillage de points de contrôle dans le repère local. Quand on déplace les points de contrôle, l'objet se trouvant au milieu du maillage se déforme en suivant le déplacement des points de contrôle. La déformation de l'objet est réalisée avec un changement de repère et une interpolation volumique de type Bézier.

Comme la déformation libre utilise l'interpolation de type Bézier, l'ordre d'interpolation dépend du nombre de points de contrôle utilisés. Pour réaliser une déformation complexe avec la déformation libre, il faut définir beaucoup de points de contrôle et effectuer une interpolation d'ordre important. On peut utiliser plusieurs déformations libres d'ordre cubique en séquence pour réaliser une déformation complexe. La continuité d'un objet entre les déformations libres est contrôlée par des contraintes sur la position relative entre les points de contrôle. Si on demande la continuité d'ordre deux entre les déformations libres, le déplacement d'un point de contrôle changera toute la déformation, et on perdra donc le contrôle local de la déformation. Nous proposons d'utiliser l'interpolation de type B-spline dans la formulation de la déformation libre. Avec cette nouvelle formulation, l'ordre d'interpolation devient indépendant du nombre de points de contrôle. Cette nouvelle formulation nécessite cependant un reparamétrage avant l'interpolation.

La déformation libre utilise un repère local cartésien et forme ainsi une région de déformation de forme cubique. Cette formulation ne paraît pas adéquate pour réaliser les déformations qui peuvent être définies comme une fonction tri-linéaire dans un repère cylindrique ou sphérique. Pour remédier à ce problème, nous proposons la déformation

libre cylindrique et la déformation libre sphérique qui utilisent respectivement un repère local cylindrique et un repère local sphérique.

La frontière d'un solide peut être décomposée en surfaces qui sont discontinues entre elles. Chaque surface peut en général être représentée comme une surface paramétrique. La déformation d'une surface paramétrique reste une surface paramétrique. La visualisation des solides déformés est donc le même problème que celle des surfaces paramétriques.

Une méthode simple pour visualiser une surface paramétrique est d'effectuer une facettisation pour l'approximer par des polygones. Ensuite, des algorithmes comme le balayage ligne par ligne ou le tampon de profondeur peuvent être utilisés pour effectuer l'élimination des parties cachées. La surface peut être plus courbée et exiger plus de polygones à certains endroits qu'ailleurs. Pour utiliser moins de polygones, tout en garantissant la précision d'approximation, il faut faire une facettisation adaptative. Chaque facette est testée suivant certains critères. Si les critères ne sont pas satisfaits, la facette sera subdivisée.

Une surface paramétrique peut être très courbée dans une direction et rester relativement plane dans l'autre. Si la subdivision de facettes n'a pas de direction privilégiée, on facettisera la surface paramétrique dans les deux directions et on produira par conséquent trop de facettes. Pour optimiser la facettisation adaptative, nous proposerons une subdivision directionnelle de facettes.

Si on cherche le réalisme de l'image, on peut visualiser une surface paramétrique en tracé de rayons. Le problème essentiel est de trouver l'intersection d'un rayon avec la surface. Si la surface est polynomiale, on peut calculer l'intersection en résolvant des équations algébriques.

Nous utilisons les déformations libres pour déformer des primitives. Leurs surfaces peuvent en général être représentées comme des fonctions polynomiales en coordonnées homogènes. Après déformation, elles resteront polynomiales. Nous pouvons donc calculer l'intersection d'un rayon avec les primitives déformées en résolvant des équations algébriques.

Nous utilisons les déformations libres sur l'arbre de construction comme une extension des transformations affines. Pour supporter cette transformation non-affine, il faut utiliser des structures de données différentes.

La définition d'une déformation libre nécessite de déplacer un nombre relativement important de points de contrôle. Il est donc important de définir une interface conviviale. Si la déformation libre est utilisée pour réaliser une déformation globale, nous pouvons utiliser une interface syntaxique, en appliquant la fonction de déformation aux points de contrôle pour les déplacer. Pour déplacer les points de contrôle de façon libre, l'interface syntaxique est difficile à utiliser. Nous présenterons une interface graphique utilisant un curseur 3D pour spécifier le déplacement des points de contrôle.

Si l'on s'intéresse non seulement au résultat final de la déformation, mais aussi au processus de déformation, on obtient une animation. L'animation cinématique d'un objet déformable peut être spécifiée en définissant le mouvement des points de contrôle.

Pour réaliser une animation réaliste, le processus de déformation doit obéir aux lois de la dynamique. Nous utilisons la déformation libre pour simplifier le calcul. Au lieu de déformer directement l'objet, nous déformons la région de déformation qui le contient en utilisant la dynamique. Les points de contrôle déplacés seront utilisés pour déformer l'objet.

Dans le premier chapitre, nous présenterons les travaux qui ont déjà été réalisés sur l'utilisation des déformations en synthèse d'images. Dans le deuxième chapitre, nous présenterons la formulation de la déformation libre avec l'interpolation de type B-spline. Dans le troisième chapitre, nous présenterons la formulation de la déformation libre avec un repère local cylindrique ou sphérique. La visualisation des solides déformés sera présentée dans le quatrième chapitre. Dans le cinquième chapitre, nous présenterons l'utilisation des déformations libres avec l'arbre de construction, l'interface de la déformation libre et l'animation à l'aide de la déformation libre.

1 Modélisation des objets de forme libre

Ce chapitre a pour but de présenter, d'une façon générale, les travaux qui ont été réalisés sur l'utilisation des déformations en synthèse d'images. Dans les deux premiers paragraphes, nous présenterons respectivement la modélisation des solides et la modélisation des surfaces de forme libre, qui sont deux domaines importants de la modélisation géométrique. Dans le troisième paragraphe, nous définirons d'une façon générale la déformation et présenterons quelques déformations globales. Nous rappellerons la formulation de la déformation libre dans le quatrième paragraphe. Dans le cinquième paragraphe, nous présenterons d'une façon générale la visualisation des solides déformés. Nous présenterons ensuite la facettisation adaptative des surfaces paramétriques dans le sixième paragraphe, et la visualisation en tracé de rayons des surfaces paramétriques dans le septième paragraphe.

1.1 Modélisation des solides

Pour faciliter différents traitements, un solide peut avoir différentes représentations. Se pose alors le problème de la conversion d'une représentation en une autre, problème qui n'est pas toujours facile à résoudre.

1.1.1 Arbre de construction

Avec la modélisation par **arbre de construction**, un solide est représenté par un arbre dont les feuilles sont des solides primitifs (cube, cylindre, sphère, ...), et dont les noeuds sont des opérations booléennes (union, intersection, différence).

Pour simplifier la description de l'arbre de construction, une primitive est définie avec une taille unitaire et dans un repère local. La primitive cube, par exemple, est un cube de côté un dont l'un des sommets est à l'origine du repère et qui se trouve entièrement dans la partie positive du repère. Une feuille sera donc composée d'une primitive et d'une transformation affine (translation, affinité, rotation). La transformation s'applique à tous les points de la primitive et produit un solide élémentaire de la taille voulue, à la position désirée dans le repère du monde.

A partir de cette définition essentielle, on peut avoir plusieurs variantes de l'arbre de construction au niveau de l'organisation :

- un sous-arbre partagé donne un graphe sans circuit,

- les transformations peuvent être affectées aux noeuds de l'arbre,

- un arbre non-binaire diminue la hauteur de l'arbre.

En utilisant l'arbre de construction, on peut construire, à partir des primitives simples, des solides très complexes.

Un arbre de construction décrit un solide implicitement. Pour visualiser le solide, il faut trouver sa représentation explicite : la représentation par frontière ou les voxels, par exemple.

1.1.2 Représentation par frontière

La **représentation par frontière** (Boundray Representation en anglais) décrit la frontière d'un solide. Une représentation par frontière mémorise les faces, les arêtes et les sommets, mais aussi les relations topologiques entre ces éléments. La validité d'une représentation par frontière n'est pas un problème trivial.

Les polygones sont beaucoup utilisés pour décrire la frontière des solides. Les opérations booléennes entre des polyèdres sont relativement simples. Si la frontière d'un solide contient des surfaces gauches, on peut toujours utiliser des polygones pour les approcher. Cette approximation a ses inconvénients : d'une part, elle demande beaucoup de polygones pour avoir une précision correcte; d'autre part, l'aliassage apparaît sur la silhouette quand on visualise les solides. Pour résoudre ces problèmes, on peut utiliser des surfaces paramétriques pour décrire les surfaces gauches des solides. En revanche, les opérations booléennes entre ces solides deviennent plus difficiles.

1.1.3 Représentation par voxels

La modélisation par **voxels** décrit un solide en discrétisant l'espace puis indiquant si le solide est présent ou pas en chaque point. Par rapport à la représentation par frontière, il est facile d'effectuer des opérations booléennes entre des solides décrits avec les voxels. En revanche, cette représentation demande beaucoup de mémoire, même si différentes techniques de compression de données ont déjà été proposées, comme l'utilisation de l'arbre octal. C'est la raison principale qui a limité jusqu'à présent l'utilisation en synthèse d'images de cette représentation à la fois générale et simple. Suivant la baisse du prix de la mémoire, cette représentation pourrait devenir plus compétitive.

1.2 Modélisation des surfaces de forme libre

L'arbre de construction est utilisé pour modéliser des solides de forme régulière. Pour modéliser des surfaces de forme libre, on peut utiliser des surfaces paramétriques comme les surfaces de Bézier, ou les surfaces B-splines.

1.2.1 Interpolation de type Bézier

Etant donnés des points $P_0, P_1, ..., P_n$, la courbe de Bézier les interpole comme suit :

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

où $$B_{i,n}(t) = C_n^i\ (1-t)^{n-i}\ t^i$$

sont des fonctions de mélange de Bernstein avec la convention $0^0 = 1$;

le paramètre t appartient à $[0, 1]$.

Sur tout le domaine du paramètre, les fonctions de mélange sont positives ou nulles. La somme des fonctions de mélange est égale à un. La courbe se trouve donc entièrement à l'intérieur de l'enveloppe convexe des points de contrôle.

La courbe passe par les deux points de contrôle extrêmes.

L'ordre d'interpolation est lié au nombre de points de contrôle. On peut utiliser plusieurs courbes cubiques en séquence pour définir une courbe plus complexe. La continuité entre deux courbes voisines peut être contrôlée en imposant des contraintes sur les points de contrôle. Pour avoir la continuité C^0, il faut que le dernier point de contrôle de la première courbe coïncide avec le premier point de contrôle de la seconde courbe. Pour avoir la continuité C^1, il faut en plus que les deux derniers points de contrôle de la première courbe et les deux premiers points de contrôle de la seconde courbe soient alignés.

Une surface de type Bézier est définie avec un réseau de points de contrôle P_{ij} comme suit :

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

1.2.2 Interpolation de type B-spline

Une courbe B-spline ayant pour points de contrôle $P_0, P_1, ..., P_n$ est définie par l'interpolation suivante :

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

où
$$N_{i,1}(t) = \begin{cases} 1 & si\ \ t_i \le t < t_{i+1} \\ 0 & sinon \end{cases}$$

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

avec la convention $\frac{0}{0} = 0$;

les noeuds $t_0, t_1, ..., t_{n+k}$ définissent le vecteur nodal qui contrôle, en plus des points de contrôle, l'interpolation ;

le paramètre t appartient à $[t_{k-1}, t_{n+1})$.

En général, la courbe de type B-spline a la continuité C^{k-2}. Si des noeuds multiples existent, la continuité sera dégradée.

Comme l'interpolation de type Bézier, sur le domaine défini du paramètre, les fonctions de mélange sont positives ou nulles et la somme des fonctions de mélange est égale à un. La courbe se trouve donc à l'intérieur de l'enveloppe convexe de l'ensemble des points de contrôle.

Au contraire de l'interpolation de type Bézier, l'ordre d'interpolation de type B-spline est indépendant du nombre des points de contrôle.

La fonction de mélange $N_{i,k}$ est nulle en dehors du domaine $[t_i, t_{i+k})$. La courbe peut donc être contrôlée localement.

On peut utiliser l'interpolation de type B-spline pour définir une courbe fermée. On l'appellera interpolation périodique, car les points de contrôle seront utilisés d'une façon périodique :

$$P(t) = \sum_{i=0}^{n+k-1} N_{i,k}(t)\, P_{mod(i,n)}$$

où la fonction $mod(i,n)$ donne le reste de la division de i par rapport à n,

le paramètre t appartient à $[t_{k-1}, t_{n+k})$.

La courbe de type B-spline uniforme utilise le vecteur nodal suivant :

$$t_i = i\,.$$

Les fonctions de mélange $N_{i,k}$ se déduisent alors les unes des autres par translation.

Une courbe de type B-spline uniforme ne passe pas par les deux points de contrôle extrêmes P_0 et P_n. Pour que la courbe passe par ces deux points comme la courbe de type Bézier, on utilise l'interpolation de type B-spline non-uniforme avec le vecteur nodal suivant [NEWM 79] :

$$t_i = \begin{cases} 0 & 0 \le i < k \\ i-k+1 & k \le i < n+1 \\ n-k+2 & n+1 \le i < n+k+1 \end{cases}$$

Le paramètre t appartiendra à $[0, n-k+2)$.

Une surface de type B-spline avec un réseau de points de contrôle P_{ij} est définie par :

$$P(s,t) = \sum_{i=0}^{m} N_{i,k}(s) \sum_{j=0}^{n} N_{j,l}(t) \; P_{ij}$$

1.3 Déformations

La modélisation des solides avec l'arbre de construction et la modélisation des surfaces de forme libre avec les surfaces paramétriques sont beaucoup utilisées pour la modélisation géométrique. En général, les deux modélisations sont bien distinguées à cause de la différence entre les théories qui les sous-tendent. Il est en revanche préférable pour un système de synthèse d'images de permettre à l'utilisateur de modéliser des solides de formes différentes, du polyèdre au solide de forme libre.

Pour modéliser un solide de forme libre, la méthode directe consiste à construire un modèle de courbe et à interpoler les courbes avec des surfaces paramétriques [CHIY 83]. Le contrôle de la continuité entre les surfaces paramétriques pose beaucoup de problèmes.

Certains solides de forme libre peuvent être considérés comme le résultat de déformation de solides de forme régulière. [BARR 84] a introduit pour la première fois la déformation dans la modélisation géométrique.

1.3.1 Définition

D'une façon générale, une déformation peut être définie comme une fonction de transformation :

$$P' = F(P)$$

où P est un point sur l'objet à déformer,
P' donne sa position après la déformation.

Pour visualiser un solide déformé, il faut transformer, en plus des points, les normales à la surface. Pour ce faire, on peut utiliser la matrice Jacobienne [BARR 84]. La transformée N' de la normale N est donnée par la formule suivante :

$$N' = det(J)\, J^{-1t}\, N$$

où

$$J = \begin{bmatrix} \frac{\partial x'}{\partial x} & \frac{\partial x'}{\partial y} & \frac{\partial x'}{\partial z} \\ \frac{\partial y'}{\partial x} & \frac{\partial y'}{\partial y} & \frac{\partial y'}{\partial z} \\ \frac{\partial z'}{\partial x} & \frac{\partial z'}{\partial y} & \frac{\partial z'}{\partial z} \end{bmatrix}$$

est la matrice Jacobienne liée à la fonction de transformation.

La formule peut être représentée sous forme explicite comme suit :

$$N' = \begin{bmatrix} J_{11}J_{22}-J_{12}J_{21} & J_{12}J_{20}-J_{10}J_{22} & J_{10}J_{21}-J_{11}J_{20} \\ J_{02}J_{21}-J_{01}J_{22} & J_{00}J_{22}-J_{02}J_{20} & J_{01}J_{20}-J_{00}J_{21} \\ J_{01}J_{12}-J_{02}J_{11} & J_{02}J_{10}-J_{00}J_{12} & J_{00}J_{11}-J_{01}J_{10} \end{bmatrix} N$$

où les J_{ij} sont les éléments de la matrice J.

1.3.2 Déformation globale

Nous allons présenter quelques déformations globales qui paraissent très utiles pour la modélisation géométrique.

1.3.2.1 Courbure

La **courbure** (bend en anglais) est utilisée pour modéliser les solides classiquement définis par extrusion (sweeping en anglais). L'utilisateur définit une trajectoire ; l'objet se déforme en suivant cette trajectoire.

Etant donnée une trajectoire $P_0(t)$ avec $0 \leq t \leq 1$, on définit un repère local $S(t)$, $T(t)$, $U(t)$ pour chaque point sur cette trajectoire (voir la figure 1.1). Pour ce faire, nous pouvons utiliser le repère de Frénet qui définit $U(t)$ comme la direction de la tangente, $T(t)$ pointant vers le centre de courbure et $S(t)$ étant orthogonal à $T(t)$ et $U(t)$. Ce repère

n'est pas défini aux points de la trajectoire où la courbure est nulle. Cette situation est surtout probable pour une trajectoire plane ; nous prenons alors pour $S(t)$ la normale au plan [BRON 85] .

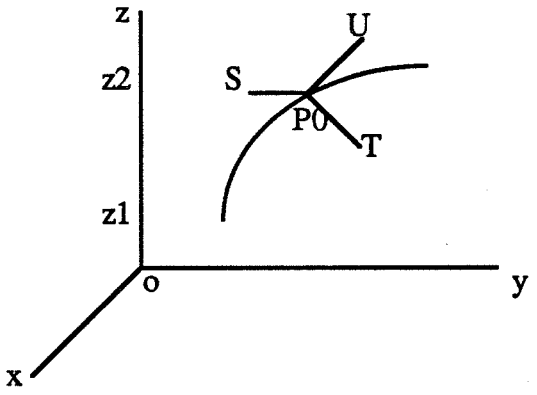


Figure 1.1 Repère local sur une trajectoire

Une courbure le long de l'axe z est définie par les équations suivantes :

$$t = \frac{z - z_1}{z_2 - z_1}$$

$$P' = P_0(t) + x\,S(t) + y\,T(t)$$

où z_1 et z_2 délimitent la région de déformation.

Un cas particulier est la courbure linéaire [BARR 84], où la trajectoire est un arc de cercle (voir la figure 1.2). La transformation est donnée par les équations suivantes :

$$a = k\,(z - z_1)$$

$$x' = x$$

$$y' = \frac{1}{k} - (\frac{1}{k} - y)\cos a$$

$$z' = z_1 + (\frac{1}{k} - y)\sin a$$

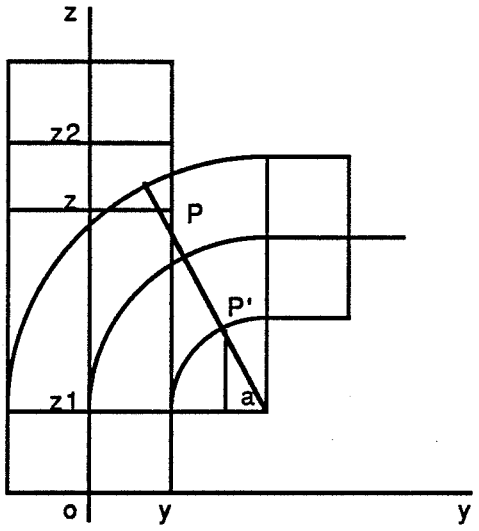


Figure 1.2 Courbure linéaire

1.3.2.2 Effilement

L'**effilement** (taper en anglais) ressemble à l'affinité [BARR 84]. Cette transformation déforme un solide en le diminuant ou en l'agrandissant dans deux directions, sans changement dans la troisième direction. Les équations qui définissent l'effilement le long de l'axe *z* sont les suivantes :

$$r = f(z)$$

$$x' = r\,x$$

$$y' = r\,y$$

$$z' = z$$

où $f(z)$ est donnée par l'utilisateur.

La fonction $f(z)$ peut être, par exemple, une fonction linéaire, ce qui donne l'effilement linéaire :

$$r = 1 + k\,(z - z_1)$$

$$x' = r\,x$$

$$y' = r\,y$$

$$z' = z$$

Une discontinuité C^1 sera produite dans les plans $z = z_1$ et $z = z_2$ (voir la figure 1.3).

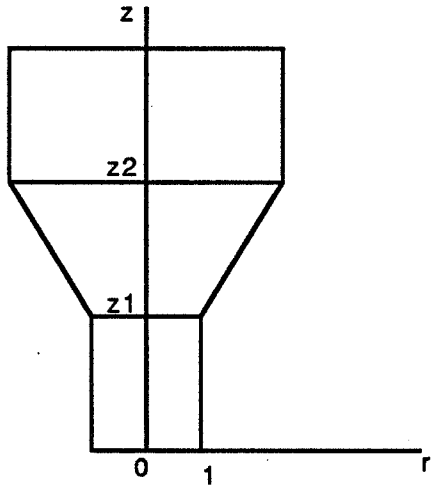


Figure 1.3 Effilement linéaire

1.3.2.3 Torsion

Comme l'effilement est une affinité différentielle, la **torsion** (twist en anglais) est définie comme une rotation différentielle [BARR 84]. Les équations suivantes définissent la torsion le long de l'axe z :

$$\theta = f(z)$$

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

$$z' = z$$

où $f(z)$ est donnée par l'utilisateur.

La fonction $f(z)$ peut aussi être une fonction linéaire, ce qui donne la torsion linéaire :

$$\theta = k\,(z - z_1)$$

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

$$z' = z$$

La courbure, l'effilement et la torsion forment un ensemble complet de déformations permettant de modéliser des solides classiquement modélisés par l'extrusion, qui définit un solide en déplaçant un profil le long d'une trajectoire [POST 86] [COUS 89].

1.3.2.4 Tirage

[ALLA 89] a défini le tirage pour manipuler un réseau de polygones. On choisit d'abord un sommet comme sommet à tirer. Le déplacement de ce sommet est spécifié par l'utilisateur. Pour simuler un tirage réel, les sommets voisins doivent aussi être déplacés. On définit donc une région de déformation qui peut être par exemple une sphère centrée au sommet à tirer. Tous les sommets à l'intérieur de la région de déformation sont déplacés dans la même direction que le sommet à tirer. Par contre, la grandeur du déplacement des sommets voisins peut être modulée par une fonction, selon leur distance au sommet à tirer.

A partir de cette procédure, on peut définir une déformation globale **tirage**. Le tirage dans la direction z peut être défini par les équations suivantes :

$$r = (x^2 + y^2)^{1/2}$$

$$x' = x$$

$$y' = y$$

$$z' = z + d f(r)$$

où d est le déplacement maximum,

$f(r)$ est la fonction de modulation.

La fonction de modulation peut être, par exemple, linéaire, pour produire une déformation de forme conique.

1.3.3 Utilisation des contraintes

Les contraintes sont souvent utilisées en modélisation géométrique. On peut imposer des contraintes sur la position d'un point ou la direction d'une normale. On peut aussi imposer des contraintes sur la relation entre des objets. La résolution des contraintes consiste à déterminer les paramètres des objets et des opérations sur les objets pour satisfaire les contraintes imposées.

Pour ce faire, on définit pour chaque contrainte géométrique une fonction d'énergie $e_i(R_i)$ qui dépend des paramètres à déterminer R_i et dont la valeur est nulle quand la contrainte est satisfaite.

Comme exemple, on applique une courbure linéaire sur un solide pour le déformer. La courbure peut être représentée par la fonction suivante :

$$P' = bend(z_1, z_2, k, P)$$

où z_1, z_2, k définissent la courbure.

Au lieu de définir explicitement les paramètres de la courbure, on peut imposer une contrainte telle que la position après la courbure d'un point P sur l'objet coïncide avec la

position fixe Q. Pour cette contrainte, la fonction d'énergie peut être définie comme la distance entre la position après la courbure du point P et la position Q :

$$e_i(z_1,z_2,k) = |bend(z_1,z_2,k, P) - Q|$$

Pour satisfaire plusieurs contraintes en même temps, on définit une fonction d'énergie globale :

$$e(R) = \sum_{i=0}^{n} e_i(R_i)$$

où R est l'ensemble des paramètres à déterminer.

Pour trouver les paramètres qui satisfont toutes les contraintes géométriques, il faut résoudre l'équation suivante :

$$e(R) = 0$$

Au lieu de la résoudre directement, on cherche les paramètres qui minimisent la fonction d'énergie. On peut utiliser par exemple la méthode du gradient.

1.4 Déformations libres

Les déformations globales donnent beaucoup de nouvelles formes. Comme ce sont des fonctions complexes, leur utilisation est coûteuse au niveau de calculs. De plus, les déformations globales ne peuvent pas déformer des objets d'une façon libre. Pour avoir un nouveau type de déformation, il faut définir la fonction de transformation correspondante.

Pour pouvoir déformer des objets d'une façon libre, [SEDE 86] a introduit la déformation libre. Comme elle est polynomiale, son utilisation est moins coûteuse que les déformations globales.

1.4.1 Formulation

Pour déformer un objet avec la déformation libre, on définit un repère local et on forme ainsi une région de déformation autour de cet objet (voir la figure 1.4). Chaque point P a des coordonnées (s, t, u) dans ce repère local telles que :

$$P = P_0 + sS + tT + uU$$

où P_0, S, T, U définissent le repère local.

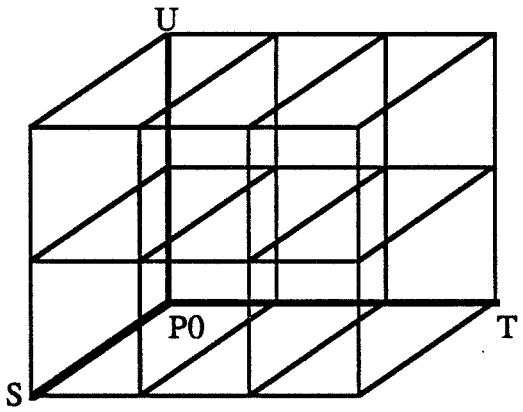


Figure 1.4 Repère local et maillage de points de contrôle

On définit un maillage de points de contrôle P_{ijk} sur la région de déformation :

$$P_{ijk} = P_0 + \frac{i}{l}S + \frac{j}{m}T + \frac{k}{n}U \quad 0 \le i \le l, 0 \le j \le m, 0 \le k \le n$$

où $l+1, m+1$ et $n+1$ donnent le nombre de couches de points de contrôle dans chaque direction.

La déformation est spécifiée par le déplacement des points de contrôle :

$$P'_{ijk} = F(P_{ijk})$$

où P'_{ijk} sont les nouvelles positions des points de contrôle après le déplacement.

Soit P un point sur l'objet à déformer. Pour trouver sa nouvelle position après la déformation, on calcule d'abord ses coordonnées locales :

$$s = \frac{T \wedge U.(P-P_0)}{T \wedge U.S}$$

$$t = \frac{S \wedge U.(P-P_0)}{S \wedge U.T}$$

$$u = \frac{S \wedge T.(P-P_0)}{S \wedge T.U}$$

où $\wedge$ et . représentent respectivement le produit vectoriel et le produit scalaire.

On effectue ensuite une interpolation volumique de type Bézier, en utilisant les coordonnées locales comme des paramètres :

$$P(s,t,u) = \sum_{i=0}^{l} B_{i,l}(s) \sum_{j=0}^{m} B_{j,m}(t) \sum_{k=0}^{n} B_{k,n}(u)\ P'_{ijk}$$

où $B_{i,n}(t)$ sont les fonctions de mélange de Bernstein.

La déformation libre peut transformer, d'une manière libre, un objet de n'importe quelle dimension et de n'importe quelle forme.

Si l'objet à déformer est une surface plane parallèle à deux axes du repère local, le résultat de la déformation libre sera une surface de type Bézier.

1.4.2 Contrôle de la continuité

Dans la formulation de la déformation libre, la région de déformation peut contenir seulement une partie de l'objet à déformer, ce qui permet de définir une **déformation libre locale** (voir la figure 1.5) .

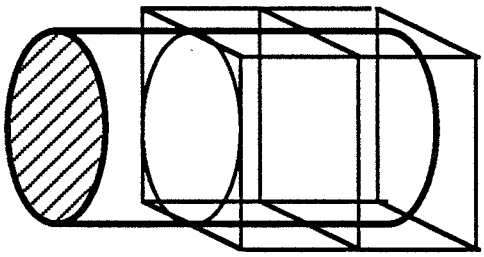


Figure 1.5 Déformation libre locale

Plus généralement, on peut définir plusieurs déformations libres en séquence pour réaliser une déformation plus complexe.

Nous allons considérer le problème de continuité entre deux parties d'un même objet qui subissent respectivement une déformation libre. Le problème est de trouver des contraintes sur le déplacement des points de contrôle de deux déformations libres pour garantir la continuité de l'objet entre les deux déformations libres.

Soit $P(v,w)$ une surface à déformer. On définit deux déformations libres en séquence sur cette surface pour la déformer de deux manières différentes (voir la figure 1.6) . Le deuxième repère local est une translation du premier :

$$P_{02} = P_{01} - \frac{b}{a} S_1$$

$$S_2 = \frac{S_1}{a}$$

$$T_2 = T_1$$

$$U_2 = U_1$$

où a est une constante positive,

b est une constante.

Avec un tel arrangement, les coordonnées locales dans les deux repères locaux d'un même point vérifient les relations suivantes :

$$s_2 = a\, s_1 + b$$

$$t_2 = t_1 = t$$

$$u_2 = u_1 = u$$

Les deux déformations libres s'expriment comme suit :

$$P_1(s_1,t,u) = \sum_{i=0}^{l_1} N_i(s_1) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} N_k(u)\ P'_{1ijk}$$

$$P_2(s_2,t,u) = \sum_{i=0}^{l_2} N_i(s_2) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} N_k(u)\ P'_{2ijk}$$

où $N_i(t)$ est utilisée à la place de $B_{i,n}(t)$ pour faciliter la présentation.

La surface peut être représentée par $s_1(v,w)$, $t(v,w)$, $u(v,w)$ dans le premier repère local, et $s_2(v,w)$, $t(v,w)$, $u(v,w)$ dans le deuxième.

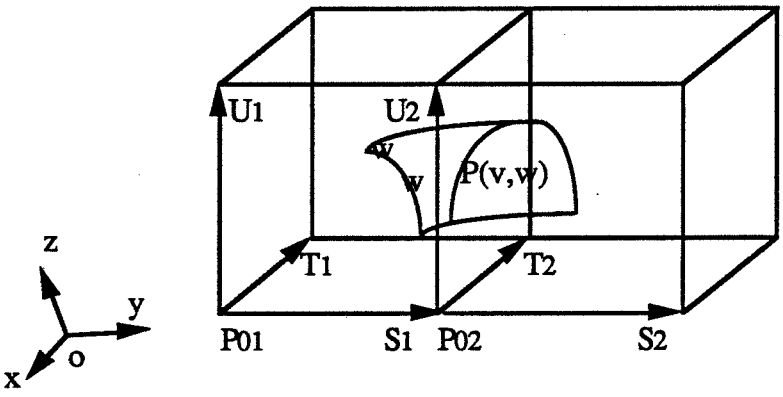


Figure 1.6 Continuité entre deux déformations libres

Après l'évaluation des deux déformations libres, la surface peut être représentée dans le premier repère local comme :

$$P_1(s_1, t, u)$$

$$= P_1(s_1(v,w), t(v,w), u(v,w))$$

$$= P_1(v,w)$$

et dans le deuxième comme :

$$P_2(s_2, t, u)$$

$$= P_2(s_2(v,w), t(v,w), u(v,w))$$

$$= P_2(v,w)$$

La continuité C^0 demande :

$$P_1(v, w) = P_2(v, w)$$

ce qui est équivalent à l'équation suivante :

$$P_1(s_1, t, u) = P_2(s_2, t, u)$$

En utilisant l'interpolation volumique, on obtient :

$$\sum_{i=0}^{l_1} N_i(s_1) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} N_k(u)\ P'_{1ijk}$$

$$= \sum_{i=0}^{l_2} N_i(s_2) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} N_k(u)\ P'_{2ijk}$$

La condition suffisante est donc :

$$\sum_{i=0}^{l_1} N_i(s_1)\ P'_{1ijk} = \sum_{i=0}^{l_2} N_i(s_2)\ P'_{2ijk} \qquad 0 \le j \le m\ ,\ 0 \le k \le n$$

La continuité C^1 demande en plus :

$$\frac{\partial P_1}{\partial v} = \frac{\partial P_2}{\partial v}$$

$$\frac{\partial P_1}{\partial w} = \frac{\partial P_2}{\partial w}$$

ce qui donne :

$$\frac{\partial P_1(s_1,t,u)}{\partial s_1}\frac{\partial s_1}{\partial v}+\frac{\partial P_1(s_1,t,u)}{\partial t}\frac{\partial t}{\partial v}+\frac{\partial P_1(s_1,t,u)}{\partial u}\frac{\partial u}{\partial v}$$

$$=\frac{\partial P_2(s_2,t,u)}{\partial s_2}\frac{\partial s_2}{\partial v}+\frac{\partial P_2(s_2,t,u)}{\partial t}\frac{\partial t}{\partial v}+\frac{\partial P_2(s_2,t,u)}{\partial u}\frac{\partial u}{\partial v}$$

$$\frac{\partial P_1(s_1,t,u)}{\partial s_1}\frac{\partial s_1}{\partial w}+\frac{\partial P_1(s_1,t,u)}{\partial t}\frac{\partial t}{\partial w}+\frac{\partial P_1(s_1,t,u)}{\partial u}\frac{\partial u}{\partial w}$$

$$=\frac{\partial P_2(s_2,t,u)}{\partial s_2}\frac{\partial s_2}{\partial w}+\frac{\partial P_2(s_2,t,u)}{\partial t}\frac{\partial t}{\partial w}+\frac{\partial P_2(s_2,t,u)}{\partial u}\frac{\partial u}{\partial w}$$

Les fonctions $s_1(v,w)$, $s_2(v,w)$, $t(v,w)$ et $u(v,w)$ sont indépendantes des déformations. En tenant compte des relations entre les deux repères locaux, on obtient les équations suivantes :

$$a\frac{\partial s_1}{\partial v}=\frac{\partial s_2}{\partial v}$$

$$a\frac{\partial s_1}{\partial w}=\frac{\partial s_2}{\partial w}$$

La condition suffisante est donc :

$$\frac{\partial P_1(s_1,t,u)}{\partial s_1}=a\frac{\partial P_2(s_2,t,u)}{\partial s_2}$$

$$\frac{\partial P_1(s_1,t,u)}{\partial t}=\frac{\partial P_2(s_2,t,u)}{\partial t}$$

$$\frac{\partial P_1(s_1,t,u)}{\partial u}=\frac{\partial P_2(s_2,t,u)}{\partial u}$$

En utilisant l'interpolation volumique, on a :

$$\sum_{i=0}^{l_1}\frac{\partial N_i(s_1)}{\partial s_1}\sum_{j=0}^{m}N_j(t)\sum_{k=0}^{n}N_k(u)\;P'_{1ijk}$$

$$=a\sum_{i=0}^{l_2}\frac{\partial N_i(s_2)}{\partial s_2}\sum_{j=0}^{m}N_j(t)\sum_{k=0}^{n}N_k(u)\;P'_{2ijk}$$

$$\sum_{i=0}^{l_1} N_i(s_1) \sum_{j=0}^{m} \frac{\partial N_j(t)}{\partial t} \sum_{k=0}^{n} N_k(u)\ P'_{1ijk}$$

$$= \sum_{i=0}^{l_2} N_i(s_2) \sum_{j=0}^{m} \frac{\partial N_j(t)}{\partial t} \sum_{k=0}^{n} N_k(u)\ P'_{2ijk}$$

$$\sum_{i=0}^{l_1} N_i(s_1) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} \frac{\partial N_k(u)}{\partial u} P'_{1ijk}$$

$$= \sum_{i=0}^{l_2} N_i(s_2) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} \frac{\partial N_k(u)}{\partial u} P'_{2ijk}$$

On obtient donc la condition suivante :

$$\sum_{i=0}^{l_1} \frac{\partial N_i(s_1)}{\partial s_1} P'_{1ijk} = a \sum_{i=0}^{l_2} \frac{\partial N_i(s_2)}{\partial s_2} P'_{2ijk} \quad 0 \le j \le m\ ,\ 0 \le k \le n$$

D'une façon récursive, on peut définir la continuité d'ordre supérieur.

On voit que ces conditions pour chaque j et chaque k sont les mêmes que celles pour la continuité de deux courbes.

Dans le cas de la déformation libre locale, pour avoir la continuité C^0 (respectivement C^1 et C^2) entre la partie déformée et la partie non-déformée, il faut que la première (respectivement les deux premières et les trois premières) couche de points de contrôle reste invariante.

1.4.3 Transformation des normales

La déformation libre est décomposée en deux étapes successives : un changement de repère et une interpolation volumique. Les fonctions de transformation correspondant à ces deux étapes sont toutes connues explicitement. On peut les utiliser pour calculer les matrices Jacobiennes correspondantes.

Les fonctions de mélange de type Bézier sont les suivantes :

$$B_{i,n}(t) = C_n^i (1-t)^{n-i} t^i$$

avec la convention $0^0 = 1$.

La dérivation est simple :

$$\frac{dB_{i,n}(t)}{dt} \qquad pour\ 0 < i < n$$

$$= C_n^i\,(i(1-t)^{n-i}\,t^{i-1} - (n-i)(1-t)^{n-i-1}\,t^i)$$

$$= C_n^i\,(i-nu)(1-t)^{n-i-1}\,t^{i-1}$$

$$\frac{dB_{0,n}(t)}{dt} = -n(1-t)^{n-1}$$

$$\frac{dB_{n,n}(t)}{dt} = nt^{n-1}$$

Pour l'interpolation volumique, la dérivation est obtenue comme suit :

$$P(s,t,u) = \sum_{i=0}^{l} B_{i,l}(s) \sum_{j=0}^{m} B_{j,m}(t) \sum_{k=0}^{n} B_{k,n}(u)\ P'_{ijk}$$

$$\frac{\partial P(s,t,u)}{\partial s} = \sum_{i=0}^{l} \frac{dB_{i,l}(s)}{ds} \sum_{j=0}^{m} B_{j,m}(t) \sum_{k=0}^{n} B_{k,n}(u)\ P'_{ijk}$$

1.5 Visualisation des solides déformés

La frontière d'un solide peut être décomposée en surfaces discontinues entre elles. Chaque surface peut en général être représentée comme une surface paramétrique :

$$P = P(u, v) \quad , \qquad 0 \le u,v \le 1$$

La déformation d'une surface paramétrique reste une surface paramétrique. La visualisation des solides déformés est donc similaire à la visualisation des surfaces paramétriques.

La visualisation la plus simple est la visualisation en fil de fer. On peut visualiser directement des courbes avec cette technique. En général, on préfère approximer toutes les courbes par des arêtes. Avec l'algorithme de Bresenham [BRES 65], la visualisation d'une arête est effectuée avec des calculs entiers et incrémentaux. De plus, cet algorithme est souvent supporté en matériel. La visualisation en fil de fer est donc rapide. Par contre, elle est souvent ambiguë : une représentation en fil de fer peut être interprétée comme des solides différentes [PERO 88]. Il convient donc d'utiliser cette visualisation pour la modélisation interactive.

Pour visualiser des surfaces paramétriques avec élimination des parties cachées, on peut utiliser le tampon de profondeur, le balayage ligne par ligne ou le tracé de rayon.

On peut visualiser une surface paramétrique en tant que telle avec le tampon de profondeur [LIEN 87] ou le balayage ligne par ligne [LANE 80]. En général, il est plus facile de facettiser la surface paramétrique pour l'approximer par une surface polygonale et de visualiser ensuite la surface polygonale avec ces algorithmes de visualisation.

La facettisation d'un solide revient à facettiser toutes les surfaces qui le composent, en gardant la cohérence entre elles.

1.5.1 Facettisation des surfaces paramétriques

On peut choisir la méthode de facettisation d'une surface selon trois critères.

Le premier critère est l'optimisation. Si l'optimisation n'est pas demandée, on utilisera la facettisation uniforme. Si l'optimisation est demandée, on utilisera la facettisation adaptative.

Le deuxième critère est la visualisation. Si on visualise en fil de fer, on préfère la facettisation non-triangulaire. Si on visualise en balayage ligne par ligne, il faut faire la facettisation triangulaire afin que les facettes soient planes.

Le troisième critère est la déformation. Sans déformation, le niveau de facettisation peut être fixé par avance. Avec la déformation, le niveau de facettisation dépendra de la déformation.

La facettisation uniforme consiste à subdiviser d'une façon uniforme le domaine de paramètres pour définir des points sur la surface :

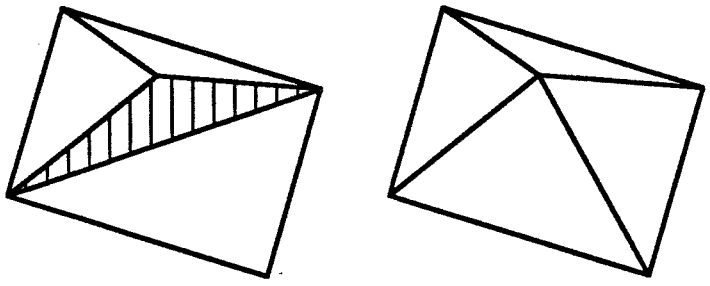
$$P_{ij} = P(\frac{i}{m}, \frac{j}{n}) \qquad 0 \le i \le m, \quad 0 \le j \le n$$

Chaque morceau de surface est ensuite approximé par un polygone.

En général, la surface est plus courbée à certains endroits qu'ailleurs. Pour utiliser moins de polygones, tout en garantissant la précision d'approximation, il faut faire une facettisation adaptative. Le niveau de facettisation doit pouvoir s'adapter automatiquement à la courbure de la surface. Pour ce faire, on effectue d'abord une facettisation initiale, qui est uniforme avec un niveau de facettisation faible. Ensuite, on teste chaque facette suivant les critères de facettisation. Si les critères ne sont pas satisfaits, on subdivise la facette et on traite récursivement les facettes produites.

1.5.1.1 Contraintes de facettisation

La facettisation adaptative doit respecter la contrainte de cohérence [CATM 74] : si on subdivise un triangle par division d'une de ses arêtes, sans subdiviser le triangle voisin qui partage l'arête divisée, on risque de créer un trou sur la surface (voir la figure 1.7(a)) . Pour garantir la cohérence de la surface, la subdivision d'un triangle doit entraîner systématiquement la subdivision du triangle voisin qui partage l'arête divisée du triangle courant (voir la figure 1.7(b)). Notons que [MATA 89] a proposé une autre solution qui consiste à ramener le point en cause sur l'arête d'adjacence du triangle voisin.



(a) (b)

Figure 1.7 Contrainte de cohérence

Les facettes produites par la facettisation adaptative doivent être planes. Pour satisfaire cette contrainte, la facettisation adaptative ne produit que des triangles.

1.5.1.2 Mécanismes de facettisation adaptative

On fait une facettisation uniforme comme facettisation initiale. On teste chaque facette suivant les critères de subdivision. Si les critères ne sont pas satisfaits, on subdivise la facette et on traite récursivement les facettes produites.

Pour satisfaire la contrainte de planéité, on peut trianguler avant la facettisation adaptative toutes les facettes issues de la facettisation initiale. La facettisation adaptative se fait à l'aide de la subdivision de triangles.

Il y a différentes façons de subdiviser un triangle. Pour favoriser la production des triangles équilatéraux, [PARR 86] désigne pour chaque triangle une des trois arêtes comme l'arête "la plus longue". La subdivision d'un triangle se fait par division de son arête "la plus longue". Après la subdivision, les deux autres arêtes du triangle deviennent systématiquement l'arête "la plus longue" des deux triangles produits. Avec cette méthode de subdivision, les trois arêtes d'un triangle sont divisées successivement. La taille du triangle diminue donc dans toutes les directions.

Pour satisfaire la contrainte de cohérence, après la subdivision d'un triangle, on doit subdiviser aussi son triangle voisin du côté de l'arête "la plus longue". Mais ce triangle voisin ne partage pas forcément la même arête "la plus longue" avec le triangle courant. Dans le cas négatif, il faut examiner le troisième triangle qui est le voisin du deuxième triangle du côté de l'arête "la plus longue". On continue de cette manière jusqu'à trouver deux triangles qui partagent la même arête "la plus longue". On subdivise alors ces deux triangles ainsi que les triangles parcourus. Cette procédure récursive peut être représentée par le programme suivant :

```
recsub(I)
{
        J = le voisin de I du côté de l'arête "la plus longue";
        si I et J ne partagent pas la même arête "la plus longue"
                recsub(J);
        subdivide(I, J);
}
```

Cette procédure récursive est illustrée sur la figure 1.8 , où un petit cercle est utilisé pour désigner l'arête "la plus longue". Pour subdiviser le triangle *A*, il faut trouver les triangles *D* et *E* qui partagent l'arête "la plus longue".

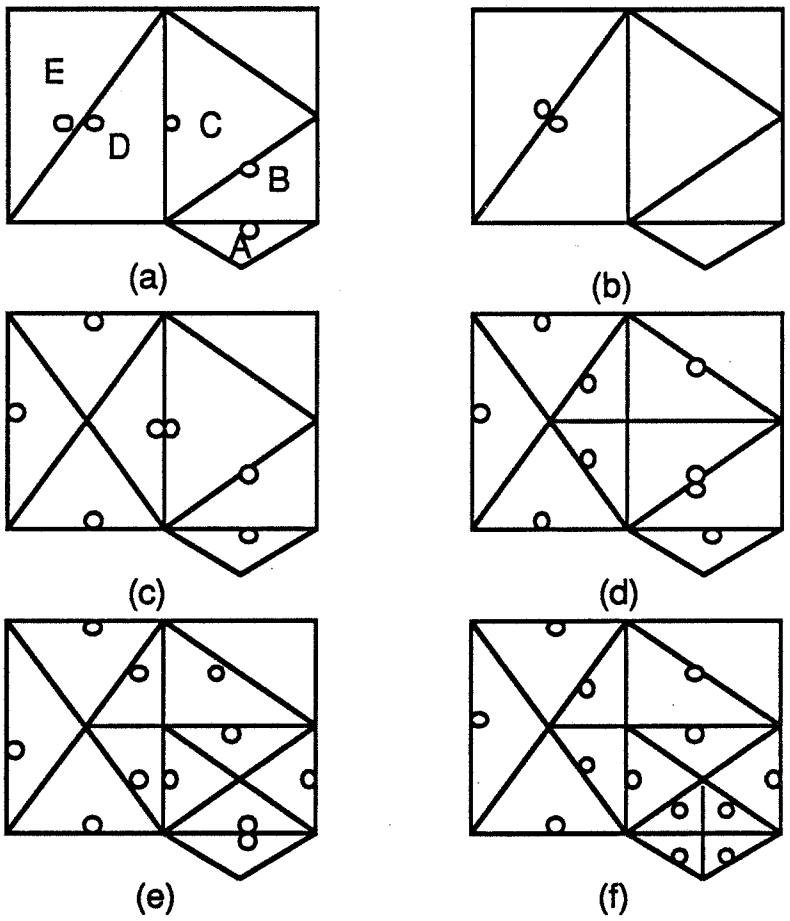
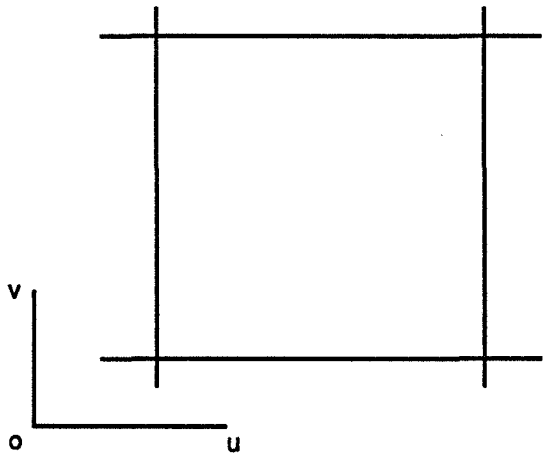


Figure 1.8 Subdivision de l'arête "la plus longue"

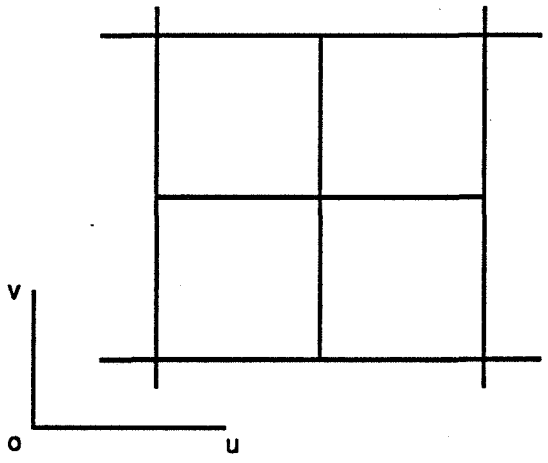
Cette procédure récursive se termine toujours, à condition que deux triangles voisins partagent l'arête "la plus longue" avant la facettisation adaptative.

Pour supporter cette procédure de facettisation, [PARR 86] utilise une pile. On effectue une facettisation uniforme en triangles comme facettisation initiale. Tous les triangles produits sont mis dans la pile. A chaque étape, on dépile un triangle et on le teste suivant les critères de facettisation. Si les critères sont satisfaits, on continue en dépilant un autre triangle. Sinon, on subdivise le triangle et les triangles voisins et on empile tous les triangles ainsi produits. La procédure se termine quand la pile est vide.

La facettisation adaptative peut aussi se faire à l'aide de la subdivision de facettes non-triangulaires. En fait, le domaine de paramètres d'une surface paramétrique est un rectangle. La subdivision de rectangles dans le domaine de paramètres correspond à la subdivision de quadrilatères dans l'espace. On teste chaque quadrilatère suivant les critères de facettisation. Si les critères ne sont pas satisfaits, on subdivise le quadrilatère dans les deux directions (voir la figure 1.9) .



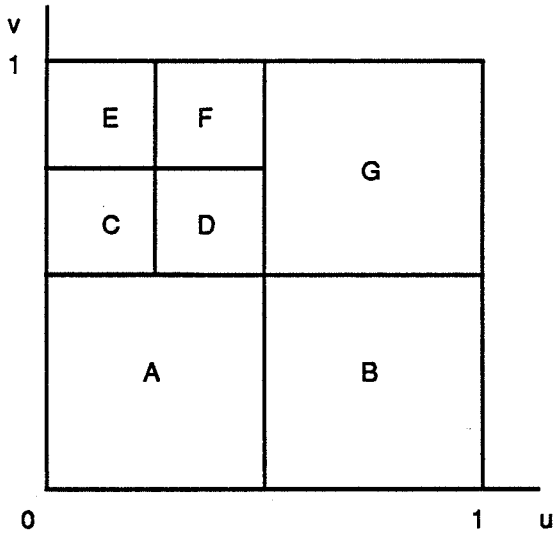
(a)



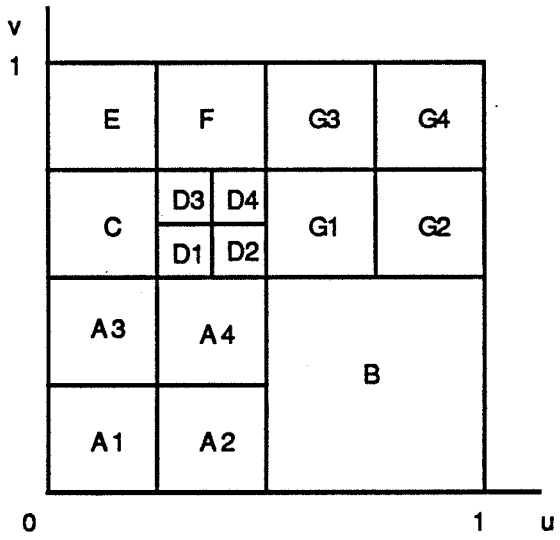
(b)

Figure 1.9 Subdivision de quadrilatères

Pour satisfaire la contrainte de cohérence, on impose une condition à la procédure de facettisation : la différence entre les niveaux de facettisation de deux quadrilatères voisins ne doit pas dépasser un. Sur la figure 1.10, la subdivision de D entraîne systématiquement la subdivision de A et de G .



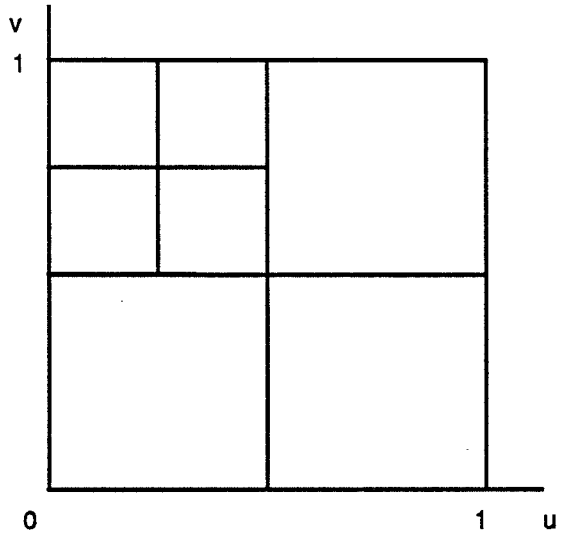
(a)



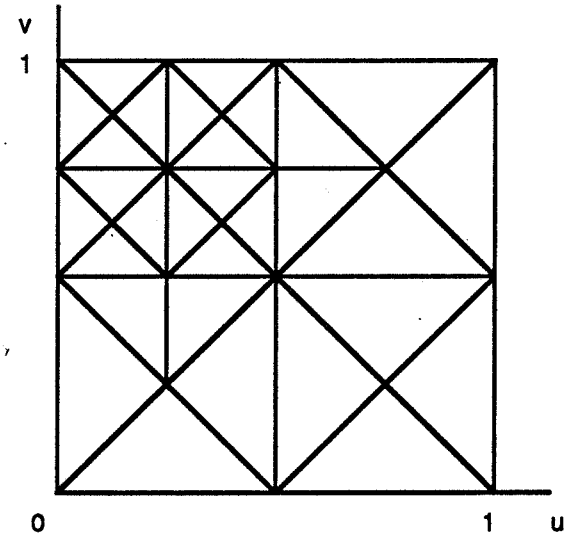
(b)

Figure 1.10 Subdivision de quadrilatères voisins

A la fin de la procédure de facettisation, il faut trianguler tous les quadrilatères pour satisfaire la contrainte de cohérence (voir la figure 1.11) . Chaque quadrilatère est divisé en quatre triangles. Si le voisin d'un triangle est plus grand, on divise ce triangle en deux.



(a)



(b)

Figure 1.11 Triangulation de quadrilatères

Pour représenter ce mécanisme de facettisation, [HERZ 87] utilise l'**arbre quaternaire.**

1.5.1.3 Critères de facettisation

On utilise la courbure de la surface comme critère de facettisation. Plus la courbure est grande, plus on doit facettiser la surface.

Au lieu de calculer directement la courbure d'une facette, on peut utiliser les normales au sommets [HERZ 87] . L'angle entre deux normales unitaires N_1 et N_2 est inférieur à *da* si la condition suivante est vérifiée :

$$N_1 . N_2 > cos\ da$$

L'angle *da* est déterminé empiriquement pour que l'image produite soit satisfaisante.

On teste toutes les paires de normales aux sommets voisins de la facette. Si une paire de normales ne satisfait pas le test, on subdivise la facette.

Si on utilise la subdivision de triangles et si on ne connait pas les normales aux sommets, on peut utiliser les normales aux triangles [PARR 86] . On compare la normale du triangle courant avec les normales des trois triangles voisins.

En utilisant les normales pour tester le critère de courbure, il est possible qu'une facette soit très courbée tandis que les normales aux sommets ont à peu près la même direction. Pour bien démarrer la procédure de facettisation, on subdivise une facette sans considérer le critère de courbure, si sa taille est trop grande.

Le critère de courbure n'est pas suffisant. Il peut arriver qu'une partie de la surface soit tellement courbée que la facettisation produise des facettes dont la taille est inférieure à un pixel après projection sur l'écran, ce qui est inutile. Pour traiter cette situation, on peut utiliser un autre critère, le critère de taille. Si la taille d'une facette est inférieure à un pixel après projection sur l'écran, on arrête la subdivision de cette facette.

Il est bien connu que l'homme est très sensible à l'aliassage sur la frontière et sur la silhouette des surfaces. Pour l'anti-aliassage, il faut facettiser d'avantage à ces endroits. Pour ce faire, on peut ajouter dans la facettisation adaptative le critère d'intersection et le critère de silhouette. Si une facette se trouve sur ces endroits critiques, on la facettise jusqu'à ce que le critère de taille soit satisfait [CARL 82] [HERZ 87] .

1.5.2 Visualisation en tracé de rayons des surfaces paramétriques

Le tracé de rayons est la technique de choix lorsque le but poursuivi dans la génération d'images est le réalisme. Le tracé de rayons visualise des objets en simulant la propagation de rayons lumineux dans le monde. Il peut simuler la plupart des phénomènes optiques comme les ombres portées, les réflexions et les réfractions [GLAS 89]. Il est bien connu que l'étape cruciale au niveau du temps de calcul par le tracé de rayons est le calcul de l'intersection entre les rayons et les objets.

Pour calculer l'intersection d'un rayon avec une surface paramétrique, on peut facettiser la surface paramétrique [SNYD 87]. En général, on préfère calculer directement l'intersection des rayons avec la surface paramétrique. Les algorithmes proposés sont souvent divisés en trois classes :

1) Découpage récursif de la surface [BARR 86]. La propriété de l'enveloppe convexe des surfaces de type Bézier ou B-spline est souvent utilisée pour faciliter le test d'intersection. Ces algorithmes peuvent être utilisés seuls, ou comme prétraitement pour trouver une solution approximative qui sera améliorée par d'autres types d'algorithmes.

2) Ramener le problème à la résolution numérique d'une équation algébrique univariée [KAJI 82].

3) Résoudre directement un système d'équations algébriques [BIAR 90].

La surface d'un objet peut être représentée sous forme implicite :

$$f(P) = 0$$

ou sous forme paramétrique :

$$P = P(u, v)$$

Le rayon peut aussi être représenté sous forme paramétrique :

$$P = A + Bt$$

ou sous forme implicite comme intersection de deux plans :

$$f(P) = 0$$

$$g(P) = 0$$

Pour faciliter le calcul de l'intersection, on préfère utiliser des représentations différentes pour le rayon et la surface. Pour calculer l'intersection entre un rayon et une surface paramétrique, on choisira donc l'intersection de deux plans pour représenter le rayon. Les points d'intersection sont obtenus en substituant l'équation de la surface paramétrique dans les équations des deux plans, ce qui donne un système de deux équations à deux variables :

$$f(P(u, v)) = 0$$

$$g(P(u, v)) = 0$$

Si la surface paramétrique est une surface bi-cubique, les deux équations ainsi obtenues seront des équations de degré six.

Si la surface est sous forme implicite, on utilise la forme paramétrique du rayon. Les points d'intersection sont obtenus en substituant l'équation du rayon dans l'équation de la surface, ce qui donne une équation univariée :

$$f(A + t\,B) = 0$$

[KAJI 82] utilise la géométrie algébrique pour résoudre le système de deux équations à deux variables :

$$f(P(u, v)) = 0$$

$$g(P(u, v)) = 0$$

On utilise la technique des résultants pour transformer ce système d'équations en une équation algébrique.

Le résultant de deux polynômes :

$$a(u) = \sum_{i=0}^{m} a^i \, u^{m-i}$$

$$b(u) = \sum_{j=0}^{n} b^j \, u^{n-j}$$

dont les racines sont respectivement $\alpha_1, ..., \alpha_m$ et $\beta_1, ..., \beta_n$, est défini comme suit :

$$R(a, b) = a_0^n \, b_0^m \, (-1)^{m+n-1} \prod_{i=1,m \; j=1,n} (\alpha_i - \beta_j)$$

Selon la théorie de la géométrie algébrique, le résultant de deux polynômes peut être représenté comme un polynôme des coefficients de ces deux polynômes. La forme déterminantale de Bezout est une méthode efficace pour calculer le résultant. Pour deux polynômes cubiques par exemple, le résultant est :

$$R(a, b) = \begin{vmatrix} \begin{vmatrix} b_0 & b_1 \\ a_0 & a_1 \end{vmatrix} & & \begin{vmatrix} b_0 & b_2 \\ a_0 & a_2 \end{vmatrix} \begin{vmatrix} b_0 & b_3 \\ a_0 & a_3 \end{vmatrix} \\ \begin{vmatrix} b_0 & b_2 \\ a_0 & a_2 \end{vmatrix} \begin{vmatrix} b_0 & b_3 \\ a_0 & a_3 \end{vmatrix} & + & \begin{vmatrix} b_1 & b_2 \\ a_1 & a_2 \end{vmatrix} \begin{vmatrix} b_1 & b_3 \\ a_1 & a_3 \end{vmatrix} \\ \begin{vmatrix} b_0 & b_3 \\ a_0 & a_3 \end{vmatrix} & & \begin{vmatrix} b_1 & b_3 \\ a_1 & a_3 \end{vmatrix} \begin{vmatrix} b_2 & b_3 \\ a_2 & a_3 \end{vmatrix} \end{vmatrix}$$

Comme on l'a rappelé, le calcul de l'intersection d'un rayon avec une surface paramétrique revient à la résolution d'un système de deux équations à deux variables :

$$f(u, v) = 0$$
$$g(u, v) = 0$$

Si on considère $f(u, v)$ et $g(u, v)$ comme des polynômes en v, dont les coefficients sont des polynômes en u, on peut calculer leur résultant $r(u)$, qui est un polynôme en u. Pour que les deux équations partagent la même solution, il faut résoudre l'équation suivante :

$$r(u) = 0$$

Avec la technique du résultant, on a transformé le système de deux équations en une équation polynomiale. Pour la résoudre, on peut utiliser la méthode itérative de Newton :

$$u_{k+1} = u_k - r(u_k) \frac{1}{\frac{dr(u_k)}{du}}$$

2 Déformation libre de type B-spline

Comme la déformation libre définie par Sederberg utilise l'interpolation de type Bézier, l'ordre d'interpolation dépend du nombre des points de contrôle utilisés. Pour réaliser une déformation complexe avec la déformation libre, il faut définir beaucoup de points de contrôle et effectuer une interpolation d'ordre important. Nous proposons d'utiliser l'interpolation de type B-spline dans la formulation de la déformation libre. Avec cette nouvelle formulation, l'ordre d'interpolation devient indépendant du nombre de points de contrôle.

Dans le premier paragraphe, nous expliquerons quel est l'intéret d'utiliser l'interpolation de type B-spline dans la formulation de la déformation libre. Dans le deuxième paragraphe, nous présenterons, d'une façon générale, la formulation de la déformation libre avec l'interpolation de type B-spline. Dans les deux paragraphes suivants, nous détaillerons la formulation de la déformation libre avec l'interpolation de type B-spline uniforme puis avec l'interpolation de type B-spline non-uniforme. Nous présenterons dans le cinquième paragraphe la formulation de la déformation libre qui combine l'interpolation de type Bézier et celle de type B-spline. Nous conclurons dans le sixième paragraphe.

2.1 Justification de la déformation libre de type B-spline

La formulation de [SEDE 86] est basée sur une interpolation volumique de type Bézier. L'ordre d'interpolation est lié directement au nombre de points de contrôle. Pour modéliser une forme complexe, il faut définir beaucoup de points de contrôle et le calcul deviendra très lourd. Pour ne pas augmenter l'ordre d'interpolation, [SEDE 86] utilise plusieurs petits maillages de points de contrôle en séquence pour définir une déformation complexe. La continuité de l'objet entre les déformations libres est assurée en imposant des contraintes sur les positions relatives de ces petits maillages. Le même principe est utilisé dans [COQU 90] [COQU 91] où les déformations libres utilisant l'interpolation cubique de type Bézier sont assemblées pour réaliser une déformation complexe.

Examinons maintenant en détail le contrôle de la continuité entre les interpolations de type Bézier. Nous utilisons plusieurs interpolations cubiques en séquence pour définir la courbe sur la figure 2.1. Le premier segment $P_0(t)$ est défini par les points de contrôle $P_{00}, P_{01}, P_{02}, P_{03}$, le deuxième $P_1(t)$ par $P_{10}, P_{11}, P_{12}, P_{13}$, et ainsi de suite.

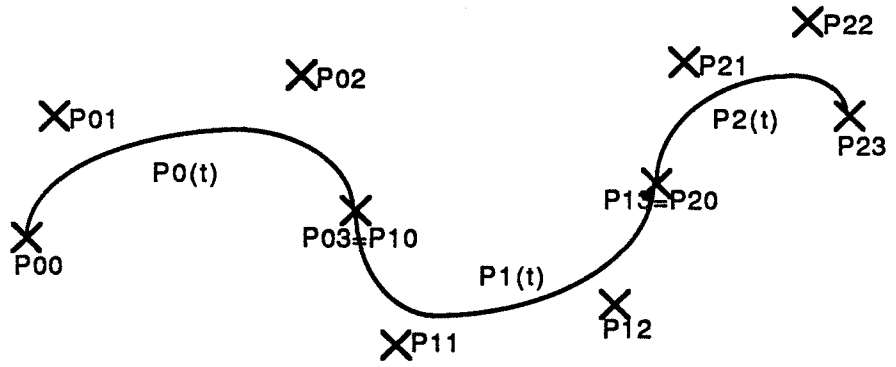


Figure 2.1 Interpolations de type Bézier en séquence

Pour obtenir la continuité C^0 entre $P_0(t)$ et $P_1(t)$, P_{10} doit coïncider avec P_{03} .

Pour avoir la continuité C^1 en P_{03}, P_{11} doit aussi être contraint. Par conséquent, quand on ajoute une nouvelle interpolation, on obtient seulement deux points de contrôle (P_{12} ,P_{13} pour $P_1(t)$, P_{22} ,P_{23} pour $P_2(t)$) qui sont libres d'être déplacés par l'utilisateur pour définir la courbe. Les deux autres sont utilisés pour garantir la continuité.

Quand on déplace les points de contrôle P_{02} et P_{03}, seuls deux segments $P_0(t)$ et $P_1(t)$ seront modifiés. En fait, il suffit de déplacer P_{11} pour maintenir la continuité C^1 en P_{03}. La continuité en P_{13} n'est pas violée. On a alors le contrôle local de la courbe.

Pour avoir la continuité C^2 en P_{03}, P_{12} doit aussi être contraint. Par conséquent, quand on ajoute une nouvelle interpolation, on obtient un seul point de contrôle (P_{13} pour $P_1(t)$ et P_{23} pour $P_2(t)$) qui est libre d'être déplacé par l'utilisateur pour définir la courbe ; les trois autres sont utilisés pour garantir la continuité.

Quand on déplace le point de contrôle P_{03}, on doit déplacer P_{12}, en plus de P_{11}, pour maintenir la continuité C^2 en P_{03}. Malheureusement, P_{12} participe au maintien de la continuité en P_{13}. Avec une telle propagation de déplacements, le déplacement d'un point de contrôle modifiera toute la courbe. On perd donc le contrôle local. Pour avoir le contrôle local, il faut utiliser l'interpolation quantique [BART 86].

Nous proposons ici une autre solution. Nous remplaçons, dans la formulation des déformations libres, l'interpolation de type Bézier par une interpolation de type B-spline ; en effet, l'ordre d'une interpolation de type B-spline ne dépend pas du nombre de points de contrôle.

[GRIE 89] a aussi utilisé l'interpolation de type B-spline pour formuler la déformation libre. Avec cette formulation, l'ordre d'interpolation devient indépendant du nombre de points de contrôle. En revanche, cette formulation peut entraîner des effets inhabituels : même si les points de contrôle ne sont pas déplacés, l'objet sera déjà déformé.

En tenant compte de la condition d'invariance que nous allons présenter après, nous allons obtenir une formulation différente.

2.2 Déformation libre de type B-spline

Nous allons présenter la formulation de la déformation libre avec l'interpolation de type B-spline.

2.2.1 Condition d'invariance de la déformation libre

Avant de procéder à la formulation de la déformation libre avec l'interpolation de type B-spline, nous allons définir une condition que doit respecter toute nouvelle formulation de la déformation libre.

Avec la déformation libre, la déformation de l'objet est spécifiée par le déplacement des points de contrôle. Il est donc naturel d'imposer une contrainte telle que l'objet ne soit pas déformé tant que les points de contrôle restent invariants. Nous appellerons **condition d'invariance** cette contrainte.

Examinons maintenant la condition que doivent satisfaire les fonctions de mélange pour satisfaire la condition d'invariance.

Pour simplifier la présentation, nous considèrerons la déformation libre en une seule dimension. Un point P dans la région de déformation a une coordonnée t dans le repère local telle que :

$$P = P_0 + tT$$

Les points de contrôle sont définis par :

$$P_i = P_0 + \frac{i}{n} T \, , \, 0 \leq i \leq n$$

En utilisant dans l'interpolation les points de contrôle non-déplacés P_i, nous obtenons :

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

$$= \sum_{i=0}^{n} N_i(t) \, (P_0 + \frac{i}{n} T)$$

$$= P_0 + T \sum_{i=0}^{n} N_i(t) \, \frac{i}{n}$$

Pour respecter la condition d'invariance :

$$P(t) = P$$

les fonctions de mélange doivent satisfaire l'équation suivante :

$$\sum_{i=0}^{n} i N_i(t) = n \, t$$

Remarque : La condition d'invariance est bien respectée par la formulation avec l'interpolation de type Bézier, car les fonctions de mélange de type Bézier vérifient l'équation suivante :

$$\sum_{i=0}^{n} B_{i,n}(t) \, \frac{i}{n} = t$$

En voici la démonstration :

$$\sum_{i=0}^{n} B_{i,n}(t)\,\frac{i}{n}$$

$$= \sum_{i=0}^{n} \frac{i}{n} C_n^i\,(1-t)^{n-i}\,t^i$$

$$= \sum_{i=1}^{n} C_{n-1}^{i-1}\,(1-t)^{n-i}\,t^i$$

$$= t \sum_{i=0}^{n-1} C_{n-1}^{i}\,(1-t)^{n-1-i}\,t^i$$

$$= t \qquad \Diamond$$

2.2.2 Formulation

La formulation des déformations libres avec l'interpolation de type B-spline ressemble à la formulation avec l'interpolation de type Bézier.

Pour déformer un objet, on définit un repère local pour former une région de déformation autour de cet objet. Chaque point P dans la région de déformation a des coordonnées (s', t', u') dans ce repère local telles que :

$$P = P_0 + s'S + t'T + u'U$$

où P_0, S, T, U définissent le repère local.

On définit un maillage de points de contrôle P_{ijk} sur la région de déformation :

$$P_{ijk} = P_0 + iS + jT + kU \qquad 0 \le i \le l\,,\ 0 \le j \le m\,,\ 0 \le k \le n$$

où $l+1, m+1, n+1$ correspondent au nombre de couches de points de contrôle dans chaque direction du repère local.

Notons que cette formule est légèrement différente de celle donnée dans la formulation de type Bézier, pour respecter la convention de paramétrage de l'interpolation de type B-spline.

La déformation est spécifiée par le déplacement des points de contrôle :

$$P'_{ijk} = F(P_{ijk})$$

où P'_{ijk} sont les nouvelles positions des points de contrôle après le déplacement.

Etant donné un point P sur l'objet à déformer, nous calculons d'abord ses coordonnées locales :

$$s' = \frac{T \wedge U.(P-P_0)}{T \wedge U.S}$$

$$t' = \frac{S \wedge U.(P-P_0)}{S \wedge U.T}$$

$$u' = \frac{S \wedge T.(P-P_0)}{S \wedge T.U}$$

où $\wedge$ et . représentent respectivement le produit vectoriel et le produit scalaire.

D'une façon générale, pour respecter la condition d'invariance, nous allons effectuer un reparamétrage avant d'effectuer l'interpolation :

$$s = f_s(s')$$

$$t = f_t(t')$$

$$u = f_u(u')$$

où $f_s(s), f_t(t), f_u(u)$ sont les fonctions de reparamétrage à définir.

En utilisant les nouveaux paramètres, nous définissons la nouvelle position du point P après déformation par une interpolation volumique de type B-spline :

$$P(s,t,u) = \sum_{i=0}^{l} N_{i,ks}(s) \sum_{j=0}^{m} N_{j,kt}(t) \sum_{k=0}^{n} N_{k,ku}(u)\ P'_{ijk}$$

où P'_{ijk} sont les nouvelles positions des points de contrôle après le déplacement,

$N_{i,k}(t)$ sont les fonctions de mélange de type B-spline.

2.2.3 Reparamétrage

Il reste à trouver la fonction de reparamétrage $f(t)$ telle que la formulation des déformations libres de type B-spline respecte la condition d'invariance.

Pour simplifier la présentation, nous regardons la déformation libre en une seule dimension. Nous avons les points de contrôle :

$$P_i = P_0 + iT \quad 0 \le i \le n$$

et un point sur l'objet à déformer :

$$P = P_0 + tT$$

En utilisant les points de contrôle non-déplacés dans l'interpolation, nous obtenons la nouvelle position du point P :

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

La condition d'invariance impose :

$$P(t) = P$$

ce qui donne :

$$\sum_{i=0}^{n} N_{i,k}(t)\ i = t'$$

Nous obtenons une équation algébrique qui définit implicitement la fonction de reparamétrage. Pour effectuer le reparamétrage, il faut résoudre cette équation. L'équation obtenue de cette façon donne toujours une et une seule racine réelle, car l'interpolation de type B-spline vérifie la propriété suivante :

Soient $n+1$ points ordonnés sur l'axe des x :

$$x_0 \leq x_1 \leq \ldots \leq x_n$$

Leur interpolation de type B-spline définit la fonction suivante :

$$x(t) = \sum_{i=0}^{n} N_{i,k}(t)\ x_i$$

Cette fonction est croissante :

$$\frac{dx(t)}{dt} \geq 0$$

En particulier, si les points ne sont pas tous les mêmes, la fonction sera strictement croissante.

Cette propriété paraît évidente. Nous en donnons ici une démonstration par récurrence, qui est en revanche relativement complexe.

Preuve :

Pour $k = 1$, on a :

$$N_{i,1}(t) = \begin{cases} 1 & si\ \ t_i \leq t < t_{i+1} \\ 0 & sinon \end{cases}$$

La démonstration est immédiate.

Supposons que la propriété soit vérifiée pour l'interpolation d'ordre $k-1$. Regardons maintenant l'interpolation d'ordre k . Dans l'intervalle $[t_i, t_{i+1})$, ce sont les fonctions de mélange $N_{i,k}(t)$, $N_{i-1,k}(t)$, ..., $N_{i-k+1,k}(t)$ qui participent à l'interpolation :

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

$$x(t) = x_i N_{i,k}(t) + x_{i-1} N_{i-1,k}(t) + \ldots + x_{i-k+1} N_{i-k+1,k}(t)$$

$$= x_i \frac{(t_{i+k}-t) N_{i+1,k-1}(t)}{t_{i+k}-t_{i+1}} + x_i \frac{(t-t_i) N_{i,k-1}(t)}{t_{i+k-1}-t_i}$$

$$+ x_{i-1} \frac{(t_{i+k-1}-t) N_{i,k-1}(t)}{t_{i+k-1}-t_i} + x_{i-1} \frac{(t-t_{i-1}) N_{i-1,k-1}(t)}{t_{i+k-2}-t_{i-1}}$$

$$+ \ldots$$

$$+ x_{i-k+2} \frac{(t_{i+2}-t) N_{i-k+3,k-1}(t)}{t_{i+2}-t_{i-k+3}} + x_{i-k+2} \frac{(t-t_{i-k+2}) N_{i-k+2,k-1}(t)}{t_{i+1}-t_{i-k+2}}$$

$$+ x_{i-k+1} \frac{(t_{i+1}-t) N_{i-k+2,k-1}(t)}{t_{i+1}-t_{i-k+2}} + x_{i-k+1} \frac{(t-t_{i-k+1}) N_{i-k+1,k-1}(t)}{t_i-t_{i-k+1}}$$

Supprimons le premier et le dernier terme qui ne participent pas à l'interpolation, et combinons deux par deux les termes qui restent :

$$x(t) = ((x_i-x_{i-1}) \frac{t-t_i}{t_{i+k-1}-t_i} + x_{i-1}) N_{i,k-1}(t)$$

$$+ \quad \ldots$$

$$+ ((x_{i-k+2}-x_{i-k+1}) \frac{t-t_{i-k+2}}{t_{i+1}-t_{i-k+2}} + x_{i-k+1}) N_{i-k+2,k-1}(t)$$

Calculons la dérivée première :

$$\frac{dx(t)}{dt} = \frac{x_i-x_{i-1}}{t_{i+k-1}-t_i} N_{i,k-1}(t)$$

$$+ ((x_i-x_{i-1}) \frac{t-t_i}{t_{i+k-1}-t_i} + x_{i-1}) \frac{dN_{i,k-1}(t)}{dt}$$

$$+ \ldots$$

$$+ \frac{x_{i-k+2}-x_{i-k+1}}{t_{i+1}-t_{i-k+2}} N_{i-k+2,k-1}(t)$$

$$+ ((x_{i-k+2}-x_{i-k+1}) \frac{t-t_{i-k+2}}{t_{i+1}-t_{i-k+2}} + x_{i-k+1}) \frac{dN_{i-k+2,k-1}(t)}{dt}$$

Les termes contenant $N_{i,k-1}(t)$, ..., $N_{i-k+2,k-1}(t)$ sont positifs ou nuls. Examinons maintenant en détail le premier terme contenant la dérivée :

$$(x_i - x_{i-1}) \frac{t-t_i}{t_{i+k-1}-t_i} \frac{dN_{i,k-1}(t)}{dt} + x_{i-1} \frac{dN_{i,k-1}(t)}{dt}$$

Si la dérivée est positive ou nulle, la première partie est positive ou nulle. Si la dérivée est négative, on peut combiner les deux parties comme suit :

$$(x_i - x_{i-1}) \frac{t-t_i}{t_{i+k-1}-t_i} \frac{dN_{i,k-1}(t)}{dt} + x_{i-1} \frac{dN_{i,k-1}(t)}{dt}$$

$$\geq (x_i - x_{i-1}) \frac{t_{i+k-1}-t_i}{t_{i+k-1}-t_i} \frac{dN_{i,k-1}(t)}{dt} + x_{i-1} \frac{dN_{i,k-1}(t)}{dt}$$

$$= x_i \frac{dN_{i,k-1}(t)}{dt}$$

La somme des termes restants (sauf ceux déjà considérés comme positifs ou nuls) est, selon l'hypothèse de récurrence pour l'interpolation d'ordre $k-1$, positive ou nulle. La dérivée de l'interpolation d'ordre k est donc positive ou nulle.

Si les points de contrôle ne sont pas tous les mêmes, on aura au moins dans la somme un terme qui est strictement positif. La fonction d'interpolation sera donc strictement croissante. ◊

Revenons maintenant à la fonction de reparamétrage. Si l'interpolation est cubique, on obtient une équation algébrique d'ordre trois :

$$a t^3 + b t^2 + c t + d = 0$$

La seule racine réelle est donnée par les formules suivantes :

$$p = \frac{c}{a} - \frac{b^2}{3a^2}$$

$$q = \frac{2b^3}{27a^3} - \frac{bc}{3a^2} + \frac{d}{a}$$

$$t = \left(-\frac{q}{2} + \left(\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3 \right)^{1/2} \right)^{1/3}$$

$$+ \left(-\frac{q}{2} + \left(\left(\frac{q}{2}\right)^2 - \left(\frac{p}{3}\right)^3 \right)^{1/2} \right)^{1/3}$$

$$-\frac{b}{3a}$$

2.2.4 Contrôle de la continuité

Examinons maintenant le problème de la continuité entre deux déformations libres de type B-spline en séquence. La position relative entre les deux repères locaux est la même que celle pour les déformations libres de type Bézier.

Par rapport à la formulation de type Bézier, on a introduit un reparamétrage avant l'interpolation volumique :

$$s_1 = f_{s1}(s'_1)$$

$$s_2 = f_{s2}(s'_2)$$

$$t = f_t(t')$$

$$u = f_u(u')$$

La continuité C^0 demande :

$$P_1(s'_1, t', u') = P_2(s'_2, t', u')$$

ce qui est équivalent à :

$$P_1(s_1, t, u) = P_2(s_2, t, u)$$

La condition pour la continuité C^0 est donc la même que celle de type Bézier.

La continuité C^1 demande en plus :

$$\frac{\partial P_1(s'_1,t',u')}{\partial s'_1} = a\frac{\partial P_2(s'_2,t',u')}{\partial s'_2}$$

$$\frac{\partial P_1(s'_1,t',u')}{\partial t'} = \frac{\partial P_2(s'_2,t',u')}{\partial t'}$$

$$\frac{\partial P_1(s'_1,t',u')}{\partial u'} = \frac{\partial P_2(s'_2,t',u')}{\partial u'}$$

où a est une constante positive,

ce qui est équivalent à :

$$\frac{\partial P_1(s_1,t,u)}{\partial s_1}\frac{ds_1}{ds'_1} = a\frac{\partial P_2(s_2,t,u)}{\partial s_2}\frac{ds_2}{ds'_2}$$

$$\frac{\partial P_1(s_1,t,u)}{\partial t} = \frac{\partial P_2(s_2,t,u)}{\partial t}$$

$$\frac{\partial P_1(s_1,t,u)}{\partial u} = \frac{\partial P_2(s_2,t,u)}{\partial u}$$

Comme les fonctions de reparamétrage sont strictement croissantes, $(a\frac{ds_2}{ds'_2} / \frac{ds_1}{ds'_1})$ est une valeur positive. On obtient donc la même condition pour la continuité C^1 que celle pour l'interpolation de type Bézier.

On voit que l'introduction du reparamétrage ne change pas les conditions de continuité.

Avec la formulation de la déformation libre de type B-spline, on est dispensé de s'occuper de la continuité entre deux déformations libres en séquence. On s'intéresse plutôt à la continuité entre une partie de l'objet qui subit une déformation libre et la partie restante de l'objet. C'est un cas particulier du cas précédent.

2.2.5 Transformation des normales

Nous utilisons la matrice Jacobienne pour transformer les normales.

La formulation de la déformation libre avec l'interpolation de type B-spline introduit une étape de reparamétrage. L'exécution du reparamétrage consiste à résoudre une équation algébrique. Pour calculer la matrice Jacobienne correspondante, on n'a pas besoin d'inverser la fonction. Il suffit de trouver la matrice Jacobienne correspondant à l'interpolation des points de contrôle non-déplacés et ensuite de l'inverser.

Les fonctions de mélange de type B-spline sont les suivantes :

$$N_{i,1}(t) = \begin{cases} 1 & si\ t_i \le t < t_{i+1} \\ 0 & sinon \end{cases}$$

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

avec la convention $\frac{0}{0} = 0$.

Sa dérivation est la suivante :

$$\frac{dN_{i,1}(t)}{dt} = 0$$

$$\frac{dN_{i,k}(t)}{dt} =$$

$$\frac{N_{i,k-1}(t) + (t-t_i)\frac{dN_{i,k-1}(t)}{dt}}{t_{i+k-1} - t_i}$$

$$+ \frac{-N_{i+1,k-1}(t) + (t_{i+k}-t)\frac{dN_{i+1,k-1}(t)}{dt}}{t_{i+k} - t_{i+1}}$$

Pour l'interpolation volumique de type B-spline, la dérivation est la suivante :

$$P(s,t,u) = \sum_{i=0}^{l} N_{i,ks}(s) \sum_{j=0}^{m} N_{j,kt}(t) \sum_{k=0}^{n} N_{k,ku}(u)\ P'_{ijk}$$

$$\frac{\partial P(s,t,u)}{\partial s} = \sum_{i=0}^{l} \frac{dN_{i,ks}(s)}{ds} \sum_{j=0}^{m} N_{j,kt}(t) \sum_{k=0}^{n} N_{k,ku}(u)\ P'_{ijk}$$

2.3 Déformation libre de type B-spline uniforme

Si on formule la déformation libre avec l'interpolation de type B-spline uniforme, nous allons montrer que le reparamétrage devient très simple.

2.3.1 Formulation

En nous limitant à l'interpolation cubique, nous pouvons déduire la forme explicite des fonctions de mélange :

$$N_{i,4}(t) = \begin{cases} \frac{1}{6}(t-i)^3 & i \le t < i+1 \\ \frac{1}{6}[-3(t-i-1)^3 + 3(t-i-1)^2 + 3(t-i-1) + 1] & i+1 \le t < i+2 \\ \frac{1}{6}[3(t-i-2)^3 - 6(t-i-2)^2 + 4] & i+2 \le t < i+3 \\ \frac{1}{6}[-(t-i-3)^3 + 3(t-i-3)^2 - 3(t-i-3) + 1] & i+3 \le t < i+4 \end{cases}$$

Dans l'intervalle du paramètre $[i, i+1)$, ce sont les fonctions de mélange $N_{i-3,4}(t)$, $N_{i-2,4}(t)$, $N_{i-1,4}(t)$, $N_{i,4}(t)$ qui participent à l'interpolation :

$$P(t) = N_{i-3,4}(t)\, P_{i-3} + N_{i-2,4}(t)\, P_{i-2} + N_{i-1,4}(t)\, P_{i-1} + N_{i,4}(t)\, P_i$$

où

$$N_{i-3,4}(t) = \frac{1}{6}(-(t-i)^3 + 3(t-i)^2 - 3(t-i) + 1)$$

$$N_{i-2,4}(t) = \frac{1}{6}(3(t-i)^3 - 6(t-i)^2 + 4)$$

$$N_{i-1,4}(t) = \frac{1}{6}(-3(t-i)^3 + 3(t-i)^2 + 3(t-i) + 1)$$

$$N_{i,4}(t) = \frac{1}{6}(t-i)^3$$

On peut noter ces fonctions de mélange sous la forme matricielle bien connue :

$$[N_{i-3,4}(t), N_{i-2,4}(t), N_{i-1,4}(t), N_{i,4}(t)]$$

$$= [(t-i)^3, (t-i)^2, t-i, 1]\, \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

Etant donnée une valeur du paramètre t, nous pouvons déterminer l'intervalle d'interpolation qui le contient comme suit :

$$t_i = int(t)$$

où *int(t)* donne la partie entière *t*.

2.3.2 Reparamétrage

Les fonctions de mélange de type B-spline uniforme vérifient la propriété suivante :

$$\sum_{i=0}^{n} N_{i,k}(t)\, i = t - \frac{k}{2}$$

Preuve :

$$\sum_{i=0}^{n} N_{i,k}(t)\, i$$

$$= \sum_{i=1}^{n} \left(i \frac{(t-i)\, N_{i,k-1}(t)}{k-1} + (i-1) \frac{(i-1+k-t)\, N_{i,k-1}(t)}{k-1} \right)$$

$$= \sum_{i=1}^{n} \frac{t-2i+1+(i-1)k}{k-1} N_{i,k-1}(t)$$

$$= \frac{k-2}{k-1} \sum_{i=1}^{n} i\, N_{i,k-1}(t) + \frac{1}{k-1} t - 1$$

Si k est égal à 2 , nous avons :

$$\sum_{i=k-2}^{n} N_{i,2}(t)\, i = t - 1$$

Pour un k quelconque, comme $\frac{k-2}{k-1} + \frac{1}{k-1} = 1$, nous avons :

$$\sum_{i=0}^{n} N_{i,k}(t)\, i = t + a_k$$

où a_k est indépendant de t .

Cherchons maintenant a_k :

$$a_2 = -1$$

$$a_k = \frac{k-2}{k-1} a_{k-1} - 1$$

On en déduit :

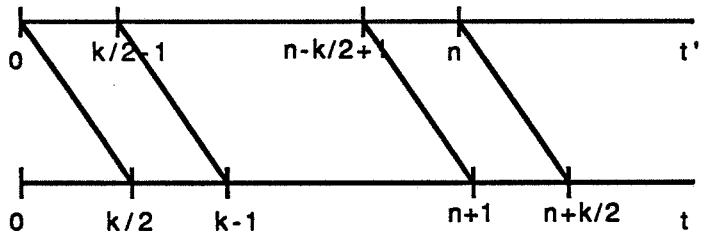
$$a_k = -\frac{k}{2}$$

$$\sum_{i=0}^{n} N_{i,k}(t)\, i = t - \frac{k}{2} \quad \Diamond$$

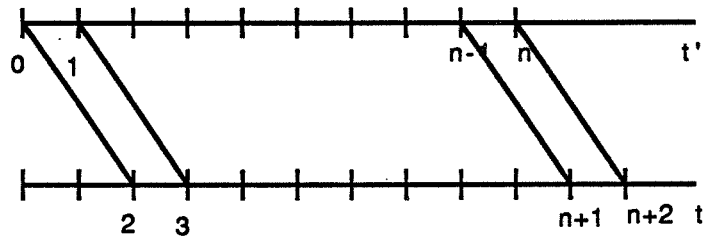
Avec cette propriété, le reparamétrage devient donc une simple translation :

$$t = t' + \frac{k}{2}$$

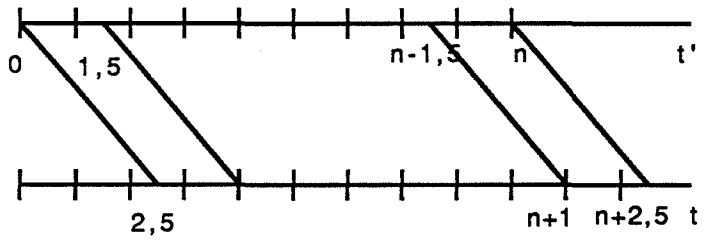
Cette translation est montrée sur la figure 2.2(a). Sur les figure 2.2(b) et 2.2(c), nous montrons le détail de la translation pour $k=4$ et $k=5$.



(a)



(b)



(c)

Figure 2.2 Translation du paramètre

Avec le reparamétrage sous forme d'une translation, la formulation des déformations libres avec l'interpolation de type B-spline uniforme est aussi simple qu'avec l'interpolation de type Bézier.

2.3.3 Contrôle de la continuité

Une courbe du type B-spline uniforme ne passe pas par les deux points de contrôle extrêmes. Pour la déformation libre formulée avec l'interpolation de type B-spline uniforme, la région de déformation est donc inférieure au maillage de points de contrôle, ce qui n'est pas naturel pour l'utilisateur. Avec l'interpolation cubique, nous pouvons résoudre ce problème d'une façon élégante.

Pour simplifier la présentation, regardons la déformation libre en une dimension. Nous avons un point sur l'objet à déformer :

$$P = P_0 + t'T \quad t' \in [0, n)$$

et les points de contrôle :

$$P = P_0 + iT \quad i = 0, 1, ..., n$$

Pour satisfaire la condition d'invariance, il faut faire un reparamétrage :

$$t = t' + 2$$

où $t' \in [0, n)$,

$t'+2 \in [2, n+2)$.

L'interpolation cubique des points de contrôle donne :

$$P(t) = \sum_{i=0}^{n} N_{i,4}(t)\ P'_i$$

où $t \in [3, n+1)$.

Pour que t ait le même domaine de paramètre que $t'+2$, nous ajoutons un point de contrôle fantôme à chaque extrémité, ce qui donne l'interpolation suivante :

$$P(t) = \sum_{i=-1}^{n+1} N_{i,4}(t)\ P_i$$

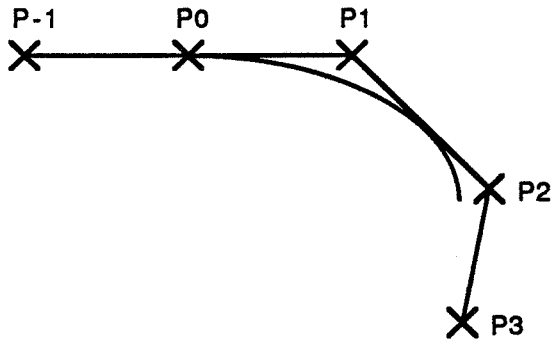


Figure 2.3 Point de contrôle fantôme

Sur la figure 2.3, le point de contrôle fantôme P_{-1} est caché pour l'utilisateur et utilisé seulement pour contrôler la continuité. En utilisant la matrice d'interpolation cubique de type B-spline, on a :

$$P(0) = \frac{1}{6}(P_{-1} + 4\,P_0 + P_1)$$

$$\frac{dP(0)}{dt} = \frac{1}{2}(P_1 - P_{-1})$$

$$\frac{d^2P(0)}{dt^2} = P_{-1} - 2\,P_0 + P_1$$

Si on demande :

$$P(0) = P_0$$

on a alors :

$$P_{-1} = 2\,P_0 - P_1$$

$$\frac{dP(0)}{dt} = P_1 - P_0$$

$$\frac{d^2P(0)}{dt^2} = 0$$

Nous laissons toujours P_0 à sa position initiale. Quand P_1 est déplacé, P_{-1} est déplacé automatiquement pour maintenir la continuité C^0. Pour avoir la continuité C^1, nous laissons P_1 inchangé. Dans ce dernier cas, la continuité C^2 est maintenue en même temps.

2.3.4 Transformation des normales

Pour l'interpolation cubique de type B-spline uniforme, les fonctions de mélange peuvent être représentées sous forme matricielle :

$$[N_{i-3,4}(t),\ N_{i-2,4}(t),\ N_{i-1,4}(t),\ N_{i,4}(t)]$$

$$= [(t-i)^3,\ (t-i)^2,\ t-i,\ 1]\,\frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

On peut aussi donner la dérivation sous forme matricielle :

$$\left[\frac{dN_{i-3,4}(t)}{dt},\ \frac{dN_{i-2,4}(t)}{dt},\ \frac{dN_{i-1,4}(t)}{dt},\ \frac{dN_{i,4}(t)}{dt}\right]$$

$$= [3(t-i)^2,\ 2(t-i),\ 1,\ 0]\,\frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

2.4 Déformation libre de type B-spline non-uniforme

Nous pouvons aussi utiliser l'interpolation de type B-spline non-uniforme pour formuler les déformations libres. En fait, l'interpolation de type Bézier est un cas particulier de l'interpolation de type B-spline qui utilise justement un vecteur nodal non-uniforme, et la déformation libre formulée avec l'interpolation de type Bézier satisfait la condition d'invariance. Mais en général, la condition d'invariance ne sera plus satisfaite et le reparamétrage consistera à résoudre une équation algébrique.

Nous allons dans ce qui suit étudier un cas particulier où le vecteur nodal est uniforme mais peut avoir des noeuds multiples. Il semble que ce soit l'utilisation la plus courante de l'interpolation de type B-spline non-uniforme.

2.4.1 Formulation

En nous limitant à l'interpolation cubique, nous allons calculer la forme explicite des fonctions de mélange.

Pour montrer la procédure de calcul, prenons le vecteur nodal suivant comme exemple :

$$[t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7] = [0, 0, 0, 0, 1, 2, 3, 4]$$

et regardons dans l'intervalle $[t_3, t_4)$. A partir de la définition récursive, nous calculons les fonctions de mélange comme suit :

Pour $k = 1$, on a :

$$N_{0,1}(t) = 0$$

$$N_{1,1}(t) = 0$$

$$N_{2,1}(t) = 0$$

$$N_{3,1}(t) = 1$$

$$N_{4,1}(t) = 0$$

$$N_{5,1}(t) = 0$$

$$N_{6,1}(t) = 0$$

Pour $k = 2$, on a :

$$N_{0,2}(t) = 0$$

$$N_{1,2}(t) = 0$$

$$N_{2,2}(t) = (1-t)\, N_{3,1}(t) = 1-t$$

$$N_{3,2}(t) = t\, N_{3,1}(t) + (2-t)\, N_{4,1}(t) = t$$

$$N_{4,2}(t) = t\, N_{4,1}(t) + (2-t)\, N_{5,1}(t) = 0$$

$$N_{5,2}(t) = t\, N_{5,1}(t) + (2-t)\, N_{6,1}(t) = 0$$

Pour $k = 3$, on a :

$$N_{0,3}(t) = 0$$

$$N_{1,3}(t) = (1-t)\, N_{2,2}(t) = (1-t)^2$$

$$N_{2,3}(t) = t\, N_{2,2}(t) + \frac{2-t}{2} N_{3,2}(t) = 2t - \frac{3}{2} t^2$$

$$N_{3,3}(t) = \frac{t}{2} N_{3,2}(t) + \frac{3-t}{2} N_{4,2}(t) = \frac{1}{2} t^2$$

$$N_{4,3}(t) = \frac{t-1}{2} N_{4,2}(t) + \frac{4-t}{2} N_{5,2}(t) = 0$$

Pour $k = 4$, on a :

$$N_{0,4}(t) = (1-t)\, N_{1,3}(t) = -t^3+3t^2-3t+1$$

$$N_{1,4}(t) = t\, N_{1,3}(t) + \frac{2-t}{2} N_{2,3}(t) = \frac{7}{4} t^3 - \frac{9}{2} t^2 + 3t$$

$$N_{2,4}(t) = \frac{t}{2} N_{2,3}(t) + \frac{3-t}{3} N_{3,3}(t) = -\frac{11}{12} t^3 + \frac{3}{2} t^2$$

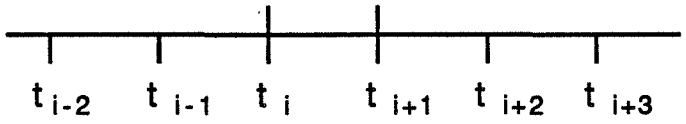
$$N_{3,4}(t) = \frac{t}{3} N_{3,3}(t) + \frac{4-t}{3} N_{4,3}(t) = \frac{1}{6} t^3$$

Nous pouvons écrire ces fonctions de mélange sous forme matricielle :

$$[N_{0,4}(t), N_{1,4}(t), N_{2,4}(t), N_{3,4}(t)]$$

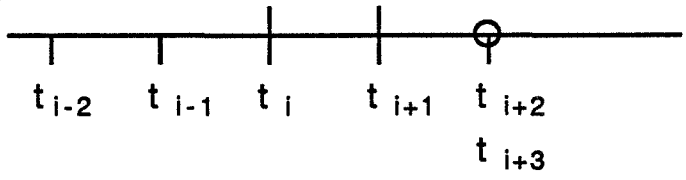
$$= [t^3, t^2, t, 1] \frac{1}{12} \begin{bmatrix} -12 & 21 & -11 & 2 \\ 36 & -54 & 18 & 0 \\ -36 & 36 & 0 & 0 \\ 12 & 0 & 0 & 0 \end{bmatrix}$$

Pour un intervalle non nul $[t_i, t_{i+1})$, nous avons au total seize configurations différentes (voir la figure 2.4). Nous avons calculé la matrice correspondant à chaque configuration.



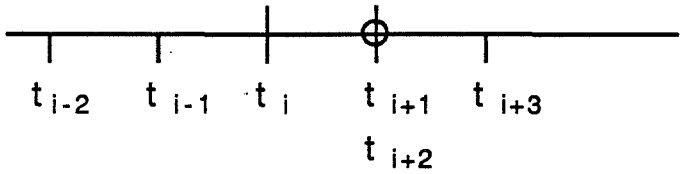
$$\frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

(a)



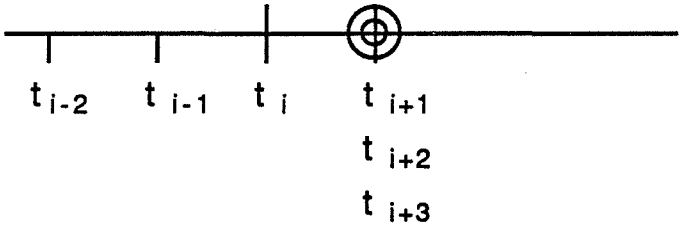
$$\frac{1}{12} \begin{bmatrix} -2 & 6 & -7 & 3 \\ 6 & -12 & 6 & 0 \\ -6 & 0 & 6 & 0 \\ 2 & 8 & 2 & 0 \end{bmatrix}$$

(b)



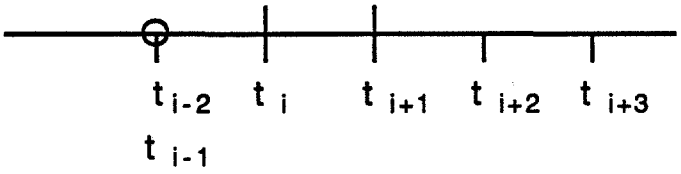
$$\frac{1}{12}\begin{bmatrix} -2 & 11 & -15 & 6 \\ 6 & -15 & 9 & 0 \\ -6 & -3 & 9 & 0 \\ 2 & 7 & 3 & 0 \end{bmatrix}$$

(c)



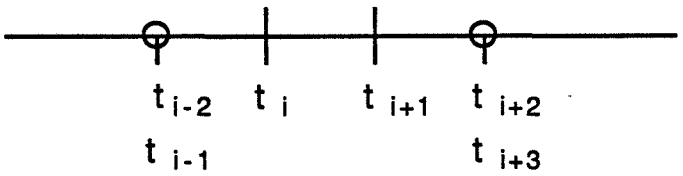
$$\frac{1}{12}\begin{bmatrix} -2 & 11 & -21 & 12 \\ 6 & -15 & 9 & 0 \\ -6 & -3 & 9 & 0 \\ 2 & 7 & 3 & 0 \end{bmatrix}$$

(d)



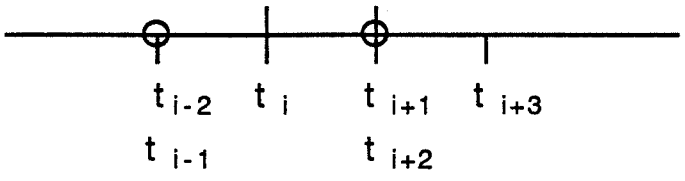
$$\frac{1}{12}\begin{bmatrix} -3 & 7 & -6 & 2 \\ 9 & -15 & 6 & 0 \\ -9 & 3 & 6 & 0 \\ 3 & 7 & 2 & 0 \end{bmatrix}$$

(e)



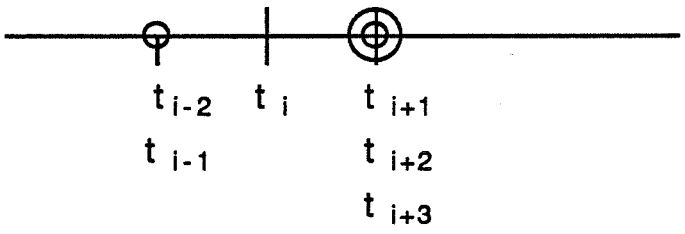
$$\frac{1}{12}\begin{bmatrix} -3 & 7 & -7 & 3 \\ 9 & -15 & 6 & 0 \\ -9 & 3 & 6 & 0 \\ 3 & 7 & 2 & 0 \end{bmatrix}$$

(f)



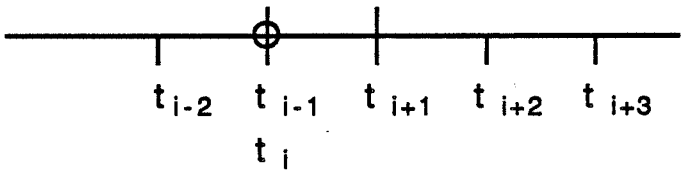
$$\frac{1}{12}\begin{bmatrix} -3 & 12 & -15 & 6 \\ 9 & -18 & 9 & 0 \\ -9 & 0 & 9 & 0 \\ 3 & 6 & 3 & 0 \end{bmatrix}$$

(g)



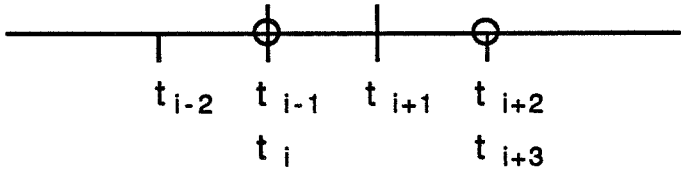
$$\frac{1}{12}\begin{bmatrix} -3 & 12 & -21 & 12 \\ 9 & -18 & 9 & 0 \\ -9 & 0 & 9 & 0 \\ 3 & 6 & 3 & 0 \end{bmatrix}$$

(h)



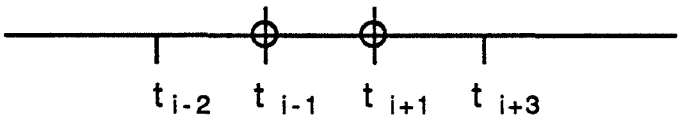
$$\frac{1}{12}\begin{bmatrix} -6 & 15 & -11 & 2 \\ 18 & -36 & 18 & 0 \\ -18 & 18 & 0 & 0 \\ 6 & 6 & 0 & 0 \end{bmatrix}$$

(i)



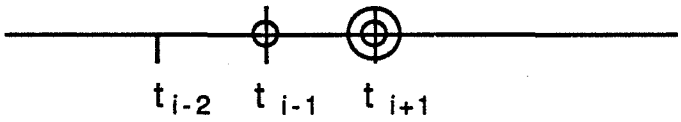
$$\frac{1}{12}\begin{bmatrix} -6 & 15 & -12 & 3 \\ 18 & -36 & 18 & 0 \\ -18 & 18 & 0 & 0 \\ 6 & 6 & 0 & 0 \end{bmatrix}$$

(j)



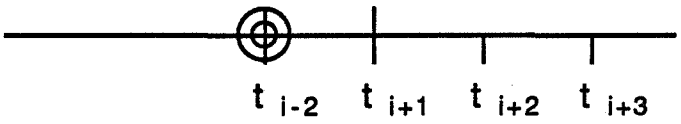
$$\frac{1}{12}\begin{bmatrix} -6 & 30 & -30 & 6 \\ 18 & -54 & 36 & 0 \\ -18 & 18 & 0 & 0 \\ 6 & 6 & 0 & 0 \end{bmatrix}$$

(k)



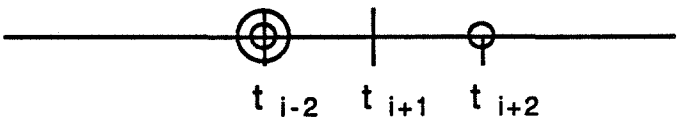
$$\frac{1}{12}\begin{bmatrix} -6 & 30 & -36 & 12 \\ 18 & -54 & 36 & 0 \\ -18 & 18 & 0 & 0 \\ 6 & 6 & 0 & 0 \end{bmatrix}$$

(l)



$$\frac{1}{12}\begin{bmatrix} -12 & 21 & -11 & 2 \\ 36 & -54 & 18 & 0 \\ -36 & 36 & 0 & 0 \\ 12 & 0 & 0 & 0 \end{bmatrix}$$

(m)



$$\frac{1}{12}\begin{bmatrix} -12 & 21 & -12 & 3 \\ 36 & -54 & 18 & 0 \\ -36 & 36 & 0 & 0 \\ 12 & 0 & 0 & 0 \end{bmatrix}$$

(n)

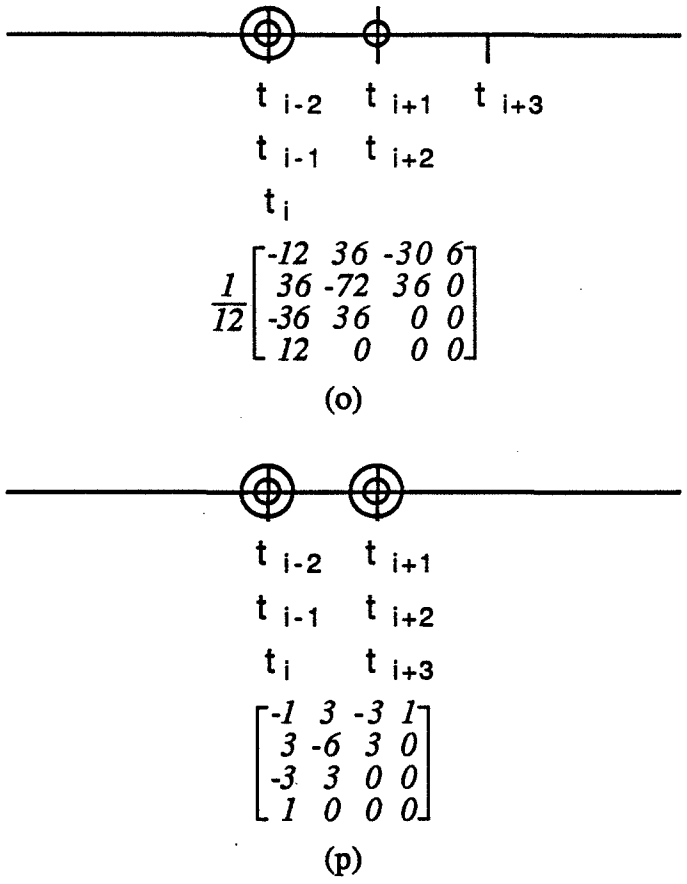


Figure 2.4 Vecteur nodal uniforme avec noeuds multiples

Etant donnée une valeur du paramètre *t*, nous pouvons déterminer, à partir du vecteur nodal, l'intervalle non nul qui la contient. A partir de la matrice *M* correspondant à la configuration de cet intervalle, on peut calculer les fonctions de mélange comme suit :

$$[N_{i-3}(t), N_{i-2}(t), N_{i-1}(t), N_i(t)] = [(t-t_i)^3, (t-t_i)^2, t-t_i, 1]\, M$$

Les vecteurs nodaux uniformes avec noeuds multiples peuvent être utilisés dans deux types d'applications. Dans le premier, nous utilisons le vecteur nodal suivant pour que la courbe ainsi définie passe par les deux points de contrôle extrêmes :

$$t_i = \begin{cases} 0 & 0 \le i \le 3 \\ i-3 & 4 \le i \le n \\ n-2 & n+1 \le i \le n+4 \end{cases}$$

Dans le deuxième type, nous utilisons les noeuds multiples pour introduire une discontinuité de différents ordres.

Dans la configuration (p) sur la figure 2.4, nous obtenons l'interpolation cubique de type Bézier. En fait, pour un n quelconque, si nous prenons k égal à $n+1$ et si nous utilisons le vecteur nodal du premier type d'application, l'interpolation de type B-spline sera une interpolation de type Bézier.

2.4.2 Reparamétrage

Soit (t_i) le vecteur nodal utilisé. A partir de l'équation qui définit implicitement la fonction de reparamétrage, nous pouvons calculer pour chaque noeud la coordonnée locale correspondante. Nous obtenons ainsi un autre vecteur (t'_i). Avec ce nouveau vecteur, nous pouvons déterminer pour une coordonnée locale quelconque l'intervalle non nul qui la contient, en utilisant une recherche linéaire.

Prenons comme exemple le vecteur nodal pour le premier type d'application :

$$0, 0, 0, 0, 1, 2, 3, ..., n\text{-}5, n\text{-}4, n\text{-}3, n\text{-}2, n\text{-}2, n\text{-}2, n\text{-}2$$

Le vecteur de coordonnées locales correspondant est le suivant :

$$0, 0, 0, 0, \frac{23}{12}, 3, 4, ..., n\text{-}4, n\text{-}3, n\text{-}\frac{23}{12}, n, n, n, n$$

L'intervalle d'interpolation $0 \leq t < 1$ correspond donc à l'intervalle de coordonnées locales $0 \leq t' < \frac{23}{12}$ (voir la figure 2.5) .

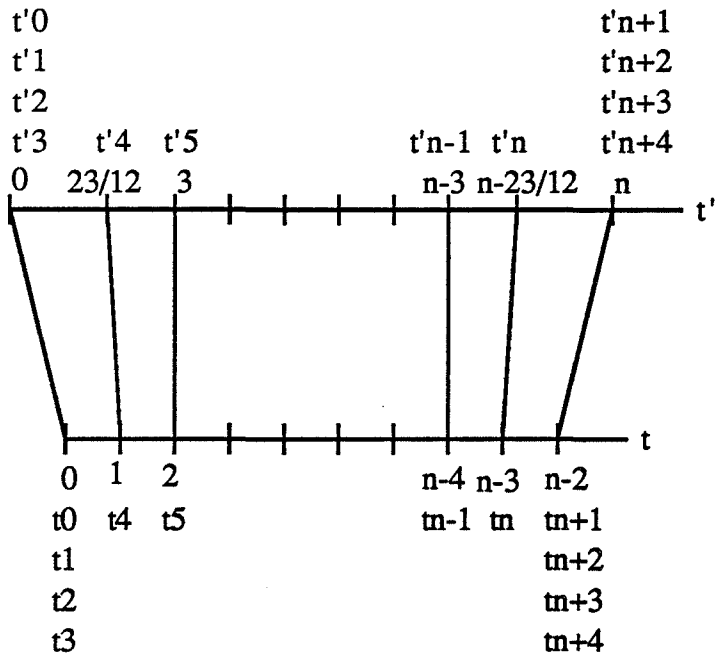


Figure 2.5 Vecteur nodal et vecteur de coordonnées locales

Etant donné un point P dans la région de déformation, nous calculons d'abord sa coordonnée locale t'. Nous déterminons ensuite l'intervalle d'interpolation $[t_i, t_{i+1})$, en utilisant le vecteur de coordonnées locales. Dans cet intervalle, ce sont les fonctions de mélange $N_{i-3,4}(t), N_{i-2,4}(t), N_{i-1,4}(t), N_{i,4}(t)$ qui participent à l'interpolation :

$$P(t) = N_{i-3,4}(t)\, P'_{i-3} + N_{i-2,4}(t)\, P'_{i-2} + N_{i-1,4}(t)\, P'_{i-1} + N_{i,4}(t)\, P'_i$$

La condition d'invariance impose :

$$t' = N_{i-3,4}(t)\,(i-3) + N_{i-2,4}(t)\,(i-2) + N_{i-1,4}(t)\,(i-1) + N_{i,4}(t)\, i$$

Prenons comme exemple le premier intervalle d'interpolation $[0, 1)$ dans la première application, qui correspond à la configuration (m) sur la figure 2.4. A partir de la matrice correspondante, on obtient l'équation algébrique suivante :

$$5t^3 - 18t^2 + 36t - 12t' = 0$$

Dans le premier type d'application, le reparamétrage dans les deux premiers intervalles d'interpolation $[0, 1)$ et $[1, 2)$ demande la résolution d'une équation algébrique d'ordre 3. A partir du troisième intervalle, l'interpolation devient uniforme, le reparamétrage se réduisant à une simple translation.

2.4.3 Contrôle de la continuité

Dans le premier type d'application, nous utilisons le vecteur nodal suivant :

$$0, 0, 0, 0, 1, 2, 3, ..., n-5, n-4, n-3, n-2, n-2, n-2, n-2$$

Comme nous l'avons dit, les conditions de continuité entre deux déformations libres de type B-spline en séquence sont les mêmes que les conditions de continuité entre deux courbes de type B-spline en séquence. Nous nous intéressons à la continuité entre une partie de l'objet qui subit la déformation libre et la partie restante de l'objet. Pour maintenir la continuité C^0 (respectivement C^1, C^2), il faut que la première (respectivement les deux premières, les trois premières) couche de points de contrôle reste invariante.

2.5 Déformation libre mixte

Dans la déformation libre, comme les interpolations en trois dimensions sont indépendantes, nous pouvons utiliser des interpolations de type différent. Si le nombre de couches de points de contrôle en une dimension est inférieur ou égal à quatre, nous utiliserons l'interpolation de type Bézier. Dans le cas contraire, nous utiliserons l'interpolation de type B-spline.

Comme l'utilisation de l'interpolation de type B-spline non-uniforme demande un reparamétrage, nous préférons utiliser l'interpolation de type B-spline uniforme, avec des points de contrôle fantômes aux deux extrémités. Malheureusement, on ne peut utiliser l'interpolation de type B-spline uniforme qu'en une seule dimension : si on utilise l'interpolation uniforme en deux dimensions comme sur la figure 2.6, le point fantôme P_{-1-1} ne peut pas satisfaire en même temps les deux conditions suivantes :

$$P_{-1-1} = 2\,P_{-10} - P_{-11}$$

$$P_{-1-1} = 2P_{0-1} - P_{1-1}$$

Dans ce cas, il faut utiliser en deuxième dimension une interpolation non-uniforme.

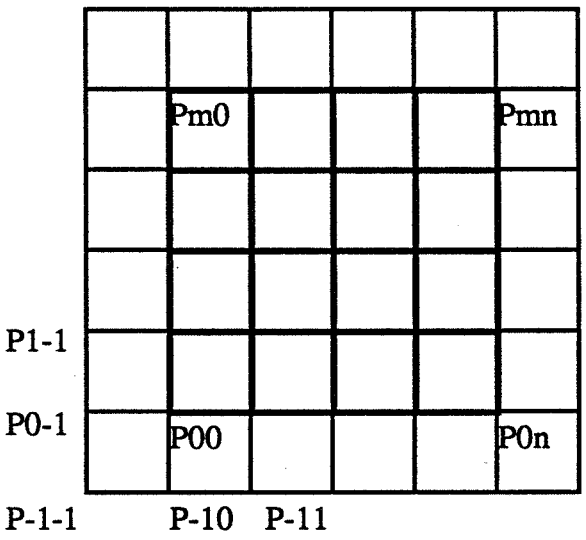


Figure 2.6 Interpolation de type B-spline uniforme en deux dimensions

Nous représentons une déformation libre par la structure de données suivante :

```
typedef struct {
        int n[3];
        double mat[4][4], mat1[4][4];
        double *p;
} FfdGrid;
```

où *n[0]+1*, *n[1]+1* et *n[2]+1* représentent le nombre de couches des points de contrôle en chaque dimension ;

mat est la matrice de transformation du repère du monde dans le repère local ;

mat1 est la matrice inverse ;

p est la liste des points de contrôle.

Voici le programme de la déformation libre mixte :

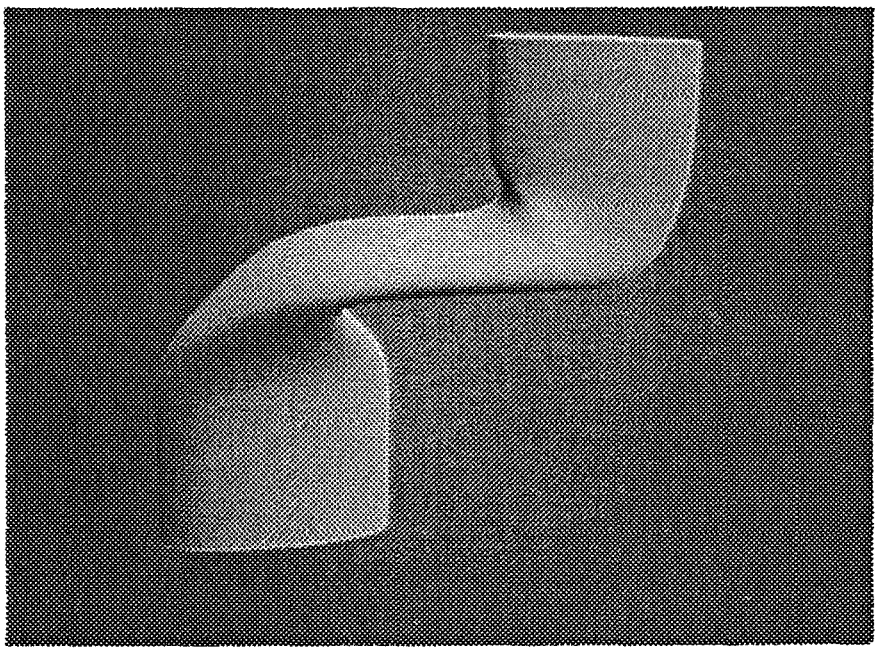
```
ffd (grid, point, norm)
FfdGrid *grid;
double *point;
double *norm;
{
        int d, e;
        int i0[3], m[3];
```

```
double f[3][4], df[3][4];
int i, j, k;
double *p;
double p1[3], p2[3]; /* le résultat d'interpolations bidimensionnelle et unidimensionnelle */
double jacob[3][3], jacob1[3][2], jacob2[3]; /* les matrices jacobiennes correspondant aux interpolations tridimensionnelle, bidimensionnelle et unidimensionnelle */

/* calculer les coordonnées locales */
affin(grid->mat, point, norm);

/* si le point est en dehors de la région de déformation, sortir de la fonction */
for(d=0; d<3; d++){
    if(point[d] < 0. II point[d] >= grid->n[d]){
        return(0);
    }
}

flag = 0;
for(d=0; d<3; d++){
    if(n[d] <= 3){
        /* calculer les fonctions de mélange de type Bézier */
        bezier(point[d], grid->n[d], f[d], df[d]);
        i0[d] = 0;
        m[d] = grid->n[d];
    }
    else{
        if(flag == 0){ /* si l'interpolation de type B-spline uniforme n'a pas encore été utilisée */
            /* calculer les fonctions de mélange de type B-spline uniforme */
            i0[d] = bspline_uniform(point[d]+2.,
                f[d], df[d]);
            m[d] = 3;
```

```
                    flag = 1;
                }
                else{
                    /* calculer les fonctions de mélange de type
B-spline non-uniforme avec reparamétrage */
                    i0[d] = bspline_mulnode(point[d],
                        grid->n[d], f[d], df[d]);
                    m[d] = 3;
                }
            }
        }

        /* effectuer l'interpolation volumique */
        for(e=0; e<3; e++){
            point[e] = 0.;
            for(d=0; d<3; d++){
                jacob[e][d] = 0.;
            }
        }
        for(i=0; i<=m[0]; i++){
            for(e=0; e<3; e++){
                p1[e] = 0.;
                for(d=0; d<2; d++){
                    jacob1[e][d] = 0.;
                }
            }
            for(j=0; j<=m[1]; j++){
                for(e=0; e<3; e++){
                    p2[e] = 0.;
                    jacob2[e] = 0.;
                }
                for(k=0; k<=m[2]; k++){
                    p = pf(grid, i+i0[0], j+i0[1], k+i0[2]);
                    for(e=0; e<3; e++){
                        p2[e] += f[2][k]*p[e];
                        jacob2[e] += df[2][k]*p[e];
                    }
                }
```

```
                    for(e=0; e<3; e++){
                        p1[e] += f[1][j]*p2[e];
                        jacob1[e][0] += df[1][j]*p2[e];
                        jacob1[e][1] += f[1][j]*jacob2[e];
                    }
                }
                for(e=0; e<3; e++){
                    point[e] += f[0][i]*p1[e];
                    jacob[e][0] += df[0][i]*p1[e];
                    jacob[e][1] += f[0][i]*jacob1[e][0];
                    jacob[e][2] += f[0][i]*jacob1[e][1];
                }
            }

            /* transformer la normale avec la matrice Jacobienne */
            jacob_norm(jacob, norm);

            return(1);
        }
```

La figure 2.7 montre la déformation d'un cylindre avec un seul maillage de points de contrôle. En utilisant l'interpolation de type B-spline, le calcul reste cubique.



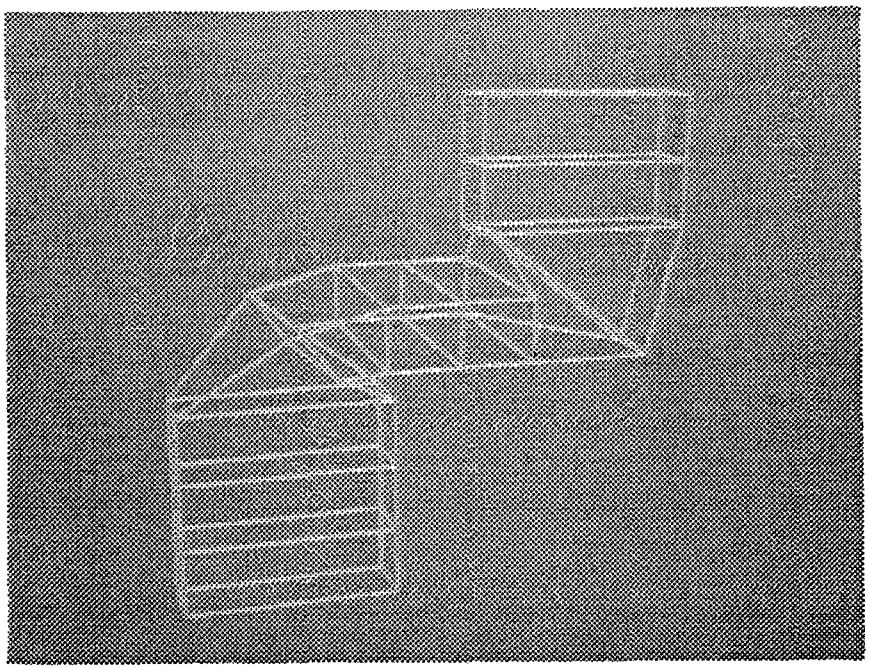


Figure 2.7 Un cylindre subit une déformation libre de type B-spline

2.6 Conclusions

La déformation libre de type B-spline rend l'ordre d'interpolation indépendant du nombre de points de contrôle utilisés. Avec une seule déformation libre, on peut réaliser une déformation complexe, tout en n'utilisant qu'une interpolation cubique. Par rapport à l'utilisation de plusieurs déformations libres de type Bézier en séquence, la déformation libre de type B-spline permet le contrôle local. En revanche, la déformation libre de type B-spline demande un reparamétrage avant l'interpolation volumique. En utilisant l'interpolation de type B-spline uniforme, le reparamétrage se réduit à une simple translation.

3 Déformations libres cylindrique et sphérique

La déformation libre utilise un repère local cartésien et forme ainsi une région de déformation de forme cubique. Cette formulation ne paraît pas adéquate pour réaliser les déformations qui peuvent être définies comme étant tri-linéaires dans un repère cylindrique ou sphérique. Pour remédier à ce problème, nous proposons la déformation libre cylindrique et la déformation libre sphérique qui utilisent respectivement un repère local cylindrique et un repère local sphérique.

Dans le premier paragraphe, nous expliquerons pourquoi formuler la déformation libre avec un repère local cylindrique ou sphérique. Dans le deuxième paragraphe, nous définirons la déformation libre cylindrique. Dans le troisième paragraphe, nous définirons la déformation libre sphérique. Nous conclurons dans le quatrième paragraphe.

3.1 Justification des déformations libres cylindrique et sphérique

La formulation de [SEDE 86] utilise un repère local cartésien, ce qui donne une région de déformation de forme cubique. [COQU 90] utilise une région de déformation quelconque pour formuler la déformation libre. Par exemple, une région de forme cylindrique est proposée. Avec cette nouvelle formulation, les paramètres d'interpolation doivent être calculés en résolvant un système d'équations non-linéaires.

Nous utilisons un repère local cylindrique pour formuler la déformation libre cylindrique, et un repère local sphérique pour la déformation libre sphérique. Par rapport à celle de [COQU 90], notre formulation se limite à définir une région de déformation de forme cylindrique ou sphérique. Par contre, nous évitons la résolution d'équations, en utilisant l'interpolation dans le repère cylindrique ou sphérique.

Pour réaliser une déformation globale, on peut appliquer directement la transformation à un objet. Pour simplifier le calcul, on peut aussi utiliser la déformation libre pour la réaliser en appliquant la déformation globale sur le maillage de points de contrôle. La déformation libre ainsi définie est appliquée sur l'objet. En général, le

résultat de la déformation libre ne sera qu'une approximation de la déformation globale. Mais pour certaines déformations globales, le résultat sera exact. De manière plus précise, on a le résultat suivant :

Le résultat de la déformation libre sera exact si la déformation globale peut être exprimée dans le repère local comme une transformation tri-linéaire :

$$s' = a_1\,stu + a_2\,st + a_3\,tu + a_4\,su + a_5\,s + a_6\,t + a_7\,u + a_8$$

$$t' = b_1\,stu + b_2\,st + b_3\,tu + b_4\,su + b_5\,s + b_6\,t + b_7\,u + b_8$$

$$u' = c_1\,stu + c_2\,st + c_3\,tu + c_4\,su + c_5\,s + c_6\,t + c_7\,u + c_8$$

où (s', t', u') représente la position transformée du point (s, t, u).

Preuve :

Les points de contrôle sont les suivants :

$$P_{ijk} = P_0 + iS + jT + kU$$

Leurs coordonnées locales sont les suivantes :

$$s_{ijk} = i$$

$$t_{ijk} = j$$

$$u_{ijk} = k$$

Pour simplifier la présentation, regardons la transformation tri-linéaire suivante :

$$s' = a\,stu$$

$$t' = b\,t$$

$$u' = c$$

Appliquons cette transformation sur le maillage de points de contrôle :

$$s'_{ijk} = a\ ijk$$

$$t'_{ijk} = b\ j$$

$$u'_{ijk} = c$$

$$P'_{ijk} = P_0 + a\ ijk\ S + b\ j\ T + c\ U$$

Soit P un point sur l'objet à déformer :

$$P = P_0 + sS + tT + uU$$

Pour trouver sa nouvelle position, effectuons l'interpolation volumique avec le maillage de points de contrôle déplacés. En utilisant la propriété d'invariance, nous obtenons :

$$P(s, t, u) = \sum_{i=0}^{l} N_i(s) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} N_k(u)\ P'_{ijk}$$

$$= \sum_{i=0}^{l} N_i(s) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} N_k(u)\ P_0$$

$$+ \sum_{i=0}^{l} N_i(s)i \sum_{j=0}^{m} N_j(t)j \sum_{k=0}^{n} N_k(u)k\ aS$$

$$+ \sum_{i=0}^{l} N_i(s) \sum_{j=0}^{m} N_j(t)j \sum_{k=0}^{n} N_k(u)\ bT$$

$$+ \sum_{i=0}^{l} N_i(s) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} N_k(u)\ cU$$

$$= P_0 + a\ stu\ S + b\ t\ T + c\ U$$

Nous retrouvons la transformation initiale :

$$s' = a\ stu$$

$$t' = b\ t$$

$$u' = c$$

Pour la transformation tri-linéaire générale, la démonstration sera similaire. ◊

Les transformations affines et l'effilement linéaire sont par exemple des transformations tri-linéaires.

Il y a des transformations qui ne sont pas tri-linéaires dans le repère cartésien, mais qui le sont dans le repère cylindrique ou sphérique. Pour mieux approximer ces transformations avec la déformation libre, nous allons utiliser un repère local cylindrique ou sphérique pour définir la déformation libre.

La torsion linéaire est une transformation complexe dans le repère cartésien. Dans le repère cylindrique, elle peut être représentée comme une transformation tri-linéaire :

$$r' = r$$

$$a' = a + k\,(h-h_1)$$

$$h' = h$$

où (r, a, h) sont des coordonnées dans le repère cylindrique.

Regardons une autre déformation globale définie comme une transformation tri-linéaire dans le repère cylindrique :

$$r' = r + h$$

$$a' = a$$

$$h' = h$$

Dans le repère cartésien, cette déformation est une transformation complexe :

$$x' = x + \frac{x\ z}{(x^2 + y^2)^{1/2}}$$

$$y' = y + \frac{y\ z}{(x^2 + y^2)^{1/2}}$$

$$z' = z$$

3.2 Déformation libre cylindrique

Nous allons présenter l'utilisation d'un repère local cylindrique pour formuler la déformation libre.

3.2.1 Interpolation cylindrique

Nous utilisons le repère cylindrique défini sur la figure 3.1, où r est le rayon, a l'angle et h la hauteur.

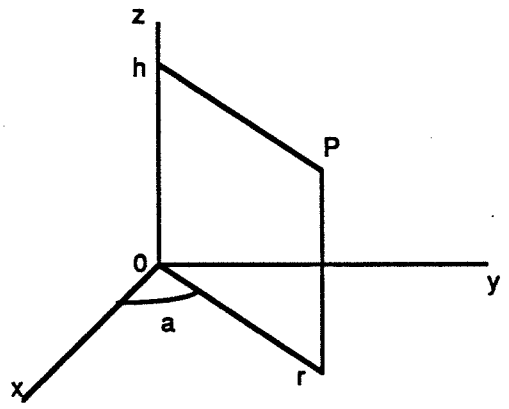


Figure 3.1 Repère cylindrique

Etant données les coordonnées cylindriques (r, a, h) d'un point P, ses coordonnées cartésiennes (x, y, z) peuvent être calculées comme suit :

$$x = r\ cos\frac{2\pi a}{m}$$

$$y = r\ sin\frac{2\pi a}{m}$$

$$z = h$$

où m est entier.

Nous utiliserons le repère cylindrique pour définir la déformation libre cylindrique. Pour respecter la convention de paramétrage de l'interpolation de type B-spline, le repère cylindrique défini ici est un peu différent du repère cylindrique classique. L'angle a appartient à l'intervalle $[0, m)$ au lieu de $[0, 2\pi)$. En fait, m sera le nombre de couches de points de contrôle dans la direction a.

La transformation inverse est la suivante :

$$r = (x^2 + y^2)^{1/2}$$

$$a = atan(m, y, x)$$

$$= \frac{m}{2\pi} \begin{cases} arctg\frac{y}{x} & x>0 \ , \ y\geq 0 \\ 2\pi + arctg\frac{y}{x} & x>0 \ , \ y<0 \\ \pi + arctg\frac{y}{x} & x<0 \\ \frac{\pi}{2} & x=0 \ , \ y>0 \\ \frac{3\pi}{2} & x=0 \ , \ y<0 \\ 0 & x=0 \ , \ y=0 \end{cases}$$

$$h = z$$

Pour un point situé sur l'axe cylindrique $(x = y = 0)$, l'angle a n'est pas défini de manière unique. Par convention, on prend a égal à 0.

Rappelons la définition classique de l'interpolation de type B-spline :

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

où les points de contrôle P_i et le point obtenu P sont tous en coordonnées cartésiennes.

Avec l'interpolation de type B-spline, on n'arrive pas à définir un cercle. Pour pouvoir le faire, [ALIA 87] a proposé la spline angulaire. Cette spline angulaire interpole

quatre points de contrôle successifs en mélangeant le cercle interpolant les trois premiers points et le cercle interpolant les trois derniers points. Avec la spline angulaire, on peut définir facilement un cercle. Par contre, cette spline angulaire est définie dans un plan. Il est difficile de la généraliser dans l'espace. De plus, les fonctions de mélange ne sont pas polynomiales.

Nous généralisons la définition classique de l'interpolation de type B-spline directement dans le repère cylindrique. Nous utilisons la même formule d'interpolation. Mais les points de contrôle et le point obtenu sont en coordonnées cylindriques. La signification d'une telle interpolation est illustrée sur la figure 3.2. Au lieu de donner un segment de droite, l'interpolation linéaire entre deux points P_1 et P_2 donne un segment d'hélice.

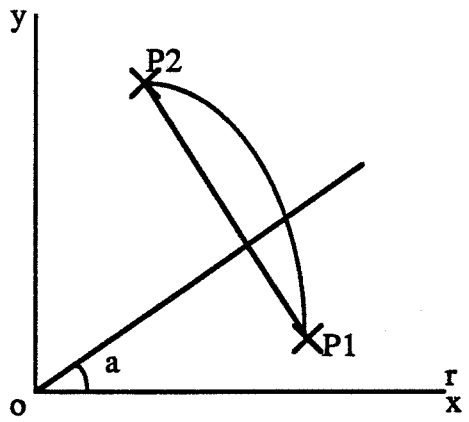


Figure 3.2 Interpolation cylindrique

Considérons des points de contrôle $P_0, ..., P_n$ en coordonnées cartésiennes. Pour définir la courbe d'interpolation cylindrique, nous calculons d'abord les coordonnées cylindriques des points de contrôle. Nous effectuons ensuite l'interpolation de type B-spline dans le repère cylindrique :

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

où P_i et $P(t)$ sont tous en coordonnées cylindriques.

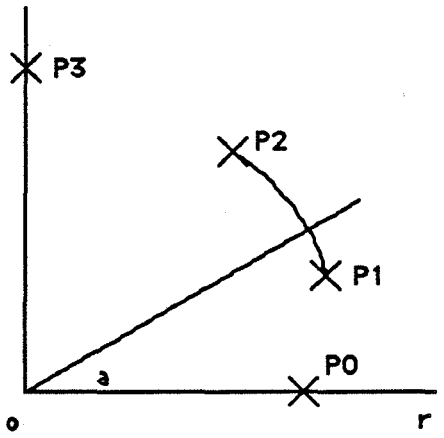
Nous calculons enfin les coordonnées cartésiennes du point obtenu.

Les transformations entre les coordonnées cylindriques et les coordonnées cartésiennes peuvent être optimisées en utilisant la technique de la tabulation. Avant d'effectuer l'interpolation cylindrique, nous remplissons les deux tableaux suivants :

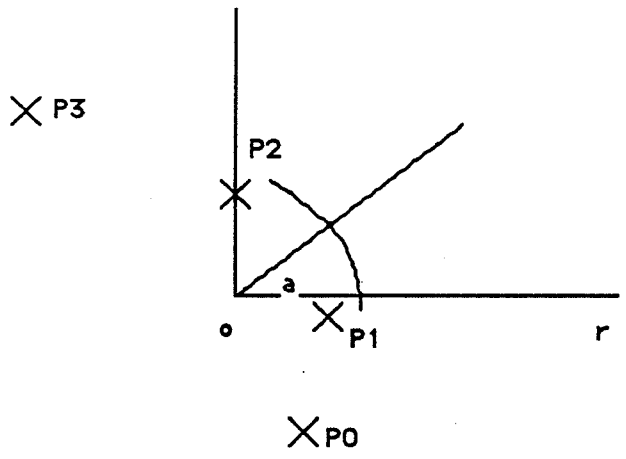
$$qsin(i) = sin\frac{i\pi}{2m} \qquad 0 \leq i < m$$

$$qarctg(i) = arctg\frac{i}{n} \qquad 0 \leq i < n$$

Avec l'interpolation cylindrique, nous pouvons définir facilement un cercle (voir la figure 3.3(a)). Par contre, cette interpolation ne possède pas la propriété d'invariance de l'interpolation classique. Si nous appliquons une transformation rigide (une translation suivie d'une rotation) à l'ensemble des points de contrôle, nous obtiendrons une courbe différente (voir la figure 3.3(b)). Cette différence est due au fait qu'une transformation rigide est une transformation linéaire dans le repère cartésien, mais pas dans le repère cylindrique. En fait, on peut définir une propriété d'invariance dans le repère cylindrique qui ne sera pas vérifiée par l'interpolation classique.



(a)



(b)

Figure 3.3 Interpolation cylindrique et propriété d'invariance

Il faut porter une attention particulière à la coordonnée a. Sur la figure 3.4(a), les points de contrôle sont :

$$P_0 = (0.20, -\frac{\pi}{2})$$

$$P_1 = (0.25, -\frac{\pi}{6})$$

$$P_2 = (0.30, \frac{\pi}{6})$$

$$P_3 = (0.35, \frac{\pi}{2})$$

Sur la figure 3.4(b), les points de contrôle ont des coordonnées cylindriques différentes :

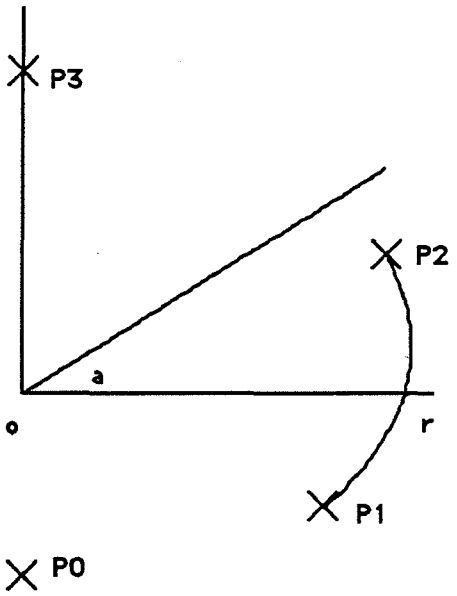
$$P_0 = (0.20, 2\pi-\frac{\pi}{2})$$

$$P_1 = (0.25, 2\pi-\frac{\pi}{6})$$

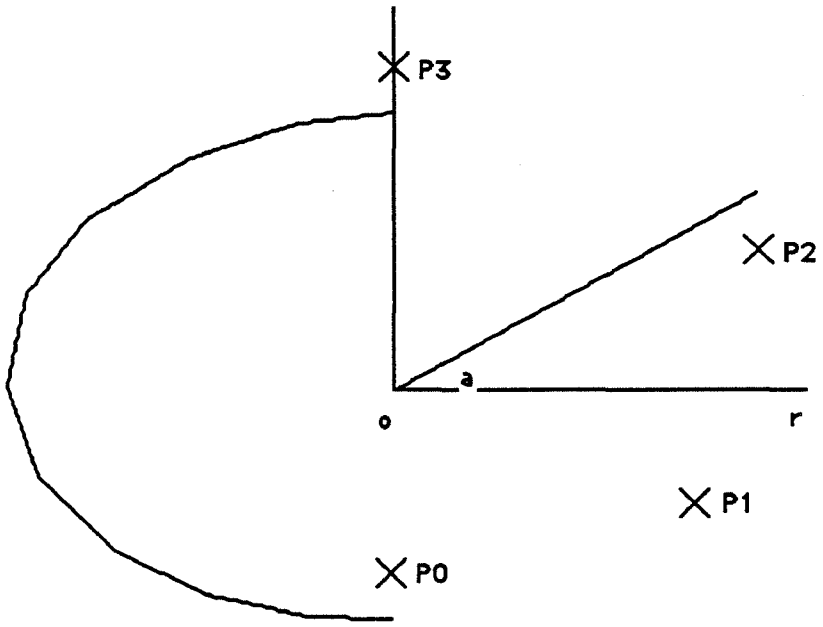
$$P_2 = (0.30, \frac{\pi}{6})$$

$$P_3 = (0.35, \frac{\pi}{2})$$

On obtient alors une courbe complètement différente.



(a)



(b)

Figure 3.4 Propriété de l'interpolation cylindrique

Avec l'interpolation classique, la courbe est contenue dans l'enveloppe convexe des points de contrôle. Avec l'interpolation cylindrique, on effectue l'interpolation dans le

repère cylindrique. Pour trouver un polyèdre qui contiendra la courbe, nous remplaçons chaque point de contrôle :

$$P_i = (r_i , a_i , h_i)$$

par les quatre sommets du rectangle qui englobe le cercle centré à l'origine et passant par le point de contrôle (voir la figure 3.5) :

$$P_{i1} = (r_i , \frac{\pi}{4} , h_i)$$

$$P_{i2} = (r_i , \frac{3\pi}{4} , h_i)$$

$$P_{i3} = (r_i , \frac{5\pi}{4} , h_i)$$

$$P_{i4} = (r_i , \frac{7\pi}{4} , h_i)$$

Après avoir remplacé de cette façon tous les points de contrôle, nous calculons l'enveloppe convexe des nouveaux points. La courbe sera contenue dans cette enveloppe convexe.

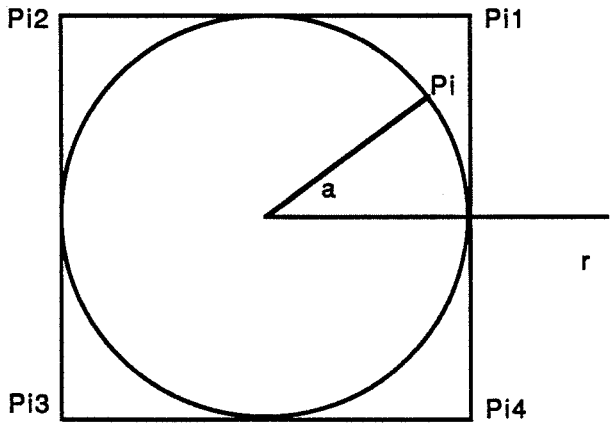


Figure 3.5 Enveloppe convexe pour l'interpolation cylindrique

3.2.2 Définition de la déformation libre cylindrique

Nous allons maintenant formuler la déformation libre en utilisant l'interpolation cylindrique.

Avec la déformation libre classique, nous définissons un repère local cartésien avec un point P_0 et trois vecteurs S, T et U. Nous formons ainsi une région de déformation de forme cubique qui contient l'objet à déformer. Pour la déformation libre cylindrique, nous utilisons comme repère local le repère cylindrique qui est associé au repère local cartésien. Nous formons ainsi une région de déformation de forme cylindrique (voir la figure 3.6). Chaque point P a des coordonnées (r, a, h) dans le repère local cylindrique et (s, t, u) dans le repère local cartésien telles que :

$$s = r\ cos\frac{a\,2\pi}{m}$$

$$t = r\ sin\frac{a\,2\pi}{m}$$

$$u = h$$

$$P = P_0 + sS + tT + uU$$

Nous définissons dans le repère local cylindrique un maillage de points de contrôle :

$$P_{ijk} = (r_{ijk}, a_{ijk}, h_{ijk}) = (i, j, k) \qquad 0 \le i \le l\ ,\ 0 \le j < m\ ,\ 0 \le k \le n$$

où $l+1, m$ et $n+1$ sont respectivement le nombre de couches de points de contrôle dans les directions r, a et h.

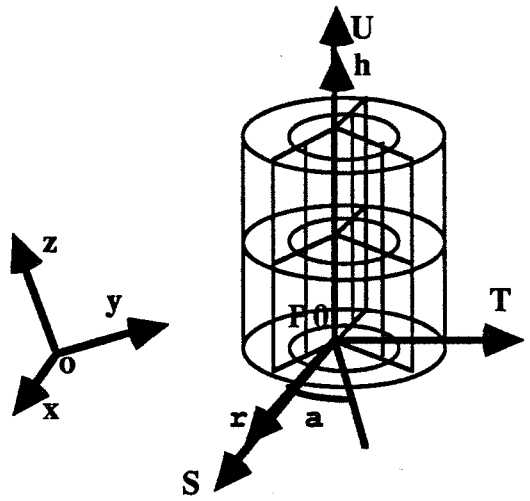


Figure 3.6 Repère local cylindrique

Nous déplaçons les points de contrôle dans le repère local cylindrique pour spécifier la déformation :

$$P'_{ijk} = F(P_{ijk})$$

Le déplacement des points de contrôle doit être fait dans le repère local cylindrique et non dans le repère cartésien du monde. En effet, si on déplace dans le repère cylindrique un point de contrôle en $(r, 2\pi+a, h)$, au lieu de le déplacer en (r, a, h), on obtiendra le même déplacement dans le repère cartésien. Par contre, les deux déplacements définiront deux interpolations cylindriques différentes.

Pour un point P sur l'objet à déformer, nous calculons d'abord ses coordonnées locales cylindriques :

$$s = \frac{T \wedge U.(P-P_0)}{T \wedge U.S}$$

$$t = \frac{S \wedge U.(P-P_0)}{S \wedge U.T}$$

$$u = \frac{S \wedge T.(P-P_0)}{S \wedge T.U}$$

$$r' = (s^2 + t^2)^{1/2}$$

$$a' = atan(m, t, s)$$

$$h' = u$$

Nous effectuons le reparamétrage pour satisfaire la condition d'invariance :

$$r = f_r(r')$$

$$a = f_a(a')$$

$$h = f_h(h')$$

En utilisant ces nouveaux paramètres, nous effectuons une interpolation volumique de type B-spline.

Comme nous l'avons dit, avec le repère local cylindrique, nous formons une région de déformation de forme cylindrique. Si on fait varier la coordonnée r en fixant les deux autres coordonnées, on obtient une ligne droite. De même, si on fait varier la coordonnée h en fixant les deux autres coordonnées, on obtient aussi une ligne droite. Par contre, si on fait varier la coordonnée a en fixant les deux autres coordonnées, on obtient un cercle, qui est une courbe fermée. Pour l'interpolation volumique, nous utilisons donc l'interpolation uniforme périodique pour la coordonnée a :

$$P''_{ijk} = P'_{i,mod(j,m),k} + \begin{bmatrix} 0 \\ j\text{-}mod(j,m) \\ 0 \end{bmatrix}$$

$$P(r,a,h) = \sum_{i=0}^{l} N_{i,kr}(r) \sum_{j=0}^{m+ka-1} N_{j,ka}(a) \sum_{k=0}^{n} N_{k,kh}(h)\ P''_{ijk}$$

où P'_{ijk} sont les points de contrôle déplacés en coordonnées cylindriques,

$mod(j, m)$ donne le reste de la division de j par m .

Le modulo dans l'indice indique que l'interpolation périodique est utilisée pour la coordonnée a . Le modulo dans le terme supplémentaire sert à définir une interpolation raisonnable : quand on utilise le point de contrôle P_0 comme le point suivant du point de contrôle P_n, sa coordonnée a est augmentée de m .

Après l'interpolation volumique dans le repère local cylindrique, on peut transformer le point obtenu du repère local cylindrique dans le repère cartésien du monde :

$$s = r\,cos\frac{a\,2\pi}{m}$$

$$t = r\,sin\frac{a\,2\pi}{m}$$

$$u = h$$

$$P = P_0 + sS + tT + uU$$

3.2.3 Transformation des points sur l'axe cylindrique

Soit P un point situé sur l'axe cylindrique. Quand nous le convertissons du repère cartésien au repère cylindrique, l'angle a n'est pas défini de manière unique. Par convention, on prend a égal à 0 (voir la figure 3.7(a)) :

$$P = (0, 0, 0)$$

Appliquons maintenant la transformation suivante :

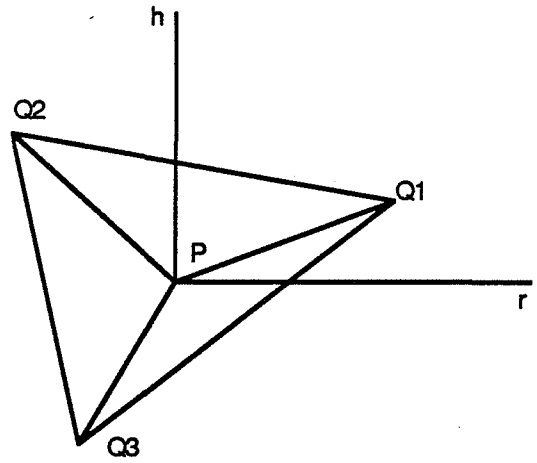
$$r' = r + r_0$$

$$a' = a$$

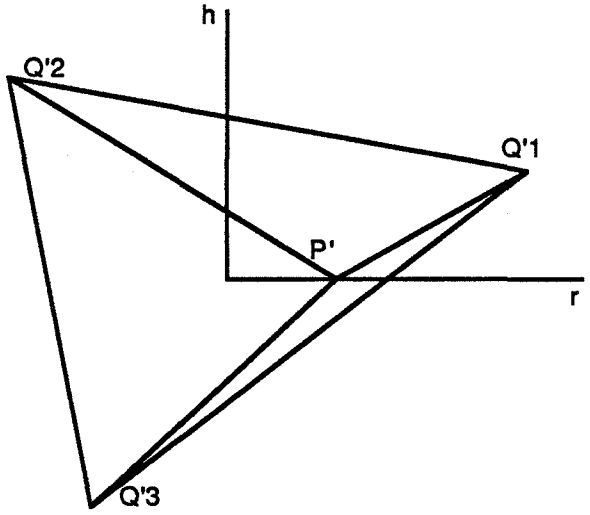
$$h' = h$$

La convention entraîne un résultat inhabituel (voir la figure 3.7(b)) :

$$P' = (r_0, 0, 0)$$



(a)



(b)

Figure 3.7 Problème de la transformation des points sur l'axe cylindrique

Pour résoudre ce problème, nous remplaçons le point sur l'axe cylindrique par les trois points suivants, formant ainsi un triangle nul (voir la figure 3.8(a)) :

$$P_1 = (0, a_1, 0)$$

$$P_2 = (0, a_2, 0)$$

$$P_3 = (0, a_3, 0)$$

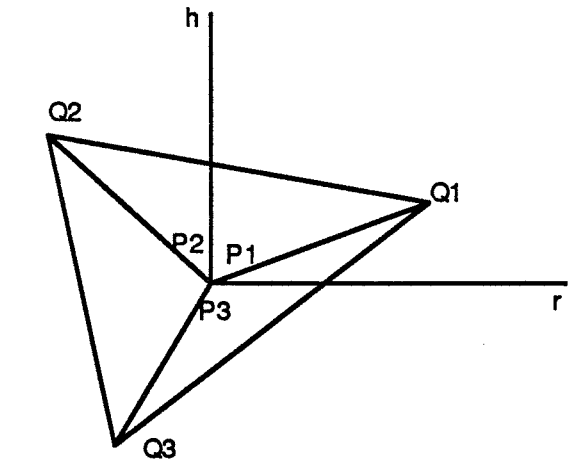
où a_1, a_2, a_3 sont respectivement la seconde coordonnée des points Q_1, Q_2, Q_3.

Ces trois points sont à la même position. Après transformation, nous obtenons trois positions différentes (voir la figure 3.8(b)) :

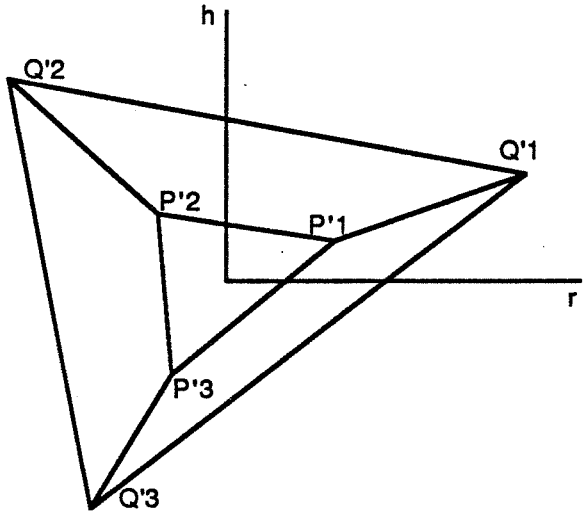
$$P'_1 = (r_0, a_1, 0)$$

$$P'_2 = (r_0, a_2, 0)$$

$$P'_3 = (r_0, a_3, 0)$$



(a)



(b)

Figure 3.8 Solution de la transformation des points sur l'axe cylindrique

3.2.4 Transformation des normales sur l'axe cylindrique

Nous utilisons la matrice Jacobienne pour transformer les normales. Comme on l'a vu, la déformation libre cylindrique est décomposée en quatre étapes : une transformation du repère cartésien du monde au repère local cylindrique, un reparamétrage, une interpolation volumique et une transformation du repère local cylindrique au repère cartésien du monde. Comme les transformations correspondant à ces quatre étapes sont toutes connues explicitement, on peut calculer les matrices Jacobiennes correspondantes.

Rappelons la formule de transformation de normales :

$$N' = det(J)\, J^{-1t} N$$

où J est la matrice Jacobienne.

Proposition :

$$\forall N \neq 0\ ,\ N' \neq 0 \quad \text{si et seulement si} \quad det(J) \neq 0$$

Preuve :

Il est bien connu qu'un système d'équations linéaires homogènes a des solutions non nulles si et seulement si le déterminant de la matrice de coefficients est nul. En considérant la formule de transformation de normales comme système d'équations linéaires ayant N comme vecteur de variables, on a :

$$\exists N \neq 0\ ,\ N' = 0 \quad \text{si et seulement si} \quad det(det(J)\, J^{-1t}) = 0$$

En utilisant l'équation suivante :

$$det(det(J)\, J^{-1t}) = det^3(J)\, \frac{1}{det(J)} = det^2(J)$$

on obtient :

$$\exists N \neq 0\ ,\ N' = 0 \quad \text{si et seulement si} \quad det(J) = 0$$

ce qui est équivalent à :

$$\forall N \neq 0\ ,\ N' \neq 0 \quad \text{si et seulement si} \quad det(J) \neq 0 \qquad \Diamond$$

A partir de la transformation du repère cylindrique au repère cartésien, calculons la matrice Jacobienne correspondante :

$$J = \begin{bmatrix} cos\frac{a\,2\pi}{m} & -r\frac{2\pi}{m}sin\frac{a\,2\pi}{m} & 0 \\ sin\frac{a\,2\pi}{m} & r\frac{2\pi}{m}cos\frac{a\,2\pi}{m} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Le déterminant est :

$$det(J) = r\frac{2\pi}{m}$$

Sur l'axe cylindrique, le déterminant de la matrice Jacobienne est nul. Par conséquent, on ne peut plus utiliser la matrice Jacobienne pour transformer les normales.

D'un autre point de vue, appliquons maintenant la transformation suivante sur un cylindre (voir la figure 3.9) :

$$r' = r$$

$$a' = a$$

$$h' = h + r$$

Une discontinuité C^1 est introduite aux points situés sur l'axe cylindrique.

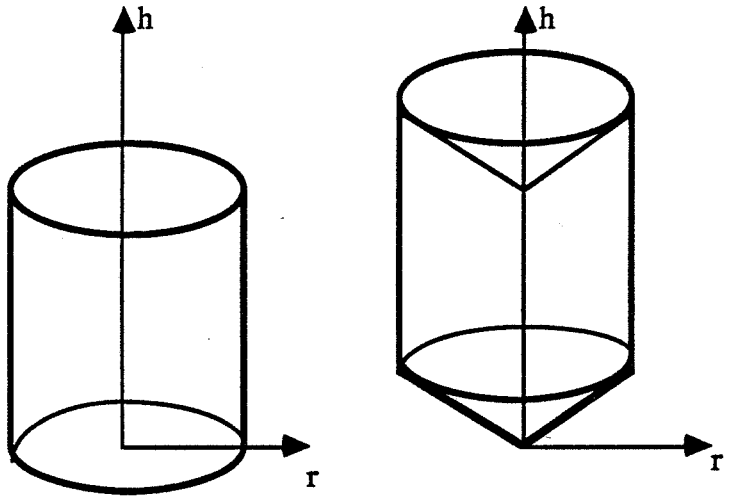
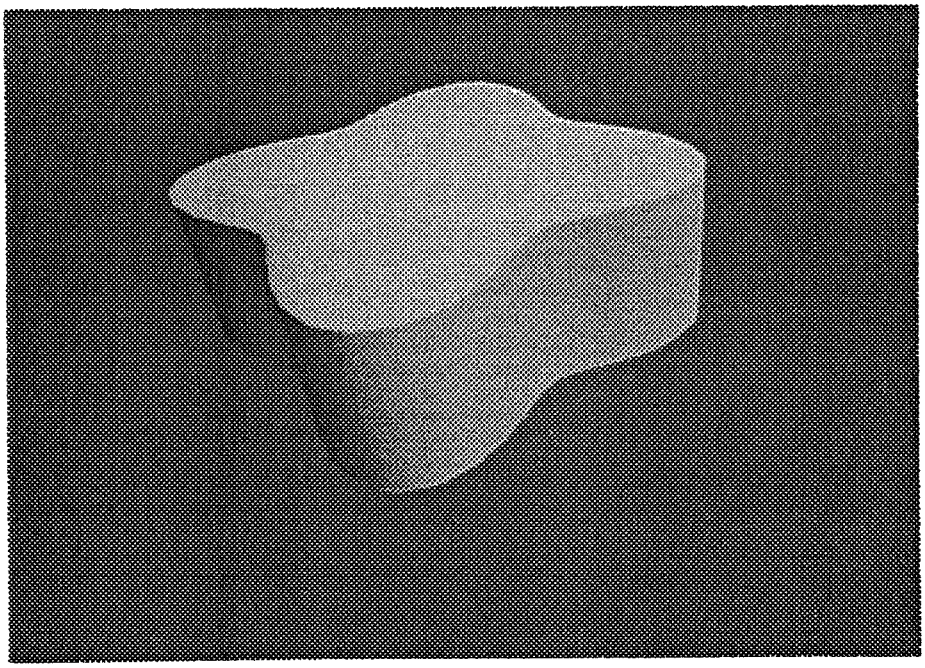


Figure 3.9 Discontinuité C^1 sur l'axe cylindrique

En tenant compte de ces remarques, si un point tombe sur l'axe cylindrique, au lieu d'utiliser la matrice Jacobienne pour transformer la normale en ce point, nous prendrons la moyenne des normales aux triangles incidents comme normale en ce point.

La figure 3.10 montre la déformation d'un cylindre avec un maillage de points de contrôle cylindrique.



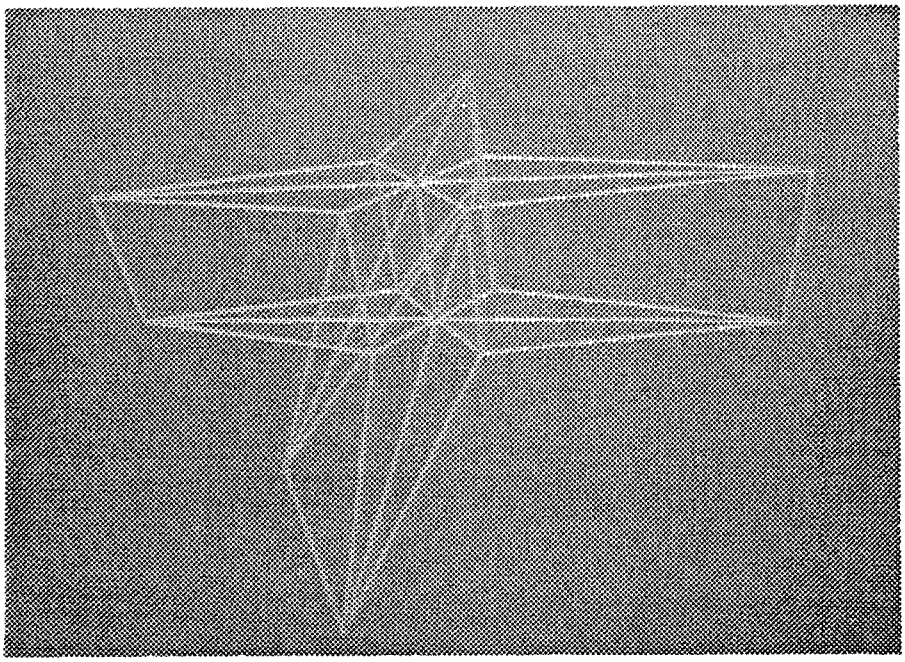


Figure 3.10 Déformation libre cylindrique d'un cylindre

3.3 Déformation libre sphérique

Nous utilisons le repère sphérique défini sur la figure 3.11. Etant données les coordonnées sphériques *(r, a, b)* d'un point *P*, ses coordonnées cartésiennes *(x, y, z)* peuvent être calculées comme suit :

$$x = r \cos\frac{\pi b}{n} \cos\frac{2\pi a}{m}$$

$$y = r \cos\frac{\pi b}{n} \sin\frac{2\pi a}{m}$$

$$z = r \sin\frac{\pi b}{n}$$

où m et n sont des entiers.

La transformation inverse est la suivante :

$$r = (x^2 + y^2 + z^2)^{1/2}$$

$a = atan(m, y, x)$

$b = atan(2n, z, (x^2+y^2)^{1/2})$

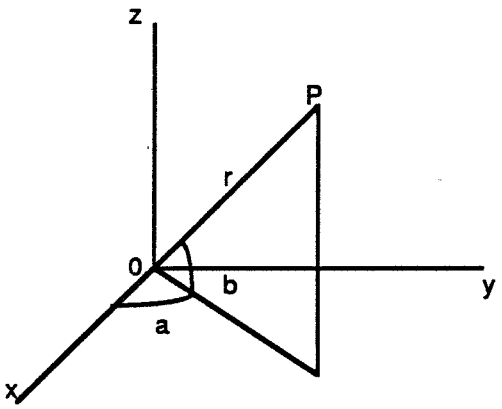


Figure 3.11 Repère sphérique

Avec la déformation libre classique, pour déformer un objet, nous définissons un repère local cartésien avec un point P_0 et trois vecteurs S, T et U. Pour la déformation libre sphérique, nous utilisons comme repère local le repère sphérique qui est associé au repère local cartésien. Nous formons ainsi une région de déformation de forme sphérique (voir la figure 3.12). Chaque point P a des coordonnées (r, a, b) dans le repère local sphérique et (s, t, u) dans le repère local cartésien telles que :

$$s = r \cos\frac{\pi b}{n} \cos\frac{2\pi a}{m}$$

$$t = r \cos\frac{\pi b}{n} \sin\frac{2\pi a}{m}$$

$$u = r \sin\frac{\pi b}{n}$$

$$P = P_0 + sS + tT + uU$$

Nous définissons dans le repère local sphérique un maillage de points de contrôle :

$$P_{ijk} = (r_{ijk}, a_{ijk}, h_{ijk}) = (i, j, k) \qquad 0 \le i \le l \ , \ 0 \le j < m \ , \ 0 \le k \le n$$

où $l+1$, m et $n+1$ sont respectivement le nombre de couches de points de contrôle dans les directions r, a et b.

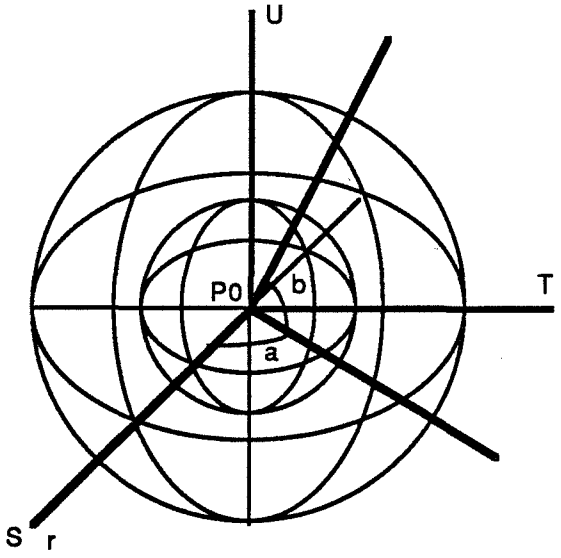


Figure 3.12 Repère local sphérique

Nous déplaçons les points de contrôle dans le repère local sphérique pour spécifier la déformation :

$$P'_{ijk} = F(P_{ijk})$$

Pour un point P sur l'objet à déformer, nous calculons d'abord ses coordonnées locales sphériques :

$$s = \frac{T \wedge U.(P-P_0)}{T \wedge U.S}$$

$$t = \frac{S \wedge U.(P-P_0)}{S \wedge U.T}$$

$$u = \frac{S \wedge T.(P-P_0)}{S \wedge T.U}$$

$$r' = (s^2 + t^2 + u^2)^{1/2}$$

$$a' = atan(m, t, s)$$

$$b' = atan(2n, u, (s^2+t^2)^{1/2})$$

Nous effectuons le reparamétrage pour satisfaire la condition d'invariance :

$$r = f_r(r')$$

$$a = f_a(a')$$

$$b = f_b(b')$$

En utilisant ces nouveaux paramètres, nous effectuons une interpolation volumique de type B-spline.

Comme nous l'avons dit, avec le repère local sphérique, nous formons une région de déformation de forme sphérique. Si on fait varier la coordonnée r en fixant les deux autres coordonnées, on obtient une ligne droite. De même, si on fait varier la coordonnée b en fixant les deux autres coordonnées, on obtient un demi cercle. Par contre, si on fait varier la coordonnée a en fixant les deux autres coordonnées, on obtient un cercle, qui est une courbe fermée. Pour l'interpolation volumique, nous utilisons donc l'interpolation périodique pour la coordonnée a :

$$P''_{ijk} = P'_{i,mod(j,m),k} + \begin{bmatrix} 0 \\ j\text{-}mod(j,m) \\ 0 \end{bmatrix}$$

$$P(r,a,b) = \sum_{i=0}^{l} N_{i,kr}(r) \sum_{j=0}^{m+ka-1} N_{j,ka}(a) \sum_{k=0}^{n} N_{k,kb}(b)\ P''_{ijk}$$

où P'_{ijk} sont les points de contrôle déplacés en coordonnées sphériques,

$mod(j, m)$ donne le reste de la division de j par rapport à m .

Après l'interpolation volumique dans le repère local sphérique, on peut transformer le point obtenu du repère local sphérique dans le repère cartésien du monde :

$$s = r\ cos\frac{\pi b}{n}\ cos\frac{2\pi a}{m}$$

$$t = r \cos\frac{\pi b}{n} \sin\frac{2\pi a}{m}$$

$$u = r \sin\frac{\pi b}{n}$$

$$P = P_0 + sS + tT + uU$$

3.4 Conclusions

Avec la déformation libre cylindrique, on peut réaliser, d'une façon exacte, des déformations qui sont tri-linéaires dans le repère cylindrique. De plus, on obtient une région de déformation de forme cylindrique. Le calcul supplémentaire dû aux transformations entre le repère cylindrique et le repère sphérique peut être optimisé en utilisant la technique de tabulation. Il en est de même pour la déformation libre sphérique.

4 Visualisation des solides déformés

Le système de synthèse d'images développé à Saint Etienne possède plusieurs algorithmes de visualisation, notamment un algorithme du type balayage ligne par ligne et un tracé de rayons. Dans ce chapitre, nous allons nous intéresser au problème de la visualisation des objets obtenus par déformation libre à l'aide des deux algorithmes mentionnés ci-dessus.

Pour visualiser des solides déformés avec le balayage ligne par ligne, nous effectuons une facettisation adaptative des surfaces de solides. Les mécanismes de facettisation adaptative classiques souffrent du problème de l'autoressemblance. Pour résoudre ce problème, nous proposons d'introduire la direction dans la facettisation adaptative.

Nous avons également cherché à visualiser des solides déformés en tracé de rayons. Nous montrons que, avec la déformation libre, le calcul de l'intersection des rayons avec les surfaces des solides peut être simplifié.

Dans le premier paragraphe, nous présenterons la facettisation adaptative directionnelle des solides déformés. Dans le deuxième paragraphe, nous présenterons notre réflexion sur la visualisation en tracé de rayons des solides obtenus à l'aide d'une déformation libre. Nous conclurons dans le troisième paragraphe.

4.1 Facettisation adaptative directionnelle des solides déformés

Nous allons exposer le problème d'autoressemblance lié aux mécanismes de facettisation adaptative, et nous présenterons une solution qui consiste à utiliser la subdivision directionnelle.

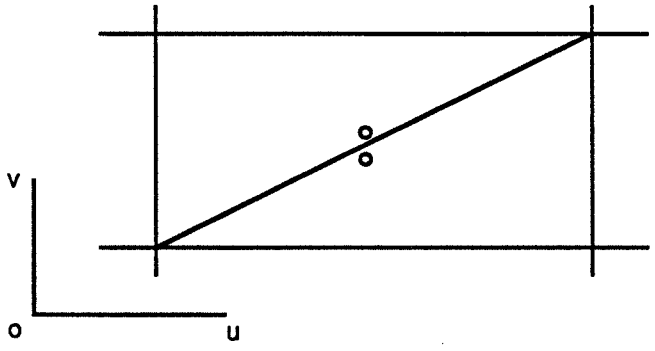
4.1.1 Autoressemblance et subdivision directionnelle

Comme nous l'avons dit dans le chapitre un, la facettisation adaptative a pour but d'optimiser la facettisation de la surface. Mais quelle est la condition d'optimisation ? Une condition raisonnable est la minimisation du nombre de facettes nécessaires pour satisfaire les critères de facettisation.

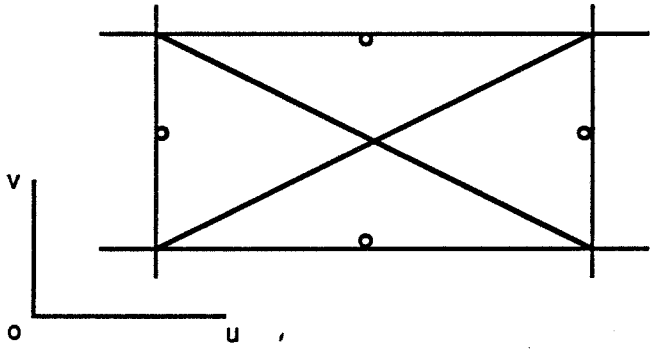
Avec les mécanismes de facettisation adaptative que nous avons présentés, la facettisation est souvent loin d'être optimale. Quand la courbure de la surface présente trop de différences entre chaque direction de facettisation, la facettisation adaptative commence à souffrir du problème d'autoressemblance. Si on subdivise dans une direction de la surface pour satisfaire les critères de facettisation, on subdivisera systématiquement dans l'autre direction, même si dans cette dernière direction les critères sont déjà satisfaits et si la subdivision n'est plus nécessaire. Par conséquent, la forme des facettes après la facettisation ressemble à celle des facettes initiales.

Avec le mécanisme de subdivision de triangles, on subdivise un triangle en divisant son arête "la plus longue". Après division de cette arête, pour favoriser la production des triangles équilatéraux, les deux autres arêtes du triangle subdivisé deviennent systématiquement les arêtes "les plus longues" des deux triangles produits. La facettisation n'est donc pas directionnelle. En fait, les triangles équilatéraux ne s'adaptent pas aux surfaces qui sont plus courbées dans une direction que dans l'autre.

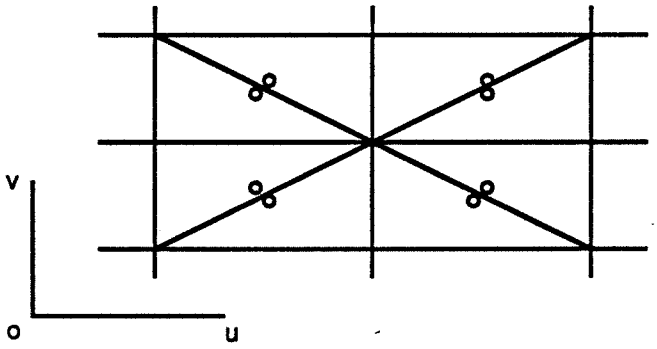
Prenons comme exemple la facettisation d'une surface paramétrique qui est courbée dans la direction u, et plane dans la direction v . Supposons qu'à une étape de facettisation on obtienne deux triangles qui partagent l'arête "la plus longue" (voir la figure 4.1(a)). Comme ils ne satisfont pas le critère de courbure, nous les subdivisons et obtenons ainsi quatre nouveaux triangles qui sont illustrés sur la figure 4.1(b) avec l'arête "la plus longue" pour chaque triangle. Ces quatre triangles ne satisfont pas non plus le critère de courbure. On subdivise chaque triangle en subdivisant son arête "la plus longue" (voir la figure 4.1(c)). Comme nous pouvons le constater, par rapport à la configuration initiale de la figure 4.1(a), nous avons subdivisé dans les deux directions et nous obtenons quatre configurations plus petites mais semblables.



(a)



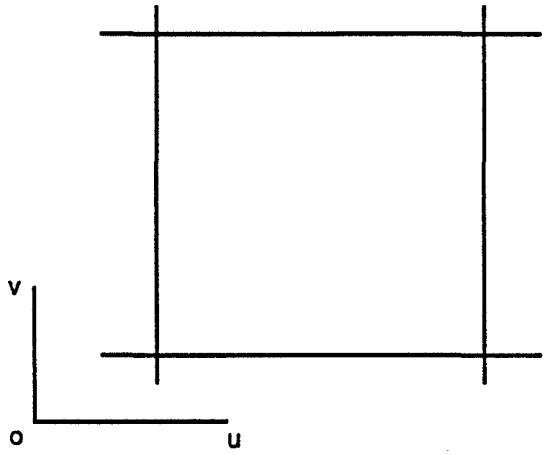
(b)



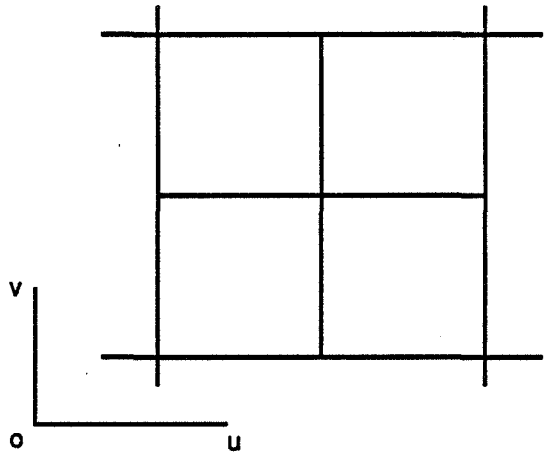
(c)

Figure 4.1 Autoressemblance avec la subdivision de triangles

Avec le mécanisme de subdivision de quadrilatères, si un quadrilatère ne satisfait pas le critère de courbure, on le subdivise dans les deux directions (voir la figure 4.2). La subdivision n'est donc pas directionnelle non plus.



(a)



(b)

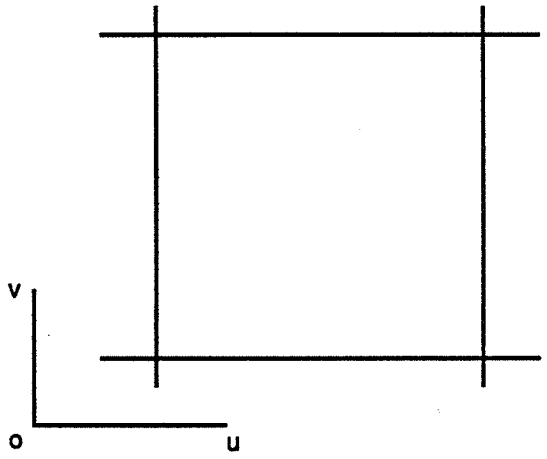
Figure 4.2 Autoressemblance avec la subdivision de quadrilatères

Avec le phénomène d'autoressemblance, la qualité de la facettisation adaptative dépend directement de la qualité de la facettisation initiale. Comme on utilise la facettisation uniforme comme facettisation initiale, elle approxime souvent mal la surface.

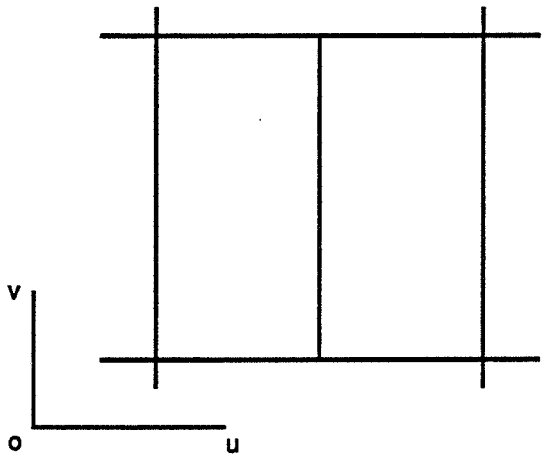
Imaginons une surface paramétrique qui soit très courbée dans la direction u dans une zone, qui soit très courbée dans la direction v dans une autre zone, et qui reste plane dans la zone restante. La facettisation initiale subdivise à un même niveau dans la direction u et dans la direction v. Avec la facettisation adaptative, la subdivision se concentre effectivement dans les deux zones courbées. Mais avec le phénomène d'autoressemblance, on subdivise trop dans la direction v dans la première zone, et on subdivise trop dans la direction u dans la deuxième zone.

Pour surmonter le problème d'autoressemblance, la subdivision de facettes doit être directionnelle. Nous devons subdiviser seulement dans la direction où les critères ne sont pas satisfaits. Si les critères ne sont pas satisfaits dans les deux directions, nous subdiviserons dans la direction où les critères sont les plus mal satisfaits.

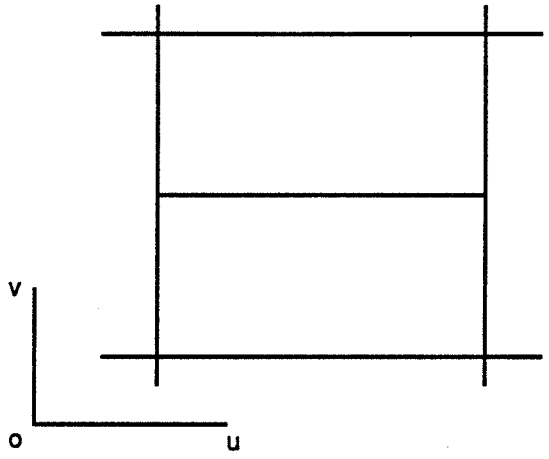
Nous avons choisi d'introduire la direction dans le mécanisme de subdivision de quadrilatères, car cela nous paraît plus simple (voir la figure 4.3). Il est aussi possible d'introduire la direction dans le mécanisme de subdivision de triangles, en utilisant un critère autre que celui de l'arête "la plus longue", pour choisir l'arête à diviser.



(a)



(b)

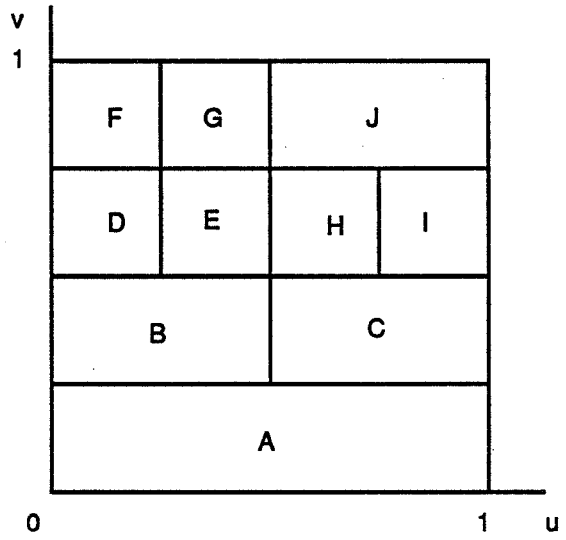


(c)

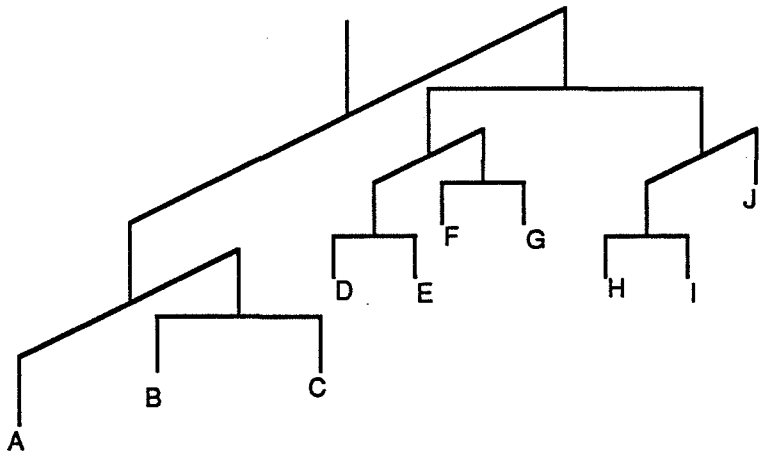
Figure 4.3 Subdivision directionnelle

4.1.2 Arbre binaire directionnel

Avec la subdivision directionnelle, au lieu d'utiliser un arbre quaternaire, nous utilisons un arbre binaire. Cet arbre binaire est directionnel. A chaque noeud, les deux fils représentent un branchement dans la direction u ou dans la direction v, suivant la direction de subdivision de la facette correspondante (voir la figure 4.4).



(a)



(b)

Figure 4.4 Arbre binaire directionnel

Dans le repère paramétrique *(u, v)*, nous utilisons deux paramètres *(direct, order)* pour distinguer les quatre voisins (voir la figure 4.5). Le premier indique la direction (*0* indique la direction *u* et *1* la direction *v*). Le deuxième indique le sens (*0* indique le sens négatif et *1* le sens positif).

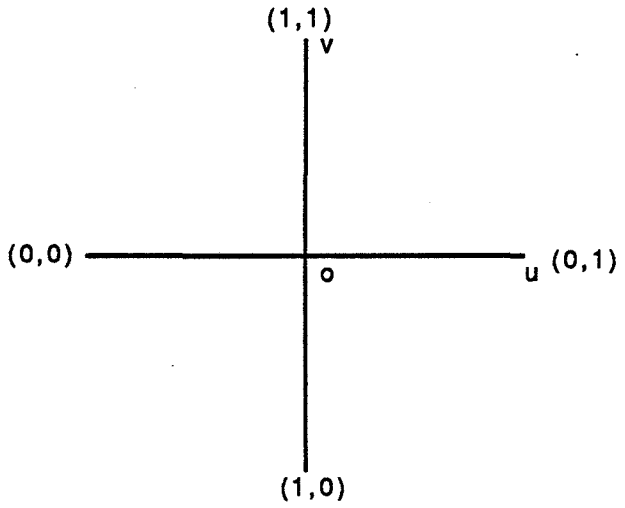


Figure 4.5 Quatre voisins dans le repère paramétrique

Dans le repère paramétrique, nous utilisons deux paramètres *(i, j)* pour distinguer les quatre points d'un quadrilatère (voir la figure 4.6).

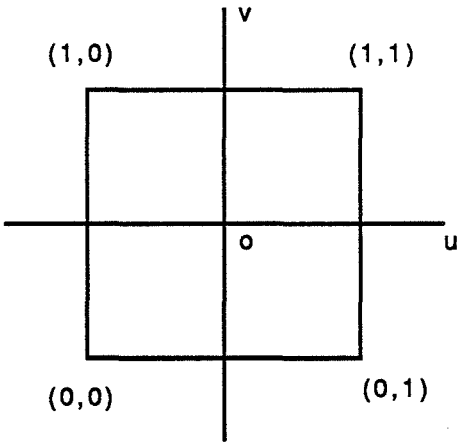


Figure 4.6 Quatre points d'un quadrilatère

Si un voisin *(direct, order)* est donnée, nous utilisons un autre paramètre *order1* pour distinguer les deux points sur l'arête commune (*0* indique le sens négatif et *1* le sens positif). La relation entre les deux représentations est la suivante :

$$i = i\_direct(direct, order, order1) = (1\text{-}direct)\ order1 + direct\ order$$

$$j = j\_direct(direct, order, order1) = direct\ order1 + (1\text{-}direct)\ order$$

Pour chaque surface, nous utilisons la structure de données suivante :

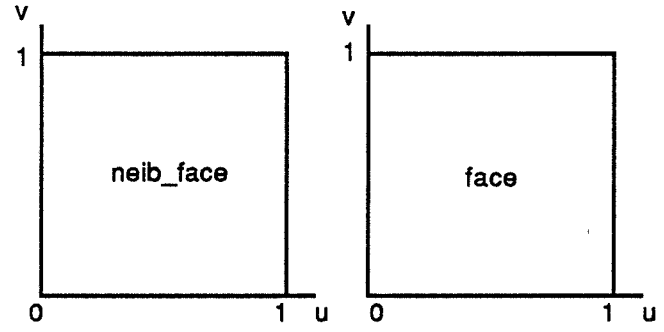
```
typedef struct AdapFace_ {
        int (*func_point)();
        int (*func_norm();
        struct AdapFace_ *neib_face[2][2];
        int neib_flag[2][2];
        AdapTree *tree;
} AdapFace;
```

où *func_point* et *func_norm* sont des fonctions qui calculent respectivement le point et la normale à partir des paramètres ;

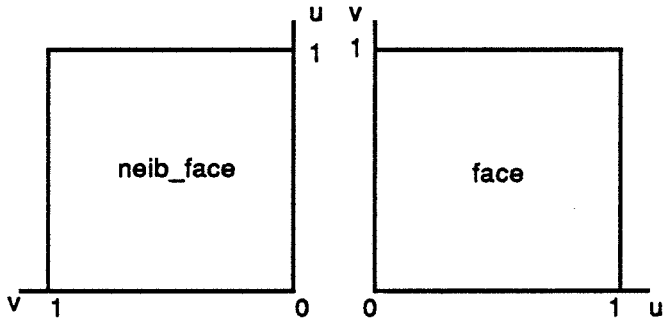
neib_face donne les quatre surfaces voisines ;

tree indique l'arbre correspondant.

Notons que les surfaces voisines de la surface courante peuvent avoir une direction *u* différente de celle-ci (voir la figure 4.7). Nous utilisons *neib_flag* pour le signaler.



(a)



(b)

Figure 4.7 Direction des surfaces voisines

Pour chaque noeud de l'arbre binaire directionnel, nous utilisons la structure de données suivante :

```
typedef struct AdapTree_ {
        int direct;
        struct AdapTree_ *father;
        struct AdapTree_ *son[2];
        AdapFacet *facet;
        int i[2], n[2];
} AdapTree;
```

où *direct* indique la direction de branchement ;
father est le père (*0* indique que ce noeud est la racine) ;
son sont les fils (*son[0]=0* indique que ce noeud est une feuille) ;
facet indique la facette si ce noeud est une feuille.

Nous utilisons les entiers *i[2]* et *n[2]* pour noter le domaine de paramètres :

$$\frac{i[0]}{n[0]} \leq u < \frac{i[0]+1}{n[0]}$$

$$\frac{i[1]}{n[1]} \leq v < \frac{i[1]+1}{n[1]}$$

Nous utilisons des entiers pour éviter des problèmes de précision. On peut avoir ces informations quand on parcourt l'arbre. Nous les notons à chaque noeud pour simplifier le parcours de l'arbre.

Pour chaque facette, nous utilisons la structure de données suivante :

```
typedef struct {
        double *point[4];
        double *norm[4];
        double *norm_point[4];
} AdapFacet;
```

où *norm_point* donne les coordonnées des quatre points de la facette dans le repère normalisé.

Une opération importante sur l'arbre binaire directionnel est de trouver les feuilles voisines d'une feuille. Etant donnée une feuille, pour trouver la facette voisine *(direct, order)* de la facette correspondante, on parcourt d'abord l'arbre en montant pour trouver le noeud voisin *(direct, order)*. Ensuite, à partir du noeud trouvé, on descend dans l'arbre en prenant le fils *1-order* pour un branchement *direct* ou le fils contenant le paramètre *i/n* pour un branchement *1-direct*. La feuille ainsi trouvée donne la facette voisine *(direct, order)* de la facette courante. Si on trouve pas le noeud voisin *(direct, order)* sur cet arbre, on prendra l'arbre correspondant à la surface voisine *(direct, order)* de la surface courante et on le descend à partir de la racine.

Voici le programme pour trouver la facette voisine d'une facette.

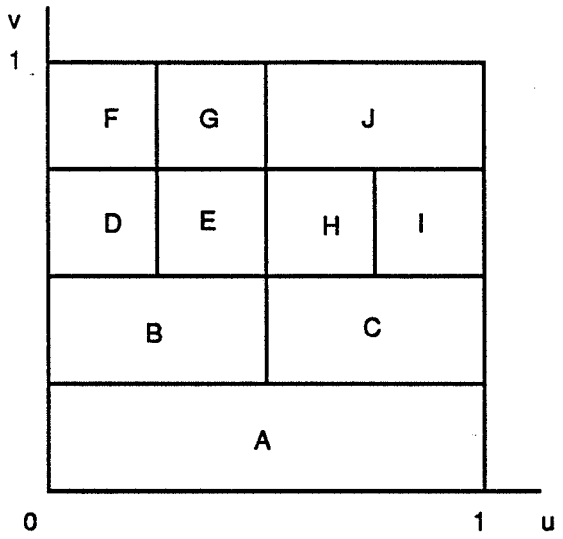
```
AdapTree *adap_neib (face, tree, direct, order, neib_face)
AdapFace *face;
AdapTree *tree;
int direct, order;
```

```
AdapFace **neib_face;
{
        int i, n;

        i = tree->i[1-direct]; n = tree->n[1-direct];

        /* on parcourt l'arbre en montant pour trouver le noeud voisin
(direct, ordre) */
        while(tree->father) { /* tant que le noeud n'est pas la racine */
                if(tree->father->direct == direct){ /* si la direction de
branchement du père est direct */
                        if(tree == tree->father->son[order]){ /* si le noeud
est le fils order de son père */
                                tree = tree->father;
                        }
                        else{
                                tree = tree->father->son[order];
                                break;
                        }
                }
                else{
                        tree = tree->father;
                }
        }

        if(tree->father){ /* si on trouve le noeud voisin sur cet arbre */
                *neib_face = face;
                order = 1 - order;
        }
        else{
                *neib_face = face->neib_face[direct][order];
                tree = (*neib_face)->tree;
                if(face->neib_flag[direct][order]){ /* si les deux surfaces
voisines n'ont pas la même direction u */
                        direct = 1-direct;
                }
                else{
                        order = 1- order;
```

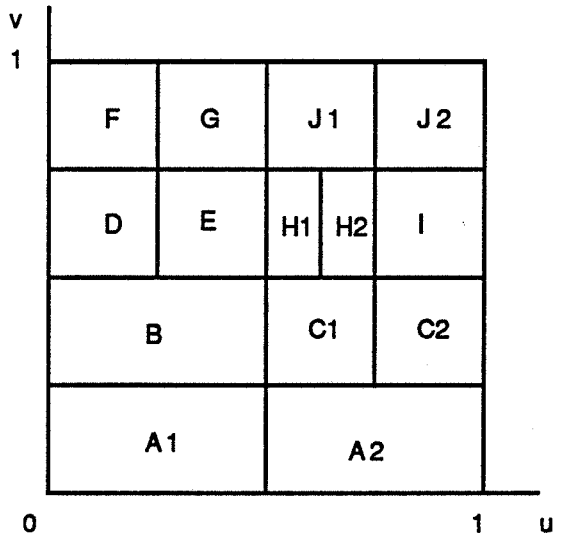
```
            }
        }

        /* on descend dans l'arbre */
        while(tree->son[0]){ /* tant que le noeud n'est une feuille */
            if(tree->direct == direct){ /* si la direction de branchement
est direct */
                tree = tree->son[order];
            }
            else{
                if((tree->i[1-direct]*2+1)*n  >  i*tree->n[1-
direct]*2){ /* si le premier fils contient le paramètre i/n */
                    tree = tree->son[0];
                }
                else{
                    tree = tree->son[1];
                }
            }
        }

        return(tree);
    }
```

4.1.3 Mécanisme de facettisation

Pour respecter la contrainte de cohérence, on impose une condition à la subdivision : la différence entre les niveaux de facettisation dans la direction de subdivision de deux quadrilatères voisins ne doit pas dépasser un. Sur la figure 4.8 par exemple, la subdivision du quadrilatère H dans la direction u entraîne la subdivision des quadrilatères J et C, et la subdivision de C entraîne à son tour celle de A.



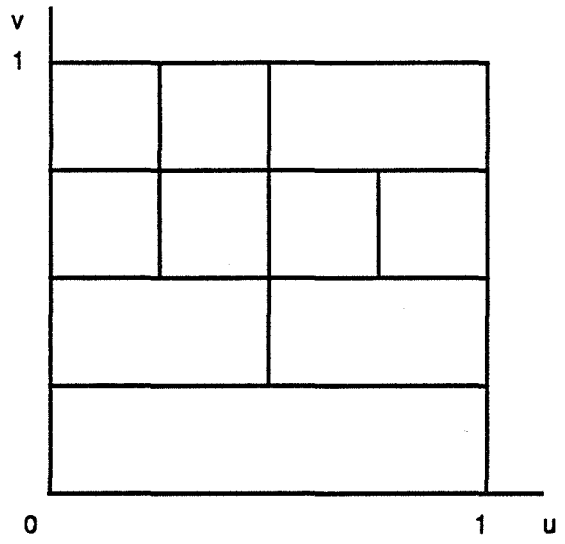
(a)



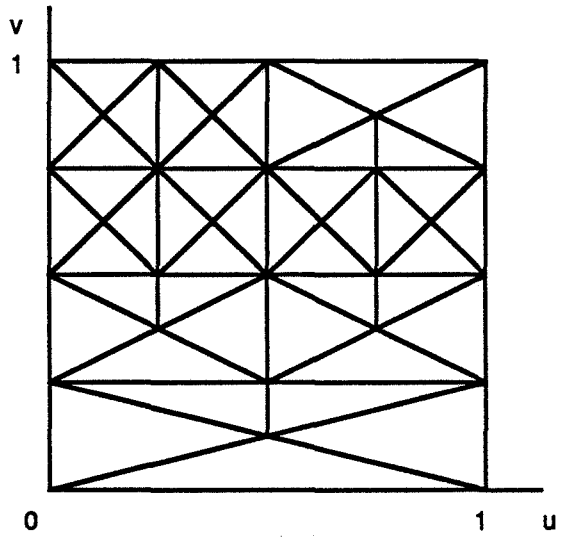
(b)

Figure 4.8 Subdivision de quadrilatères voisins

A la fin de la facettisation adaptative, on triangule tous les quadrilatères (voir la figure 4.9). Chaque quadrilatère est divisé en qutre triangles. Si le voisin d'un triangle est plus grand, on divise ce triangle en deux. La contrainte de cohérence est ainsi satisfaite.



(a)



(b)

Figure 4.9 Triangulation de quadrilatères

Pour démarrer la facettisation adaptative, il faut effectuer une facettisation initiale.

Le niveau de la facettisation initiale peut être le plus bas possible. Nous préférons utiliser une facettisation uniforme à un niveau adéquat pour économiser des calculs lors de la facettisation adaptative.

Le niveau de la facettisation initiale peut être fixé par avance. Nous préférons le laisser spécifier par l'utilisateur, pour garantir le démarrage de la facettisation adaptative.

Le choix de ce niveau détermine à quel moment la facettisation adaptative remplace la facettisation uniforme. S'il est trop grand, la facettisation adaptative risque de se terminer avant de pouvoir refléter les détails géométriques de la surface. S'il est trop petit, la facettisation risque de se dégrader en facettisation uniforme.

En fait, comme on utilise les normales aux sommets pour tester la courbure d'une facette, une facettisation initiale ne sera jamais suffisante pour toutes les surfaces. Le niveau de la facettisation initiale doit être lié à la courbure de la surface. [BARR 87] indique que la condition de Lipschitz peut être utilisée pour déterminer la courbure, donc le niveau de facettisation initiale. Dans la pratique, nous laissons à l'utilisateur le choix du niveau de la facettisation initiale.

Après la facettisation initiale, on construit pour chaque surface du solide un arbre binaire directionnel. Pour pouvoir le faire, les niveaux de la facettisation initiale doivent prendre comme valeurs des puissances de deux.

Pour chaque feuille d'un arbre, on teste la facette correspondante suivant les critères de subdivision. Si les critères ne sont pas satisfaits, on subdivise la facette et on traite d'une façon récursive les deux nouvelles facettes.

Pour subdiviser la facette, on transforme la feuille correspondante en un noeud interne et crée deux nouvelles feuilles comme ses fils. On prend la facette de l'ancienne feuille comme facette du premier fils et crée une nouvelle facette pour le deuxième fils. Pour créer ces deux facettes, on cherche les deux points de division et les normales en ces points sur les deux arêtes de la facette qui sont dans l'autre direction que celle de subdivision.

```
facet_tree (face, tree)
AdapFace *face; /* la surface à laquelle appartient la facette */
AdapTree *tree; /* la feuille correspondant à la facette */
{
        int direct;
        int order;

        direct = test_facet(tree->facet);
```

```
            if(direct != -1){ /* si les critères ne sont pas satisfaits */
                adap_tree_divide(tree, direct); /* créer deux fils pour tree
```
dans la direction *direct* */

```
                for(order=0; order<2; order++){
                    recur_divide(face, tree, 1-direct, order);
                }

                facet_tree(face, tree->son[0]);
                facet_tree(face, tree->son[1]);
            }
        }
```

Pour trouver le point de division et la normale dans une direction donnée de la facette courante, on cherche la facette voisine. Selon son niveau de facettisation par rapport à celui de la facette courante, on obtient le point de division et la normale de façon différente.

```
recur_divide (face, tree, direct, order)
AdapFace *face;
AdapTree *tree;
int direct, order;
{
    AdapFacet *facet, *facet1;
    AdapFace *neib_face;
    AdapTree *neib_tree;
    int neib_direct, neib_order;
    int i1,j1, i2,j2;

    facet = tree->son[0]->facet;
    facet1 = tree->son[1]->facet;

    i1 = i_direct(direct, order, 0); j1 = j_direct(direct, order, 0);
    i2 = i_direct(direct, order, 1); j2 = j_direct(direct, order, 1);

    if(facet->pointoint[i1][j1] == facet->pointoint[i2][j2]){ /* si l'arête
```
est nulle */

```
                facet1->point[i1][j1] = facet->point[i1][j1];
                facet1->norm[i1][j1] = facet->norm[i1][j1];
            }
            else{
                neib_tree = adap_neib(face, tree, direct, order,
&neib_face);
                if(neib_face == face){
                    neib_direct = direct;
                    neib_order = 1-order;
                }
                else{
                    if(face->neib_flag[direct][order]){
                        neib_direct = 1-direct;
                        neib_order = order;
                    }
                    else{
                        neib_direct = direct;
                        neib_order = 1-order;
                    }
                }

                if(neib_tree->n[1-neib_direct] == tree->n[1-direct]*2){
                    divide1(face, tree, direct, order,
                    neib_face, neib_tree, neib_direct, neib_order);
                }
                else if(neib_tree->n[1-neib_direct] == tree->n[1-direct]){
                    divide2(face, tree, direct, order);
                }
                else{
                    divide3(face, tree, direct, order,
                    neib_face, neib_tree, neib_direct, neib_order);
                }
            }
            facet1->point[i2][j2] = facet->point[i2][j2];
            facet1->norm[i2][j2] = facet->norm[i2][j2];
            facet->point[i2][j2] = facet1->point[i1][j1];
            facet->norm[i2][j2] = facet1->norm[i1][j1];
        }
```

Il y a trois cas pour diviser une arête. Si le niveau de facettisation de la facette voisine est plus important (voir la figure 4.10), on trouve le point de division dans la facette voisine. Si les deux facettes sont sur la même surface, on trouve aussi la normale dans la facette voisine. Sinon, on crée la normale pour cette facette.

```
divide1 (face, tree, direct, order,
neib_face, neib_tree, neib_direct, neib_order)
AdapFace *face;
AdapTree *tree;
int direct, order;
AdapFace *neib_face;
AdapTree *neib_tree;
int neib_direct, neib_order;
{
        AdapFacet *facet1, *neib_facet;
        int i1, j1, i3, j3;
        int *i, *n;
        double u[2];

        i1 = i_direct(direct, order, 0);
        j1 = j_direct(direct, order, 0);
        i3 = i_direct(neib_direct, neib_order, 1);
        j3 = j_direct(neib_direct, neib_order, 1);

        facet1 = tree->son[1]->facet;
        neib_facet = neib_tree->facet;

        facet1->point[i1][j1] = neib_facet->point[i3][j3];

        if(neib_face == face){
                facet1->norm[i1][j1] = neib_facet->norm[i3][j3];
        }
        else{
                i = tree->i; n = tree->n;
                u[1-direct] = (i[1-direct]*2+1.)/(n[1-direct]*2);
                u[direct] = (i[direct] + order*1.)/n[direct];
```

```
            facet1->norm[i1][j1] = (*face->func_norm)(u);
        }
    }
```

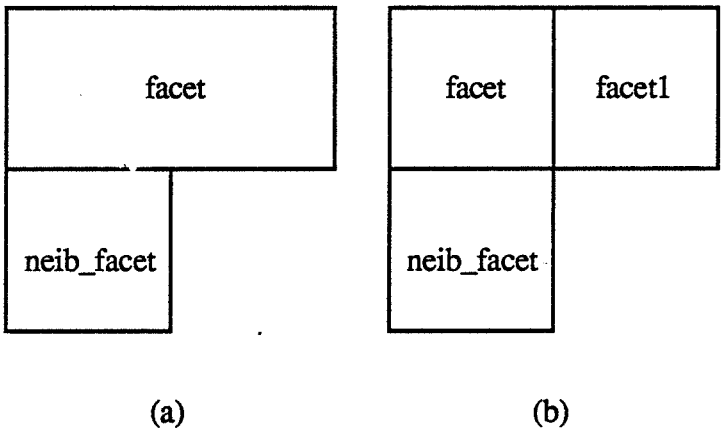


Figure 4.10 Cas d'une facette voisine avec un niveau de facettisation plus important

Si les deux facettes ont le même niveau de facettisation (voir la figure 4.11), on crée un point et une normale.

```
divide2 (face, tree, direct, order)
AdapFace *face;
AdapTree *tree;
int direct, order;
{
        AdapFacet *facet1;
        int i1, j1;
        int *i, *n;
        double u[2];

        i1 = i_direct(direct, order, 0); j1 = j_direct(direct, order, 0);

        facet1 = tree->son[1]->facet;

        i = tree->i; n = tree->n;
        u[1-direct] = (i[1-direct]*2+1.)/(n[1-direct]*2);
        u[direct] = (i[direct] + order*1.)/n[direct];

        facet1->point[i1][j1] = (*face->func_point)(u);
        facet1->norm[i1][j1] = (*face->func_norm)(u);
```

}

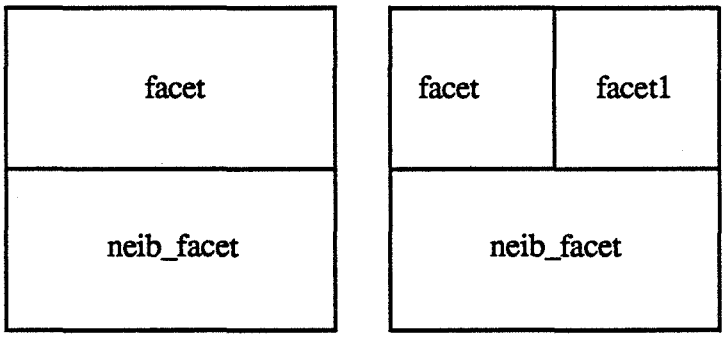


Figure 4.11 Cas d'une facette voisine avec le même niveau de facettisation

Si le niveau de facettisation de la facette voisine est plus faible (voir la figure 4.12), il faut subdiviser la facette voisine pour satisfaire la contrainte imposée. On transforme la feuille correspondant à la facette voisine en un noeud interne et on crée deux nouvelles feuilles comme ses fils. Pour la facette voisine, on retrouve le premier cas dans la direction vers la facette initiale. Dans la direction inverse, on cherche le point de division et la normale d'une façon récursive. Après subdivision de la facette voisine, on retrouve le deuxième cas pour la facette initiale.

```
divide3 (face, tree, direct, order,
neib_face, neib_tree, neib_direct, neib_order)
AdapFace *face;
AdapTree *tree;
int direct, order;
AdapFace *neib_face;
AdapTree *neib_tree;
int neib_direct, neib_order;
{
        adap_tree_divide(neib_tree, 1-neib_direct);
        divide1(neib_face, neib_tree, neib_direct, neib_order, face, tree,
direct, order);
        recur_divide(neib_face, neib_tree, neib_direct, 1-neib_order);
        divide2(face, tree, direct, order);
}
```

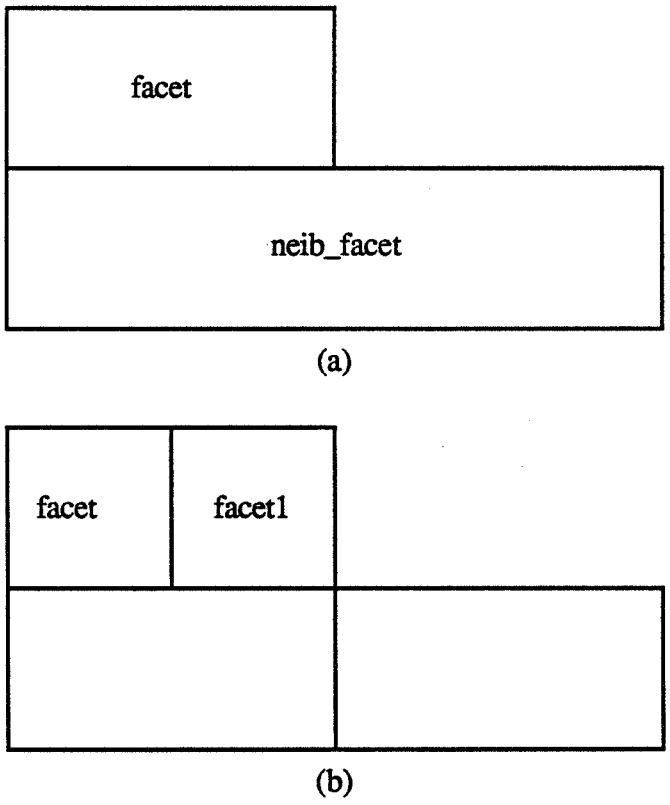


Figure 4.12 Cas d'une facette voisine avec un niveau de facettisation plus faible

4.1.4 Critères de facettisation

Le mécanisme de subdivision développé dans le paragraphe précédent repose sur des critères de facettisation. Ces critères peuvent être variés. Pour notre part, nous avons implanté deux critères de facettisation : la courbure de la surface et la taille de la facette après projection sur l'écran.

Pour tester la courbure d'une facette, on utilise les normales aux quatre sommets de la facette. On calcule les angles entre les deux paires de normales dans la direction u et on choisit l'angle le plus grand comme courbure de la facette dans cette direction. De même, on calcule les angles entre les deux paires de normales dans la direction v et on choisit l'angle le plus grand comme courbure de la facette dans cette direction.

Le choix du critère de courbure *da* dépend de deux demandes contradictoires : la précision d'approximation et l'efficacité de calcul. Par expérience, nous avons choisi *da* égal à $\frac{\pi}{36}$.

Si l'angle dans la direction u ne satisfait pas le critère et si l'angle dans la direction v le satisfait, on subdivise dans la direction u. Si l'angle dans la direction v ne satisfait pas le critère et si l'angle dans la direction u le satisfait, on subdivise dans la direction v. Si aucun des deux angles ne satisfait le critère, on subdivise dans la direction où l'angle est plus grand.

```
test_curvature (facet)
AdapFacet *facet;
{
        int direct, order;
        double w[2][2]; /* les poids des quatre arêtes */

        for(direct=0; direct<2; direct++){
                for(order=0; order<2; order++){
                        i1 = i_direct(0, order, direct);
                        j1 = j_direct(0, order, direct);
                        i2 = i_direct(1, order, direct);
                        j2 = j_direct(1, order, direct);

                        w[direct][order] = 1.
                        -facet->norm[i1][j1][0]*facet->norm[i2][j2][0]
                        -facet->norm[i1][j1][1]*facet->norm[i2][j2][1]
                        -facet->norm[i1][j1][2]*facet->norm[i2][j2][2];
                }

                /* mettre le plus grand dans w[direct][0] */
                if(w[direct][0] < w[direct][1]){
                        w[direct][0] = w[direct][1];
                }
        }

        if(w[0][0] >= w[1][0]){
                direct = 0;
        }
        else{
                direct = 1;
        }
```

```
        if(w[direct][0] < cos(da)){
            return(direct);
        }
        else{
            return(-1);
        }
}
```

Pour tester la taille d'une facette après projection sur l'écran, on utilise les coordonnées des quatre sommets dans le repère normalisé. On calcule les longueurs des deux arêtes dans la direction u et on choisit la longueur la plus grande comme taille de la facette dans la direction u. On fait de même dans la direction v.

Le choix du critère de taille *de* est facile. Supposons que la définition de l'écran soit de $m * n$. Pour que la taille de la facette soit inférieure à un pixel, il faut choisir *de* égal à $\frac{2}{max(m,n)}$.

```
test_size (facet)
AdapFacet *facet;
{
        int direct, order;
        double w[2][2];
        double *p1, *p2;

        for(direct=0; direct<2; direct++){
            for(order=0; order<2; order++){
                i1 = i_direct(0, order, direct);
                j1 = j_direct(0, order, direct);
                i2 = i_direct(1, order, direct);
                j2 = j_direct(1, order, direct);

                p1 = facet->norm_point[i1][j1];
                p2 = facet->norm_point[i2][j2];

                w[direct][order]
                = (p1[0]-p2[0])*(p1[0]-p2[0])
                + (p1[1]-p2[1])*(p1[1]-p2[1]);
            }
```

```
            if(w[direct][0] < w[direct][1]){
                w[direct][0] = w[direct][1];
            }
        }

        if(w[0][0] >= w[1][0]){
            direct = 0;
        }
        else{
            direct = 1;
        }

        if(w[direct][0] > de*de){
            return(direct);
        }
        else{
            return(-1);
        }
    }
```

En combinant ces deux critères, on obtient le test suivant :

```
    test_facet (facet)
    AdapFacet *facet;
    {
        int direct;

        if( (direct = test_size(facet)) != -1
        || (direct = test_curvature(facet)) != -1){
            return(direct);
        else{
            return(-1)
        }
    }
```

La figure 4.13 montre la facettisation uniforme et la facettisation adaptative d'un cube courbé. La facettisation adaptative subdivise plus à l'endroit de courbure. Notons

aussi que la facettisation adaptative, qui est directionnelle, subdivise plus dans la direction de courbure.

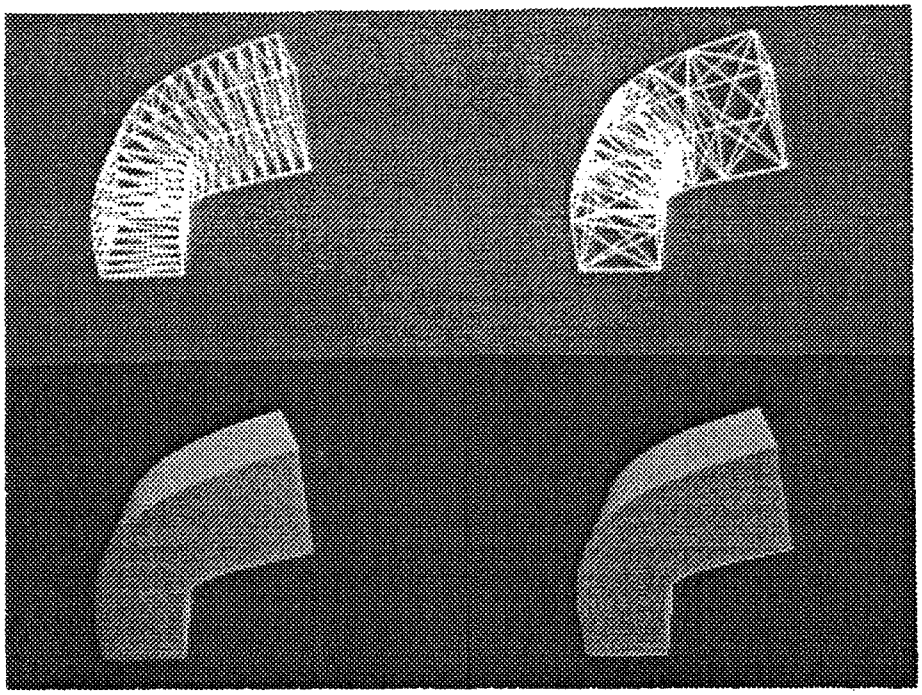


Figure 4.13 Facettisation uniforme et facettisation adaptative

4.2 Visualisation en tracé de rayons des solides déformés librement

La déformation libre de type Bézier ou de type B-spline uniforme est une transformation polynomiale. En se limitant à l'interpolation cubique, on peut l'écrire sous la forme suivante :

$$x' = \sum_{i=0}^{3} \sum_{j=0}^{3} \sum_{k=0}^{3} a_{ijk}\, x^i\, y^j\, z^k$$

$$y' = \sum_{i=0}^{3} \sum_{j=0}^{3} \sum_{k=0}^{3} b_{ijk}\, x^i\, y^j\, z^k$$

$$z' = \sum_{i=0}^{3} \sum_{j=0}^{3} \sum_{k=0}^{3} c_{ijk}\, x^i\, y^j\, z^k$$

Représentons maintenant cette transformation en coordonnées homogènes :

$$\frac{x'}{w'} = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} a_{ijk}\,\frac{x^i}{w^i}\frac{y^j}{w^j}\frac{z^k}{w^k}$$

$$\frac{y'}{w'} = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} b_{ijk}\,\frac{x^i}{w^i}\frac{y^j}{w^j}\frac{z^k}{w^k}$$

$$\frac{z'}{w'} = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} c_{ijk}\,\frac{x^i}{w^i}\frac{y^j}{w^j}\frac{z^k}{w^k}$$

$$x' = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} a_{ijk}\,x^i\,y^j\,z^k\,w^{9-i-j-k}$$

$$y' = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} b_{ijk}\,x^i\,y^j\,z^k\,w^{9-i-j-k}$$

$$z' = \sum_{i=0}^{3}\sum_{j=0}^{3}\sum_{k=0}^{3} c_{ijk}\,x^i\,y^j\,z^k\,w^{9-i-j-k}$$

$$w' = w^9$$

En coordonnées homogènes, la déformation libre est aussi une transformation polynomiale.

Les surfaces des solides à déformer peuvent en général être représentées par une surface polynomiale en coordonnées homogènes. Par exemple, un demi-cercle :

$$r = 1$$

$$0 \le a < \pi$$

peut être représenté par les équations suivantes :

$$x = 0.5 - t$$

$$y = t\,(1-t)$$

$$w = t^2 - t + 0.5$$

où le paramètre t appartient à $[0, 1)$.

En résumé, en coordonnées homogènes, la déformation libre d'une surface polynomiale reste une surface polynomiale.

Dans le cas de la déformation libre locale, la région de déformation ne contient qu'une partie de l'objet à déformer. On peut utiliser l'enveloppe convexe du maillage de points de contrôle pour déterminer si le rayon intersecte la partie déformée ou la partie restante de l'objet.

Si on utilise la déformation libre de type B-spline non-uniforme, le reparamétrage demande la résolution d'une équation algébrique, dont la solution n'est pas une fonction polynomiale. Par conséquent, on obtient un système d'équations non-polynomiales.

Si on utilise la déformation libre cylindrique ou sphérique, comme les transformations des coordonnées entre le repère cartésien et le repère cylindrique ou sphérique ne sont pas polynomiales, on obtient aussi un système d'équations non-polynomiales. En fait, le rayon peut être représenté par des fonctions linéaires dans le repère cartésien, et ne peut qu'être représenté par des fonctions non-polynomiales dans le repère cylindrique ou sphérique.

4.3 Conclusions

Nous utilisons deux méthodes pour visualiser des solides déformés. La première consiste à facettiser les solides déformés et à utiliser les algorithmes de visualisation classiques comme le balayage ligne par ligne pour les visualiser. La deuxième méthode consiste à les visualiser directement en tracé de rayons.

Pour optimiser la facettisation, il faut utiliser la facettisation adaptative. Les mécanismes classiques de facettisation adaptative souffrent du problème d'autoressemblance. Pour surmonter ce problème, nous proposons la subdivision directionnelle.

La facettisation adaptative directionnelle que nous avons présentée est valable pour toutes les déformations. Pour la déformation libre, la déformation est liée au déplacement

des points de contrôle. Nous envisageons d'utiliser le déplacement des points de contrôle pour déterminer la subdivision dans le mécanisme de facettisation adaptative.

Les surfaces de solides à déformer peuvent en général être représentées comme des surfaces polynomiales en coordonnées homogènes. La déformation libre peut aussi être représentée comme une transformation polynomiale en coordonnées homogènes. Le résultat de la déformation est donc constitué des surfaces polynomiales. On peut utiliser les algorithmes classiques pour calculer l'intersection de rayons avec des surfaces.

5 Utilisation des déformations libres

Dans le premier paragraphe, nous allons présenter l'utilisation de la déformation libre dans la modélisation par arbre de construction, comme extension des transformations affines.

La déformation libre est spécifiée par le déplacement des points de contrôle. Quand il faut beaucoup de points de contrôle pour obtenir une déformation un peu complexe, leur spécification devient une tâche lourde pour l'utilisateur. Une interface propre est donc importante pour l'utilisation de la déformation libre en synthèse d'images.

Dans le deuxième paragraphe, nous présenterons une interface syntaxique qui décrit la scène sous forme textuelle. Nous décrirons l'utilisation des déformations globales pour faciliter le déplacement des points de contrôle.

Une interface graphique, qui utilise un curseur 3D pour spécifier le déplacement des points de contrôle, sera présentée dans le troisième paragraphe.

Dans le quatrième paragraphe, nous présenterons l'animation à l'aide des déformations libres.

Nous conclurons dans le cinquième paragraphe.

5.1 Déformation libre et arbre de construction

Dans l'arbre de construction classique, les transformations utilisées sont affines, ce qui permet de les représenter à l'aide de matrices homogènes. Cela donne une grande efficacité quand on parcourt l'arbre de construction, car on peut combiner les transformations par produit de matrices. L'inconvénient est que les transformations affines ne changent pas la forme des objets, à l'exception de certains cas particuliers (transformations perspectives par exemple). Cela limite largement l'espace de modélisation de l'arbre de construction. Pour dépasser cette limitation, nous allons

introduire les transformations non-affines dans l'arbre de construction. Nous avons choisi la déformation libre pour étendre l'arbre de construction, car elle donne un grand espace de modélisation et garde en même temps une certaine efficacité de calcul (calculs polynomiaux). En revanche, avec cette extension, on ne peut plus combiner les transformations quand on parcourt l'arbre.

5.1.1 Structures de données

Dans un arbre de construction classique, toutes les transformations sont affines et peuvent être représentées par une matrice. Quand on parcourt l'arbre de construction, on peut combiner les transformations successives en calculant le produit des matrices correspondantes.

Avec l'utilisation de la déformation libre, on ne peut plus combiner les transformations successives. Il faut donc maintenir une liste de transformations que l'on va appliquer à un objet. Nous utilisons une liste chaînée pour représente la liste de transformations. Voici la structure de données correspondante :

```
typedef struct Transform_ {
        int type;                   /* 0=affine, 1=non-affine */
        struct FfdGrid *grid;       /* une déformation libre */
        double mat[16];             /* une transformation affine */
        struct Transform_ *next;    /* la transformation suivante */
} Transform;
```

5.1.2 Facettisation

Si on utilise le balayage ligne par ligne pour visualiser une scène décrite par un arbre de construction, toutes les primitives sur les feuilles de l'arbre seront facettisées.

Si une primitive ne subit que des transformations affines, on peut utiliser sa facettisation standard, qui peut être, par exemple, une facettisation uniforme à un niveau donné dont les facettes ne sont pas forcément des triangles. Si la primitive subit des déformations, la facettisation dépendra de ces déformations. Il faut donc tester les facettes suivant des critères de facettisation.

Pour détecter les primitives à déformer, on utilise une variable booléenne *deform* initialisée à *faux*. Quand on parcourt l'arbre de construction, si on rencontre un noeud qui porte une déformation, on met *deform* à *vrai*. De cette manière, quand on arrive à une feuille, on sait si la primitive subira des déformations.

Après la facettisation, une primitive est représentée par un ensemble de facettes. Nous utilisons la structure de donnés suivante :

```
typedef struct {
        int type;          /* le type du solide facettisé */
        int np;            /* le nombre de points */
        double *point;     /* l'ensemble de points */
        int ne;            /* le nombre d'arêtes */
        int *edge;         /* l'ensemble d'arêtes */
        int nf;            /* le nombre de facettes */
        int *pface;        /* le début d'une facette dans face et
vnorm */
        int *topol;        /* le numéro de la surface à laquelle
appartient une facette */
        int *snorm;        /* le numéro de la normale surfacique */
        int sf;            /* la taille de face et vnorm */
        int *face;         /* le nombre et les numéros des arêtes d'une
facette */
        int *vnorm;        /* le nombre et les numéros des normales
nodales */
        int nn;            /* le nombre de normales */
        double *norm;      /* l'ensemble de normales */
} Polyhedra;
```

Une arête est définie par les numéros des deux points extrêmes, au lieu de leurs coordonnées, car un point peut être partagé par plusieurs arêtes.

L'ensemble de points est défini par un tableau de *4*np* réels, au lieu d'une liste chaînée, de façon à ce que la transformation affine de l'ensemble de points puisse être effectuée à l'aide d'une seule multiplication matricielle.

Si on utilise la facettisation adaptative, l'ensemble des points, des arêtes, des normales et des facettes deviendront dynamiques. Les listes chaînées s'adapteront mieux

à cette situation. Nous gardons l'utilisation de tableaux pour privilégier la situation sans déformation.

Le tableau *pface* est utilisé, car les facettes peuvent contenir des nombres d'arêtes différents. Si la surface est à déformer, nous n'utiliserons que des facettes triangulaires. Nous gardons le tableau *pface* pour utiliser la même structure de données qu'avec une surface non déformée.

Nous utilisons le type du solide facettisé et le numéro de surface d'une facette pour déterminer le type de cette facette. Par exemple, on peut déterminer si une facette est plane ou gauche pour choisir le type de rendu pour cette facette.

Les normales nodales sont stockées pour chaque facette, car deux facettes voisines peuvent avoir deux normales différentes en un même point.

Toutes les normales sont regroupées dans un tableau et ce sont les numéros des normales qui sont utilisés pour noter la normale surfacique et les normales nodales pour chaque facette, car les normales, comme les points, peuvent être partagées.

Sur l'arbre de construction, toutes les primitives sont facettisées. Au lieu d'évaluer directement les opérations booléennes sur les polyèdres ainsi obtenus, nous laissons ce travail à l'algorithme de visualisation. Pour ce faire, nous avons choisi l'algorithme Atherton [ATHE 83]. C'est un algorithme d'élimination des parties cachées de type balayage ligne par ligne, qui résond en même temps les opérations booléennes.

5.1.3 Boîte englobante

Les boîtes englobantes sont utilisées pour optimiser la visualisation. Sur l'arbre de construction, on attache une boîte englobante à chaque noeud. Si le noeud n'est pas partageable, la boîte peut être la vraie boîte dans le repère du monde de l'objet correspondant à ce noeud. Dans le cas contraire, on ne peut qu'attacher au noeud la boîte de l'objet correspondant à ce noeud mais sans tenir compte des transformations apparaissant au dessus de ce noeud. Il faut donc appliquer toutes les transformations au dessus de ce noeud sur cette boîte pour trouver la boîte dans le repère du monde.

Pour une transformation affine, la boîte englobante de l'objet final est la transformée de celle de l'objet initial. Pour la déformation libre, ce n'est plus le cas. On

utilise alors son enveloppe convexe pour calculer la boîte englobante de l'objet déformé. Plus précisément, la boîte englobante de l'objet final est l'union de celle de l'objet original avec l'enveloppe convexe du maillage de points de contrôle.

Pour éviter le calcul explicite de l'union de deux polyèdres, on cherche une boîte englobante dont les bords soient parallèles aux axes du repère du monde (sans tenir compte des transformations au dessus de ce noeud). Cela est traduit par la procédure suivante :

```
deform_box (grid, box)
FfdGrid *grid;          /* déformation libre */
double box[8][3];       /* boîte englobante de l'objet à déformer */
{
        double p[2][3];         /* p[0] est le point minimum, p[1] est le
point maximum */
        int i, j, k;
        int d;

        /* chercher le point minimum et le point maximum de box */
        for(d=0; d<3; d++){
                p[0][d] = p[1][d] = box[0][d];
        }
        for(i=1; i<8; i++){
                for(d=0; d<3; d++){
                        if (p[0][d] > box[i][d]) p[0][d] = box[i][d];
                        if (p[1][d] < box[i][d]) p[1][d] = box[i][d];
                }
        }

        /* chercher le point minimum et le point maximum du maillage de
points de contrôle */
        for(i=0; i<=grid->n[0]; i++)
        for(j=0; j<=grid->n[1]; j++)
        for(k=0; k<=grid->n[2]; k++){
                for(d=0; d<3; d++){
                        if (p[0][d] > grid->p[i][j][k][d]){
                                p[0][d] = grid->p[i][j][k][d];
                        }
                        if (p[1][d] < grid->p[i][j][k][d]){
```

```
                        p[1][d] = grid->p[i][j][k][d];
                    }
                }
            }
            /* construire la nouvelle boîte englobante */
            for(i=0; i<2; i++)for(j=0; j<2; j++)for(k=0; k<2; k++){
                box[i*4+j*2+k][0] = p[k][0];
                box[i*4+j*2+k][1] = p[j][1];
                box[i*4+j*2+k][2] = p[i][2];
            }
        }
```

Si la déformation libre est cylindrique, nous remplaçons chaque point de contrôle par quatre points comme nous l'avons décrit dans le chapitre trois. Le reste demeure inchangé.

La lampe sur la figure 5.1 illustre l'utilisation des déformations sur l'arbre de construction. L'enveloppe est la différence de deux cylindres, qui subit un effilement. La barre mobile est un cylindre, qui subit une courbure. La base est l'union de deux cylindres et la boule est une sphère. L'union de ces quatre éléments donne la lampe.

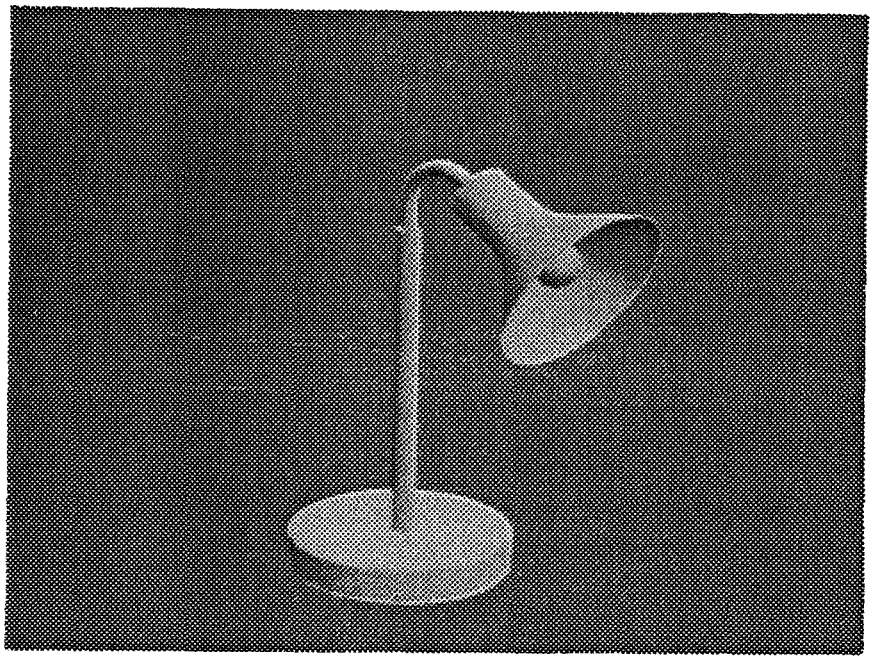


Figure 5.1 Une lampe

5.2 Interface syntaxique pour la déformation libre

Dans son sens le plus général, une interface réalise la communication entre deux systèmes. Une interface pour l'utilisateur d'un système de synthèse d'images a pour but de l'aider à décrire la scène à visualiser. L'utilisateur donne des commandes de conception, et le système répond par une image.

La caractéristique principale d'une telle interface est l'interactivité. C'est le temps de réponse du système à une commande de l'utilisateur.

Une autre caractéristique de l'interface est la facilité de modélisation pour l'utilisateur. Par exemple, au lieu de déplacer un nombre important de sommets d'un polyèdre, l'utilisateur préfère appeler une fonction de déformation pour le déformer.

Avec une interface syntaxique, la scène à visualiser est spécifiée sous forme textuelle. Aucun matériel spécial n'est nécessaire pour employer une interface syntaxique.

Une interface syntaxique peut se présenter sous deux formes différentes. La première décrit la scène en spécifiant les données des objets dans la scène et les paramètres des opérations appliquées sur les objets : c'est l'interprétation. La deuxième décrit la scène en appelant des fonctions prédéfinies : c'est la compilation.

5.2.1 Interprétation

Nous avons implanté la déformation libre dans le système de synthèse d'images ILLUMINES [BEIG 89]. Rappelons rapidement ses fonctionnalités (voir la figure 5.2).

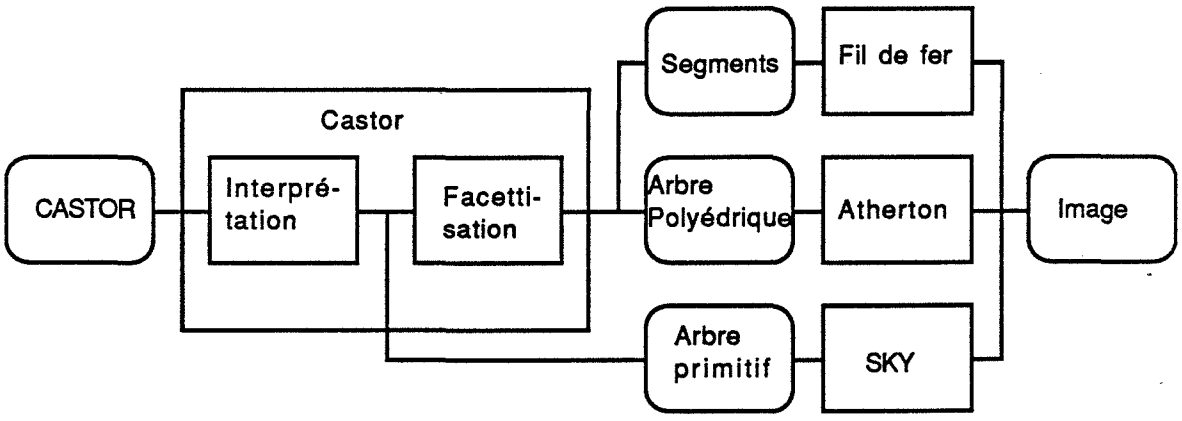


Figure 5.2 Système de synthèse d'images ILLUMINES

Au coeur du système on trouve le module Castor. C'est un modeleur de type arbre de construction. L'entrée de Castor est une description de la scène sous forme syntaxique CASTOR. Le module Castor interprète cette description syntaxique et, sauf pour la visualisation en tracé de rayons, effectue la facettisation des primitives et fournit en sortie une autre description de la scène.

Plusieurs méthodes de visualisation sont disponibles dans le système. Après le module Castor, on trouve le module Atherton qui utilise l'algorithme de même nom pour la visualisation d'un arbre de polyèdres. L'algorithme d'Atherton est un algorithme de visualisation du type balayage ligne par ligne qui effectue en même temps les opérations booléennes. Un autre module de visualisation SKY utilise l'arbre primitif fourni par Castor pour le visualiser en tracé de rayons.

Nous avons implanté la déformation libre dans le module Castor. Un des rôles principaux de Castor est d'analyser la description de la scène fournie sous forme syntaxique CASTOR par l'utilisateur. Nous ne donnons pas ici la description complète de ce langage CASTOR qu'on peut trouver dans [BEIG 88]. Notons seulement que pour faciliter l'analyse syntaxique, certains caractères spéciaux sont utilisés pour indiquer les mots clés du langage, ce qui est illustré par les exemples suivants.

Pour les paramètres de projection, on utilise le symbole % :

%e(x, y, z) /* oeil */

%a(x, y, z) /* mire */

Pour les primitives, c'est le symbole *!* :

!cu(r, v, b, a) /* cube */

!cyl(r, v, b, a) /* cylindre */

!sp(r, v, b, a) /* sphère */

Pour les transformations, @ a été choisi :

@a(a, b, c) /* affinité */

@t(a, b, c) /* translation */

Pour les opérations booléennes, c'est *$* :

$u(obj1, obj2) /* union */

$i(obj1, obj2) /* intersection */

$d(obj1, obj2) /* différence */

La déformation libre est une transformation. On utilise donc la syntaxe suivante pour la décrire :

@d
(l, m, n)
(P_0, S, T, U)
(P_{000}, P_{001}, ..., P_{ijk}, ..., P_{lmn})

où @ indique une transformation,

d indique une déformation libre,

l+1, *m+1* et *n+1* sont le nombre de couches de points de contrôle dans chaque direction du repère local,

P_0, *S*, *T*, *U* définissent le repère local par rapport au repère du monde,

P_{ijk} sont les positions des points de contrôle.

5.2.2 Compilation

Pour utiliser les facilités du langage C, nous avons développé un module Cicas qui fournit une interface syntaxique sous forme d'un ensemble de fonctions à appeler. Le module Cicas prend comme entrée une description de la scène sous forme d'un programme en langage C.

Nous avons défini les fonctions suivantes pour utiliser la déformation libre.

La fonction *ffd_grid()* est utilisée pour définir un repère local cartésien et le maillage de points de contrôle associé :

*FfdGrid *ffd_grid (l, m, n, P0, S, T, U)*
int l, m, n; /* $l+1$, $m+1$ et $n+1$ sont le nombre de couches de points de contrôle dans chaque direction du repère local */
*double *P_0, *S, *T, *U;* /* P_0, S, T, U définissent le repère local par rapport au repère du monde */

De même, pour définir des repères locaux cylindrique et sphérique, on utilise les fonctions *ffd_grid_cy()* et *ffd_grid_sp()*.

Pour déplacer les points de contrôle, on a défini la fonction de translation :

transl(dp, point, norm);
*double *dp;*
*double *point;*
*double *norm;*

où *dp* est le vecteur de translation,
point est le point à transformer,
norm est la normale au point à transformer en utilisant la matrice Jacobienne en ce point (*norm* peut être nul pour indiquer qu'il n'y a pas de normale à transformer).

Pour appliquer la déformation libre, on a la fonction suivante :

ffd (grid, point, norm)
*FfdGrid *grid;*
*double *point;*
*double *norm;*

Pour utiliser la déformation libre, on peut appeler les fonctions précédentes pour définir le repère local et le maillage de points de contrôle, déplacer les points de contrôle, et appliquer la déformation sur un objet. En langage C, cela peut être réalisé comme suit :

```
FfdGrid *grid;

grid = ffd_grid(l, m, n, P0, S, T, U);

transl(dp0, grid->p[0][0][0], 0);
...
transl(dp1, grid->p[l][m][n], 0);

ffd(grid, point, norm);
```

Par rapport à la forme d'interprétation, la forme de compilation possède des avantages : comme elle est implantée dans un langage existant, on peut utiliser tous les mécanismes de contrôle, les structures de données et les facilités de programmation de ce langage. L'inconvénient est qu'il faut recompiler le programme chaque fois qu'on change la description de la scène.

5.2.3 Utilisation des déformations globales

On peut utiliser l'interface syntaxique assez aisément pour décrire un arbre de construction classique. Avec l'introduction de la déformation libre, l'interface syntaxique devient difficile à utiliser. En effet, pour définir un maillage de points de contrôle de *4*4*10*, il faut spécifier *160* points dans l'espace!

Une solution consiste à utiliser des déformations globales pour déplacer les points de contrôle. Nous allons définir un ensemble de déformations globales qui sont les plus utiles dans la pratique. Pour réaliser le déplacement, nous appliquons une ou plusieurs déformations prédéfinies sur les points de contrôle. Avec l'utilisation des déformations globales, l'utilisateur n'a souvent que quelques paramètres de déformation à spécifier au lieu d'avoir à déplacer un nombre de points de contrôle beaucoup plus important.

Les avantages de cette méthode indirecte, par rapport à l'application directe de la déformation sur l'objet, sont les suivants :

- L'homogénéité de programmation : toutes les déformations sont réalisées par l'intermédiaire de la déformation libre.

- L'efficacité de calcul : la transformation polynomiale est certainement la déformation la moins coûteuse à évaluer.

L'inconvénient est aussi évident. A travers la déformation libre, la méthode indirecte ne donne qu'un résultat de déformation approximatif par rapport à la méthode directe.

Comme nous l'avons montré dans le chapitre trois, si la déformation peut être exprimée dans le repère local comme une transformation tri-linéaire, la déformation libre réalisera la même transformation que la déformation directe. A partir de cette remarque, nous appliquons en général les déformations aux points de contrôle dans le repère local. Les points de contrôle sont donc en coordonnées locales. Quand on a terminé le déplacement des points de contrôle, on les utilise pour effectuer l'interpolation volumique, et on transforme le point obtenu dans le repère du monde.

Quatre déformations globales ont été définies dans le chapitre un : la courbure, l'effilement, la torsion et le tirage. Nous définissons pour chacune d'elles une fonction qui demande certains paramètres comme arguments. Pour l'effilement linéaire par exemple, nous avons la fonction suivante :

```
taper(z1, z2, k, point, norm)
double z1, z2, k;       /* paramètres de l'effilement linéaire */
double *point;
double *norm;
```

Pour appliquer cette déformation sur le point de contrôle P_{ijk} du maillage *grid*, nous l'appelons comme suit :

```
taper(u1, u2, k, grid->p[i][j][k], 0);
```

Pour appliquer la déformation à tous les points de contrôle (voir la figure 5.3), nous utilisons une boucle :

```
for(i=0; i<=grid->n[0]; i++)
for(j=0; j<=grid->n[1]; j++)
for(k=0; k<=grid->n[2]; k++){
        taper(u1, u2, k, grid->p[i][j][k], 0);
}
```

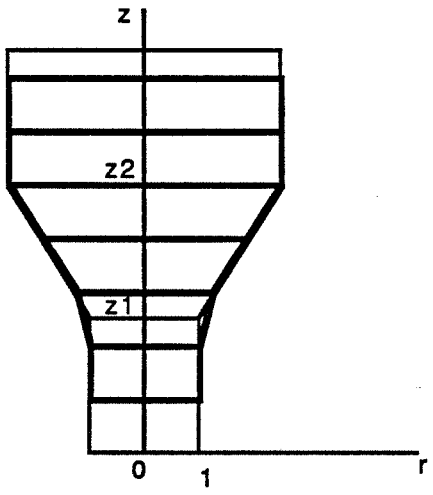


Figure 5.3 Effilement sur un maillage de points de contrôle

Rappelons que l'effilement linéaire crée une discontinuité C^1 dans les plans des deux extrémités. Quand on le réalise avec la déformation libre, on peut supprimer la discontinuité en utilisant une région de déformation un peu plus grande. En fait, sur la figure 5.2, seuls les points de contrôle entre z_1 et z_2 ont subi l'effilement. Mais avec la déformation libre, la partie d'objet inférieure à z_1 se trouve dans la région de déformation et peut être déformée.

Quand on veut appliquer plusieurs déformations successives à un objet, on peut le réaliser de deux manières différentes. La première méthode consiste à définir un seul maillage de points de contrôle qui englobe les régions de déformation de toutes les déformations. On applique toutes les déformations sur le maillage de points de contrôle global et on déforme l'objet avec ce maillage. La deuxième méthode consiste à définir un maillage de points de contrôle pour chaque déformation. Chaque déformation est appliquée sur le maillage correspondant, et toutes les déformations libres définies sont appliquées successivement sur l'objet. En général, on utilise la première méthode qui est plus simple. Si les régions de déformation sont éloignées l'une de l'autre (on veut, par exemple, appliquer deux courbures aux deux bouts d'un cylindre), la deuxième est plus propre. Si on utilise la première méthode, les points situés au milieu du cylindre, qui sont en dehors des régions de déformation des deux courbures, sont dans le maillage global et passent donc par la procédure de déformation libre, ce qui va augmenter les temps de calcul.

La figure 5.4 illustre la courbure d'un cube, l'effilement d'un cylindre et la torsion d'un cube. Toutes ces déformations sont réalisées à l'aide de la déformation libre.

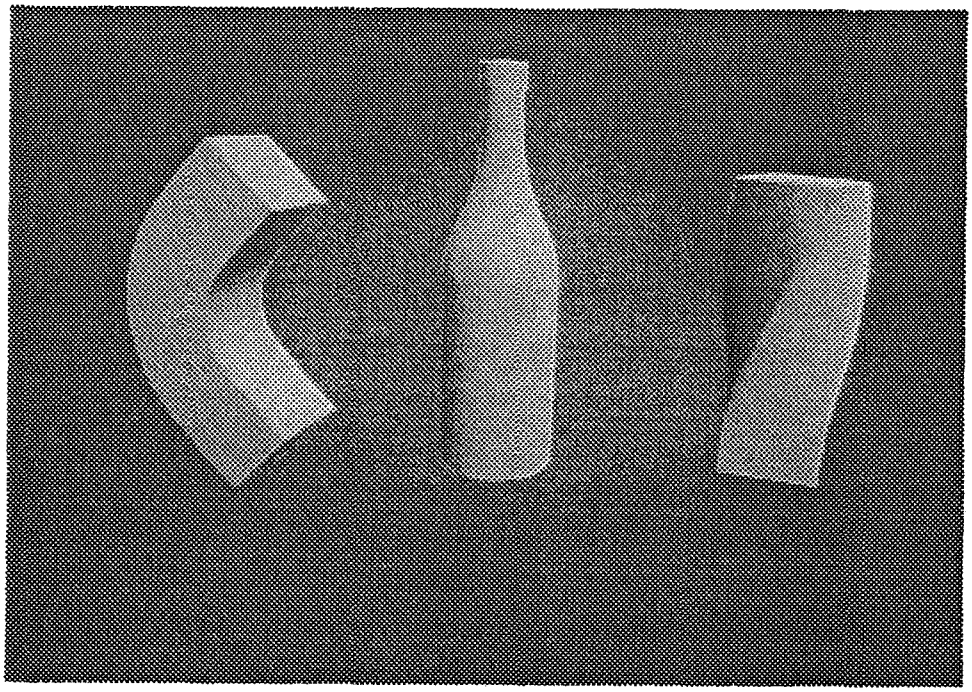


Figure 5.4 Déformations globales réalisées avec la déformation libre

5.3 Interface graphique de la déformation libre

Quand l'utilisateur se sert d'une interface graphique pour concevoir une scène, il a l'impression d'être en train de la concevoir sur le champ. Il a, à sa disposition, des objets comme des primitives et des sources de lumière, des outils comme des transformations et des opérations booléennes, et il peut observer la scène avec une caméra virtuelle. Avec l'utilisation de plus en plus générale des stations de travail graphiques, les systèmes de synthèse d'images ont tendance à bénéficier de plus en plus les interfaces graphiques.

Sur une station de travail graphique ordinaire, on a, à sa disposition, la souris pour l'entrée et l'écran graphique pour la sortie. Une action de l'utilisateur correspond à une procédure, comme la transformation des objets, l'observation de la scène. On peut la réaliser avec le déplacement d'une souris. Comme on a seulement une souris, avant de déplacer la souris, on peut choisir dans un menu en cliquant le bouton sur la souris pour distinguer différents types d'actions.

Pour certaines actions, comme la création des objets ou les opérations booléennes, on ne s'intéresse pas à la procédure mais seulement au résultat de l'action. On peut utiliser un bouton de la souris pour la réaliser. De même, on choisit dans un menu en cliquant le bouton pour distinguer les actions.

Il est intéressant d'utiliser une interface graphique pour définir la déformation libre. L'utilisateur spécifie le déplacement des points de contrôle en manipulant un appareil interactif, et il voit le résultat de la déformation en même temps. Deux problèmes se posent pour réaliser une telle interface. Premièrement, il faut trouver une solution propre pour spécifier le déplacement des points dans l'espace 3D avec un appareil interactif qui se déplace en général sur un plan. Deuxièmement, il faut être capable de calculer un résultat de déformation rapidement, quitte à ce qu'il ne soit pas très réaliste, pour satisfaire la contrainte de réponse instantanée.

5.3.1 Spécification de la position 3D

[ALLA 89] spécifie un déplacement 3D en deux étapes. On choisit une direction significative pour le point à déplacer, qui peut être la normale en ce point quand celui-ci est le sommet d'un maillage de polygones. Un plan est défini qui est perpendiculaire à cette direction et passe par le point. On déplace d'abord le plan le long de la direction (un déplacement 1D), puis on effectue un déplacement 2D dans le plan, pour désigner la nouvelle position du point.

[OVER 90] décrit un curseur 3D pour spécifier la position d'un point dans l'espace 3D. Ce curseur 3D est implanté en logiciel en utilisant un appareil interactif 2D comme une souris. En considérant la différence entre la dimension de ce curseur "soft" et celle de l'appareil interactif, il faut imposer des contraintes pour spécifier une position avec ce curseur.

A partir des idées sur le curseur 3D de [OVER 90], nous avons défini notre propre curseur 3D, qui est plus général. Nous utiliserons notre curseur 3D pour transformer le maillage de points de contrôle, l'objet, et le curseur lui-même.

Le curseur 3D est représenté par sa position L et une direction principale D . Quatre contraintes différentes sont proposées qui correspondent aux quatre modes d'utilisation de notre curseur 3D.

5.3.1.1 Ligne de projection

La position d'un curseur 2D sur l'écran détermine une ligne qui passe par E et qui a la direction A. Si on utilise la projection perspective, E sera la position de l'oeil et A sera la direction de la ligne qui passe par l'oeil et la position du curseur 2D. Si la projection orthogonale est utilisée, E sera la position du curseur 2D et A sera la direction de projection.

En général, dans un système de synthèse d'images, on connait la matrice qui transforme des points du repère du monde dans le repère normalisé. A partir de sa matrice inverse N, on peut déterminer E et A .

Soient (x', y', z') les coordonnées normalisées du curseur 2D. x' et y' sont connues à partir du curseur 2D. Les coordonnées dans le repère du monde (x, y, z) sont données par les équations suivantes, en fonction de z' :

$$xw = N_{00}\,x' + N_{01}\,y' + N_{02}\,z' + N_{03}$$

$$yw = N_{10}\,x' + N_{11}\,y' + N_{12}\,z' + N_{13}$$

$$zw = N_{20}\,x' + N_{21}\,y' + N_{22}\,z' + N_{23}$$

$$w = N_{30}\,x' + N_{31}\,y' + N_{32}\,z' + N_{33}$$

$$x = \frac{N_{02}\,z' + N_{00}x' + N_{01}y' + N_{03}}{N_{32}\,z' + N_{30}x' + N_{31}y' + N_{33}}$$

$$y = \frac{N_{12}\,z' + N_{10}x' + N_{11}y' + N_{13}}{N_{32}\,z' + N_{30}x' + N_{31}y' + N_{33}}$$

$$z = \frac{N_{22}\,z' + N_{20}x' + N_{21}y' + N_{23}}{N_{32}\,z' + N_{30}x' + N_{31}y' + N_{33}}$$

où N_{ij} sont des éléments de la matrice N.

En prenant $z' = 0$, on obtient la position du point E :

$$E_x = \frac{N_{00}x' + N_{01}y' + N_{03}}{N_{30}x' + N_{31}y' + N_{33}}$$

$$E_y = \frac{N_{10}x'+N_{11}y'+N_{13}}{N_{30}x'+N_{31}y'+N_{33}}$$

$$E_z = \frac{N_{20}x'+N_{21}y'+N_{23}}{N_{30}x'+N_{31}y'+N_{33}}$$

Pour une projection orthogonale, N_{32} est nul. On a donc :

$$A = (N_{02}, N_{12}, N_{22})$$

qui est à normaliser.

Pour une projection perspective, N_{32} n'est pas nul. On a donc :

$$x = \frac{N_{02}}{N_{32}} + \frac{N_{32}(N_{00}x'+N_{01}y'+N_{03}) - N_{02}(N_{30}x'+N_{31}y'+N_{33})}{N_{32}N_{32}\,z' + N_{32}(N_{30}x'+N_{31}y'+N_{33})}$$

$$y = \frac{N_{12}}{N_{32}} + \frac{N_{32}(N_{10}x'+N_{11}y'+N_{13}) - N_{12}(N_{30}x'+N_{31}y'+N_{33})}{N_{32}N_{32}\,z' + N_{32}(N_{30}x'+N_{31}y'+N_{33})}$$

$$z = \frac{N_{22}}{N_{32}} + \frac{N_{32}(N_{20}x'+N_{21}y'+N_{23}) - N_{22}(N_{30}x'+N_{31}y'+N_{33})}{N_{32}N_{32}\,z' + N_{32}(N_{30}x'+N_{31}y'+N_{33})}$$

En définissant le paramètre suivant qui augmente quand z' augmente :

$$t = -\frac{1}{N_{32}N_{32}\,z' + N_{32}(N_{30}x'+N_{31}y'+N_{33})}$$

on obtient donc la direction :

$$A_x = N_{02}(N_{30}x'+N_{31}y'+N_{33}) - N_{32}(N_{00}x'+N_{01}y'+N_{03})$$

$$A_y = N_{12}(N_{30}x'+N_{31}y'+N_{33}) - N_{32}(N_{10}x'+N_{11}y'+N_{13})$$

$$A_z = N_{22}(N_{30}x'+N_{31}y'+N_{33}) - N_{32}(N_{20}x'+N_{21}y'+N_{23})$$

qui est aussi à normaliser.

5.3.1.2 Utilisation pour les translations

Comme le curseur 3D de [OVER 90], notre curseur 3D peut être utilisé pour la translation d'un point. Deux modes de translations sont définis.

Dans le mode translation 2D, le point est contraint de se déplacer dans le plan passant par L et orthogonale à la direction D. La dimension libre du curseur 3D est la même que celle de l'appareil interactif. Comme [OVER 90], la position du point P est définie comme l'intersection de la ligne EA avec le plan LD (voir la figure 5.5). La ligne EA est définie par :

$$P = E + tA$$

Le plan LD a pour équation :

$$(P\text{-}L).D = 0$$

A partir de ces équations, l'intersection P est donc :

$$P = E + \frac{(L\text{-}E).D}{A.D} A$$

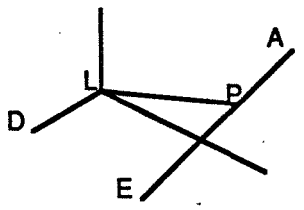


Figure 5.5 Translation 2D

Dans le mode translation 1D, le point est contraint de se déplacer sur la ligne passant par L et ayant la direction D. L'appareil interactif a une dimension libre de plus que le curseur 3D. Pour passer du curseur 2D à la translation 1D, il faut imposer une contrainte. Au lieu d'imposer une contrainte sur l'écran comme dans [OVER 90] (voir la figure 5.6(a)), nous imposons une contrainte dans l'espace telle que la ligne qui passe par Q et P soit perpendiculaire à la ligne EA et à la ligne LD (voir la figure 5.6(b)). Cette nouvelle contrainte est indépendante de la position de l'écran, tout en garantissant que si la ligne EA intersecte la ligne LD, P sera l'intersection des deux lignes. La ligne EA est définie par :

$$Q = E + sA$$

La ligne LD a pour équation :

$$P = L + tD$$

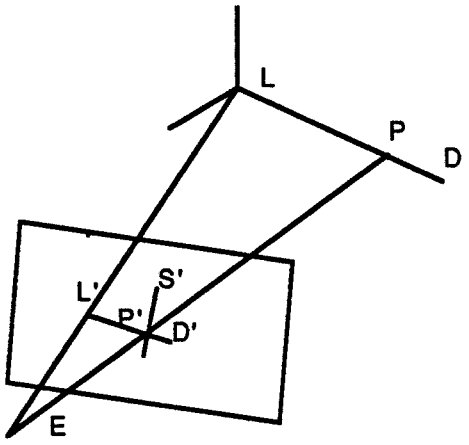
La contrainte impose :

$$(P-Q).A = 0$$

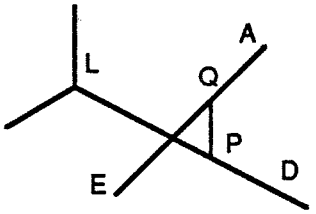
$$(P-Q).D = 0$$

A partir de ces équations, on peut calculer la position du point P :

$$P = L + \frac{((L-E).A)(D.A)-(L-E).D}{1-(D.A)^2} D$$



(a)



(b)

Figure 5.6 Translation 1D

5.3.1.3 Utilisation pour les rotations

Notre curseur 3D peut aussi être utilisé pour la rotation d'un point. Comme la translation, la rotation a aussi deux modes de fonctionnement.

Dans le mode rotation 2D, le point est contraint de se déplacer sur la sphère qui est centrée en L et qui passe par la position R du point au moment de la saisie. La position du point P est déterminée par l'intersection de la ligne EA et de la sphère LR (voir la figure 5.7). La ligne EA a pour équation :

$$P = E + tA$$

La sphère est définie par :

$$(P-L).(P-L) = (R-L).(R-L)$$

A partir de ces deux équations, on obtient l'équation suivante :

$$t^2 + 2(E-L).A\ t + (E-L).(E-L) - (R-L).(R-L) = 0$$

dont la solution donne la position du point P :

$$P = E + (-(E-L).A-(((E-L).A)^2-((E-L).(E-L)-(R-L).(R-L)))^{1/2})\ A$$

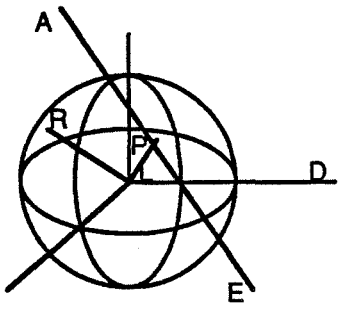


Figure 5.7 Rotation 2D

Dans le mode rotation 1D, le point est contraint de se déplacer sur le cercle qui est perpendiculaire à la direction D et qui passe par la position R du point au moment de la saisie. Comme dans le mode translation 1D, pour passer du curseur 2D à la rotation 1D, il faut aussi imposer une contrainte. Si M est l'intersection de la ligne LD et du plan RD,

et si Q est l'intersection de la ligne EA et du plan MD, nous avons choisi la contrainte suivante : P est sur la ligne MQ (voir la figure 5.8). Avec cette contrainte, quand la ligne EA intersecte le cercle, P sera l'intersection.

Calculons d'abord le point M. La ligne LD est définie par :

$$M = L + tD$$

Le plan qui passe par R et qui est perpendiculaire à la direction D est :

$$(M-R).D = 0$$

A partir de ces équations, on peut trouver le point M :

$$M = L + ((R-L).D)\, D$$

Calculons ensuite le point Q. La ligne EA a pour équation :

$$Q = E + tA$$

Le plan qui passe par M et qui est perpendiculaire à la direction D est :

$$(Q-M).D = 0$$

A partir de ces équations, on peut trouver le point Q :

$$Q = E + ((M-E).D)\, A$$

Calculons enfin le point P. La ligne MQ a pour équation :

$$P = M + t\,(Q-M)$$

La contrainte impose :

$$(P-M).(P-M) = (R-M).(R-M)$$

On obtient donc la nouvelle position du point :

$$P = M + ((R-M).(R-M) / (Q-M).(Q-M))^{1/2} (Q-M)$$

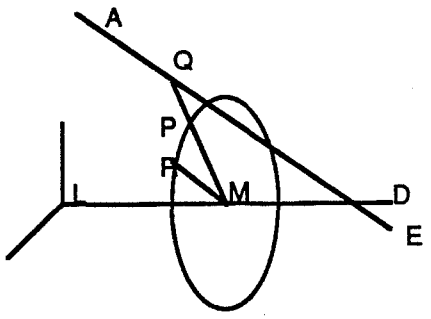


Figure 5.8 Rotation 1D

Au niveau de l'implémentation, le curseur est représenté par trois vecteurs orthogonaux. L'un d'entre eux a une couleur spéciale qui indique la direction principale du curseur. Des fonctions sont définies pour changer la direction principale et transformer le curseur.

5.3.2 Attachements rigide et élastique

Quand on déplace un point P, on peut lui attacher un autre point Q tel que le déplacement de P entraîne le déplacement de Q.

Cet attachement peut être rigide ou élastique. Si Q est attaché à P d'une manière rigide, le déplacement de P entraîne le même déplacement de Q. Si P est déplacé en P', Q est déplacé en Q' de telle sorte que :

$$Q' = Q + P' - P$$

Si Q est attaché à P d'une manière élastique, le déplacement de P entraîne le déplacement de Q d'une façon affine. Si $P = (x_1, y_1, z_1)$ est déplacé en $P' = (x'_1, y'_1, z'_1)$, $Q = (x_2, y_2, z_2)$ est déplacé en $Q' = (x'_2, y'_2, z'_2)$ de telle manière que :

$$x'_2 = x_2 \frac{x'_1}{x_1}$$

$$y'_2 = y_2 \frac{y'_1}{y_1}$$

$$z'_2 = z_2 \frac{z'_1}{z_1}$$

Dans le développement ci-dessus, nous avons implicitement supposé que l'attachement élastique était défini par rapport au repère du monde. Si l'attachement élastique est défini par rapport à un repère local (le repère local défini par le curseur 3D par exemple) , il faut transformer les points P et Q dans le repère local avant leurs déplacements, effectuer les déplacements d'une manière élastique dans le repère local, et transformer les points déplacés dans le repère du monde.

On peut aussi définir un attachement élastique pour la rotation, qui n'est en revanche pas une transformation affine.

On peut attacher plusieurs points à un point à déplacer. Prenons le maillage de points de contrôle de la déformation libre par exemple. On peut attacher à un point de contrôle à déplacer une ligne de points de contrôle, un plan de points de contrôle, ou tout le maillage de points de contrôle.

Si on attache à un point d'un polyèdre touts les autres points, le déplacement de ce point entraînera une transformation affine sur le polyèdre, qui peut être une translation, une rotation, ou une affinité, selon le type d'attachement utilisé.

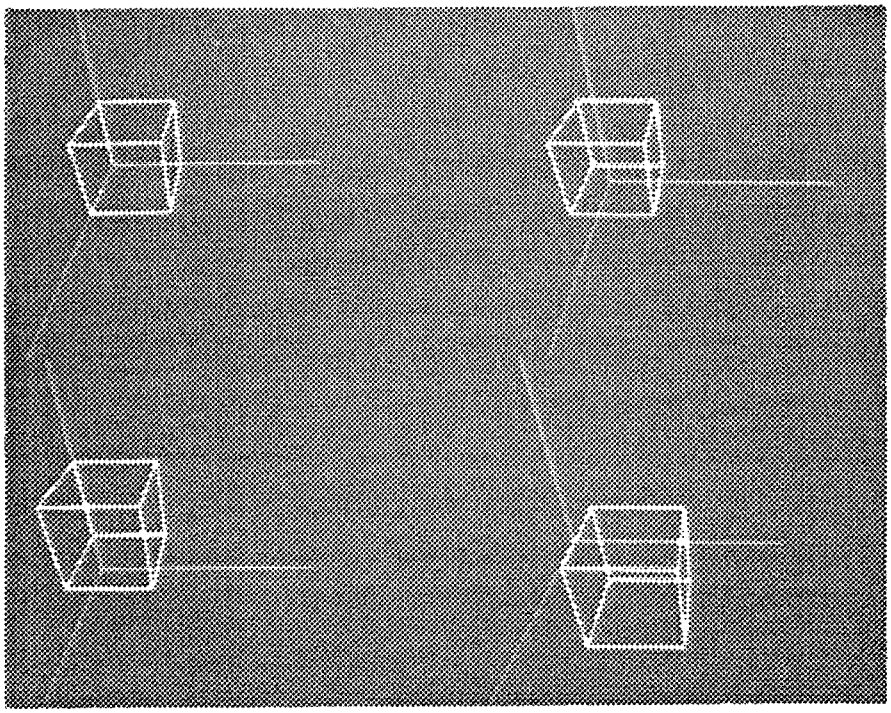
5.3.3 Temps de réponse

Pour que l'interface graphique soit réellement utilisable, il est important de pouvoir calculer rapidement une nouvelle image. L'expérience montre qu'il faut calculer au moins cinq images par seconde pour obtenir un effet de réponse instantanée.

Pour chaque déplacement de points de contrôle, on a plusieurs façons de visualiser l'objet déformé. Il serait idéal de pouvoir visualiser directement l'objet déformé avec rendu. Cela n'est possible qu'avec un support matériel spécial (une machine graphique haut de gamme). Une simplification consiste à visualiser l'objet déformé en fil de fer. La façon la plus simple consiste à visualiser seulement le maillage de points de contrôle en fil de fer, au lieu de visualiser directement l'objet déformé. Dès qu'on est satisfait de la forme du maillage, on peut visualiser l'objet déformé.

Nous choisirons la méthode de visualisation selon la complexité de l'objet à déformer et la puissance de la machine utilisée, telle que la réponse donne un effet instantané.

La figure 5.9 montre l'utilisation du curseur 3D. Nous utilisons le curseur 3D pour déplacer un sommet d'un cube (a). Notons que le curseur 3D peut aussi être déplacé comme un objet (b). Si on utilise un attachement rigide, le déplacement en mode translation 1D (c) et en mode translation 2D (d) réalise la translation du cube. Si on utilise un attachement élastique, le déplacement en mode translation 1D (e) et en mode translation 2D (f) réalise l'affinité du cube. Si on utilise un attachement rigide, le déplacement en mode rotation 1D (g) et en mode rotation 2D (h) réalise la rotation du cube. L'expérience montre que le déplacement en mode rotation 2D est peu util.



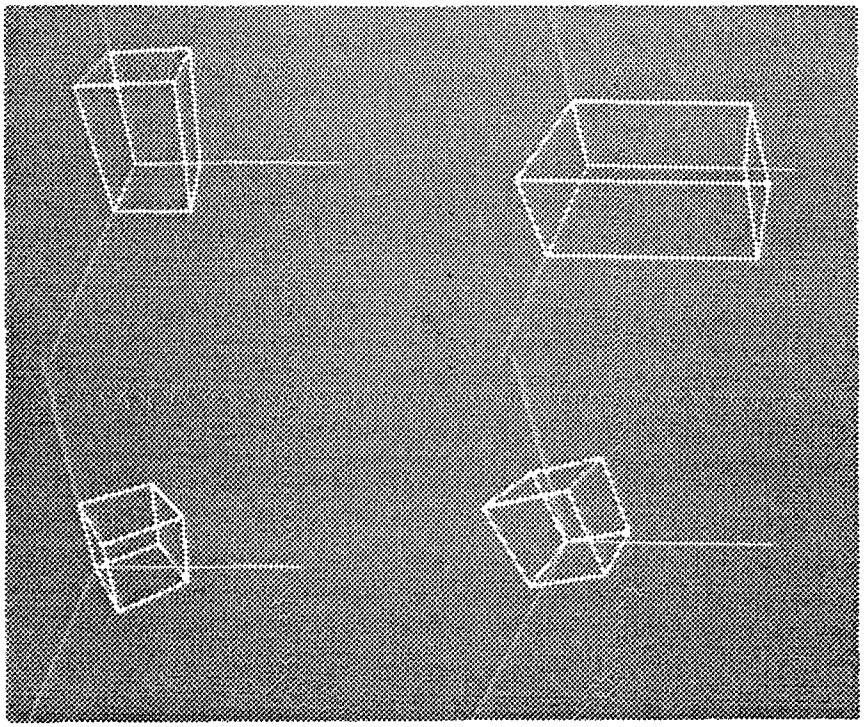


Figure 5.9 Utilisation du curseur 3D

5.4 Animation à l'aide de la déformation libre

L'animation en synthèse d'images consiste à calculer une série d'images fixes et à les projeter à une certaine fréquence pour donner un effet de mouvement de la scène. Un système d'animation donne à l'utilisateur des méthodes pour spécifier l'animation. Ces méthodes doivent être flexibles et faciles à utiliser.

Dans l'animation cinématique, c'est l'utilisateur qui spécifie le mouvement des objets. Le problème essentiel pour le système est d'économiser les calculs.

Pour réaliser une animation réaliste, il faut introduire la dynamique dans l'animation. Dans ce cas-là, le problème principal pour le système est de formuler les équations dynamiques et de les résoudre.

5.4.1 Animation cinématique

La déformation libre est spécifiée par le déplacement des points de contrôle. Pour animer la déformation, il suffit de considérer les points de contrôle comme des fonctions du temps :

$$P_{ijk}(v)$$

où v représente le temps.

Pour un point P qui a (s, t, u) comme paramètres d'interpolation, son animation est calculée avec l'interpolation volumique des points de contrôle mobiles :

$$P(s,t,u,v) = \sum_{i=0}^{l} N_i(s) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} N_k(u)\, P_{ijk}(v)$$

Les fonctions de mélange $N_i(s)$, $N_j(t)$, $N_k(u)$ sont indépendantes du temps. On peut les évaluer avant l'animation pour chaque point sur l'objet à déformer. A chaque étape d'animation, le calcul se limitera à une addition des points de contrôle multipliés par les fonctions de mélange précalculées.

Les trajectoires des points de contrôle peuvent, elles aussi, être un résultat d'interpolation. Etant données les positions clés des points de contrôle, on peut définir leurs trajectoires par une interpolation. Dans ce cas-là, l'animation à l'aide de la déformation libre devient :

$$P(s,t,u,v) = \sum_{i=0}^{l} N_i(s) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} N_k(u) \left[\sum_{h=0}^{o} N_h(v)\, P_{ijkh} \right]$$

$$= \sum_{h=0}^{o} N_h(v) \left[\sum_{i=0}^{l} N_i(s) \sum_{j=0}^{m} N_j(t) \sum_{k=0}^{n} N_k(u)\, P_{ijkh} \right]$$

On obtient une interpolation de dimension quatre qui est homogène pour l'objet à déformer et le temps. D'après l'équation ci-dessus, on a deux façons pour calculer l'animation. La façon directe consiste à évaluer les points de contrôle et à effectuer l'interpolation volumique à chaque étape d'animation. On peut aussi calculer avant l'animation les déformations libres clés, c'est-à-dire les déformations libres avec les positions clés des points de contrôle. A chaque étape d'animation, on aura seulement une

interpolation de dimension un à effectuer. Cette approche procure un gain de temps mais nécessite plus de place mémoire.

L'animation à l'aide de la déformation libre peut être utilisée pour animer un drapeau, les muscles humains, etc. Dans le système Critter Construction de [CHAD 88], le corps humain est modélisé en deux couches. La couche intérieure est une structure rigide qui simule le squelette. La couche extérieure est une structure déformable qui simule les muscles. L'utilisateur contrôle l'animation du squelette, qui contrôle à son tour l'animation des muscles. Prenons le bras comme exemple. Le squelette est modélisé avec deux objets rigides articulés, et le muscle est modélisé avec un objet déformable. Un maillage de points de contrôle est défini autour du muscle. Le déplacement des points de contrôle est contrôlé par le squelette, et la déformation libre est appliquée sur le muscle. Si un morceau du squelette effectue une rotation par rapport à l'autre, on appliquera une courbure sur le maillage de points de contrôle et le muscle sera déformé. Si la rotation est trop importante, une discontinuité C^1 sera introduite pour simuler le pli entre les deux zones.

5.4.2 Animation dynamique

Actuellement les systèmes d'animation utilisent de plus en plus la dynamique pour produire des animations réalistes.

La modélisation de l'animation dynamique consiste à fournir une équation dynamique qui contrôle le mouvement des objets. Pour supporter une telle modélisation, le système d'animation doit inclure des modules permettant de résoudre des équations dynamiques, qui sont en général des équations différentielles.

Les travaux menés portent sur la dynamique des objets rigides articulés, la dynamique des objets déformables et la collision entre des objets. Nous nous intéressons à l'utilisation de la dynamique pour animer des objets déformables [TERZ 87] [HAUM 87] [TERZ 88] [GASC 90].

On peut utiliser la déformation libre pour simplifier l'animation dynamique des objets déformables. Pour calculer l'animation dynamique d'un objet déformable, on définit une déformation libre dont la région de déformation contient l'objet à déformer. On considère la région de déformation comme un objet qui hérite de toutes les propriétés dynamiques de l'objet à déformer. On calcule l'animation dynamique de cet objet virtuel,

ce qui donne l'animation du maillage de points de contrôle, et donc l'animation de l'objet à l'aide de la déformation libre.

Avec l'utilisation de la déformation libre, l'animation dynamique d'un objet déformable devient indépendante de la forme géométrique de l'objet à déformer. Comme la forme de la région de déformation est en général beaucoup plus simple que celle de l'objet à déformer, la résolution de l'équation dynamique est simplifiée. On peut utiliser la méthode des différences finies au lieu des éléments finis pour effectuer la résolution.

5.5 Conclusions

Nous utilisons la déformation libre sur l'arbre de construction comme extension des transformations affines. Cette extension élargit considérablement l'espace des objets qu'un arbre de construction peut modéliser. Une déformation libre est plus coûteuse à évaluer qu'une transformation affine. Mais cela reste raisonnable, car la déformation libre est une transformation polynomiale.

On peut utiliser la déformation libre pour réaliser une déformation globale, en appliquant la déformation globale sur le maillage de points de contrôle. En général, la déformation libre ne donne qu'une déformation approximative par rapport à la déformation initiale. En revanche, l'utilisation de la déformation libre simplifie la programmation et économise le temps de calcul.

Comme la définition d'une déformation libre est complexe, il est important d'utiliser une interface propre. Si on utilise la déformation libre pour réaliser une déformation globale, on peut utiliser une interface syntaxique. Sinon, une interface graphique sera indispensable. Nous utilisons un curseur 3D pour réaliser l'interface graphique. Avec le curseur 3D, on peut appliquer une translation ou une rotation sur un point ou un ensemble de points.

On peut réaliser l'animation d'un objet déformable à l'aide de la déformation libre. Pour l'animation cinématique, l'utilisateur spécifie les trajectoires des points de contrôle et le système d'animation effectue la déformation libre pour animer l'objet. Les trajectoires peuvent être spécifiées soit comme des fonctions du temps, soit comme une interpolation à partir de positions clés. Pour l'animation dynamique, on peut considérer la région de déformation comme un objet et calculer son animation dynamique. L'animation de l'objet est réalisée à l'aide de la déformation libre. L'animation dynamique devient alors indépendante de la forme de l'objet à déformer.

Bibliographie

[ALIA 87] G. Alia, F. Barsi, E. Martinelli and N. Tani, "Angular spline: a new approach to the interpolation problem in computer graphics", Computer Vision, Graphics, and Image Processing, 39, 56-72, 1987.

[ALLA 89] J. B. Allan, B. Wyvill, and I. H. Witten, "A Methodology for direct manipulation of polygon meshes", CGI 89, pp451-469.

[ATHE 83] P. Atherton, "A scan-line hidden surface removal procedure for constructive solid geometry", SIGGRAPH'83, Computer Graphics, vol.17, no.3, pp.73-82.

[BARR 81] A. H. Barr, "Superquadrics and angle-preserving transformations", IEEE Computer Graphics and Applications, 1(1), 11-23.

[BARR 84] A. H. Barr "Global and local deformations of solid primitives", Computer Graphics, July 1984, pp 21-30.

[BARR 86] A. H. Barr, "Ray tracing deformed surfaces", Computer Graphics, August 1986, pp 287-296.

[BART 86] R. H. Bartels, "Introduction to the use of splines in free-form curve and surface design", Siggraph 86, Course Notes 4.

[BEIG 88] M. Beigbeder, "Un développement pour la modélisation et la visualisation en synthèse d'images: CASTOR", thèse de doctorat, Ecole des mines de Saint-Etienne, Avril 1988.

[BEIG 89] M. Beigbeder et B. Peroche "Un système de synthèse d'images 3D : ILLUMINES", les Journées UNIX de Grenoble, 1989.

[BIAR 90] L. Biard, "Parametric surfaces and ray tracing", Research report, L.M.C. IMAG Grenoble, Mai 1990.

[BRES 65] J. E. Bresenham, "Algorithm for computer control of a digital plotter", IBM System Journal, vol.4, no.1, 1965.

[BRON 85] W. F. Bronswoort and F. Klok "Ray Tracing Generalized Cylinders", ACM trans. Graphics, Oct. 1985, pp 291-303.

[CARL 82] W. Carlson, "An algotithm and data structure for 3D object synthesis using surface patch intersections", Computer Graphics, vol.16, no.3, July 1982.

[CATM 74] E. Catmull, "A subdivision algorithm for computer display of curved surfaces", Ph.D. Dissertation, University of Utah, 1974.

[CHAD 88] J. E. Chadwick and E. Parent, "Critter Construction : developing characters for computer animation", PIXIM 88, pp 283-305.

[CHIY 83] H. Chiyokura, F. Kimura, "Design of solids with free-form surfaces", Computer Graphics, Siggraphe 83, pp 289-298.

[COQU 90] S. Coquillart, "Extended Free-Form Deformation: A Sculpturing Tool for 3D Geometric Modeling", Computer Graphics, Volume 24, Number 4, 1990, pp 187-196.

[COUS 90] O. Cousin, G. Hégron, "Une modélisation des cônes généralisé à l'aide des splines", CFAO, Volume 4, Number 3, 1989.

[DERO 89] T. Derose and al. "Apex : two architectures for generating parametric curves and surfaces", The visual cmputer, May 1989, pp 264-276.

[FARI 88] G. Farin, "Curves and surfaces for computer aided geometric design : a pratical guide", Academic Press, 1988.

[FOLE 82] J. Foley, A. Van Dam, "Fundamentals of interactive computer graphics", Addison-Wesley, 1982.

[FOUR 83] A. Fournier, M. Wesley, "Bending polyhedral objects", Computer Aides Design, vol.15, no.2, March 1983.

[GAIL 88] V. Gaildrat, R. Caubet, Y. Duthen, "VOXAR, a tridimentional architecture for fast realistic images synthesis", Computers graphics International, Genève, 1988.

[GASC 90] M. P. Gascuel, A. Verroust, C. Puech, "Objets déformables vertébrés : animation dynamique et traitement des collisions", GROPLAN 90.

[GLAS 89] A. Glassner, "An introduction to ray tracing", Academic Press, 1989.

[GRIE 89] J. Griessmair and W. Purgathofer "Deformation of solids with trivariate B-splines", EUROGRAPHICS 89, pp 137-148.

[HAUM 87] D. Haumann, "Modeling the physical behavior of flexible objects", Siggraph 87, Tutorial 17.

[HERZ 87] B. Herzen, A. Barr, "Accurate triangulations of deformed, intersecting surfaces", Computer Graphics, vol.21, no.4, July 1987.

[JESS 89] J.P. Jessel, P. Pitot, R. Caubet, Y. Duthen, "Déformations d'objets modélisés par réseaux d'atomes", GROPPLAN 89.

[KAJI 82] J. T. Kajiya, "Ray tracing parametric patches", Computer Graphics, July 1982, pp 245-254.

[LANE 80] J. Lane, L. Carpenter, T. Whitted, J. Blinn, "Scan line methods for displaying parametrically defined surfaces", Communications of the ACM, vol.23, No.1, January 1980.

[LIEN 87] S. Lien, M. Shantz, V. Pratt, "Adaptive forward differencing for rendering curves and surfaces", Computer graphics, vol.21, no.4, July 1987.

[MATA 89] N. Matallah, R. Caubet, "Une approximation optimisée de surfaces paramétriques par des facettes triangulaires", MICAD 89, pp 773-782.

[MICH 87] D. Michelucci, "les représentations par frontières: quelques constructions; difficultés rencontrées", thèse de doctorat, Ecole des mines de Saint-Etienne, Novembre 1987.

[MILL 88] G. S. P. Miller, "The motion dynamics of snakes and worms", Computer Graphics, August 1988, pp 169-178.

[NEWM 79] W. M. Newman and R. F. Sproull "Principes of interactive computer graphics", edition McGRAW-HILL, 1979.

[NISH 90] T. Nishita, T. Sederberg, M. Kakimoto, "Ray tracing trimmed rational surface patches", Computer Graphics, vol.24, no.4, August 1990.

[NIE 89] Nie Z. et B. Péroche, "Déformations libres et arbre de construction", Gropplan 89.

[NIE 90] Nie Z. et B. Péroche, "Contributions to Free-Form Deformation", Rapport de recherche, Ecole des mines de Saint-Etienne, No.90.12, Décembre 1990.

[OVER 89] C. Overveld, "Application of a perspective cursor as a 3D locator device", Computer Aided Design, vol.21, no.10, December 1989.

[PARE 77] R. E. Parent, "A system for sculpting 3-D data", Computer graphics, July 1977, 138-147.

[PARR 86] S. R. Parry "Free-form deformations in a constructive solid geometry modeling system", Ph.D, Brigham Young University, 1986.

[PERO 88] B. Péroche, J. Argence, D. Ghazanfarpour, D. Michelucci, "la synthèse d'images", Hermès, Paris.

[POST 86] F. Post, F. Klock, "Deformations of sweep objects in solid modelling", EUROGRAPHICS'86, pp.103-114.

[REQU 80] A. Requicha, "Representations for rigid solids: theory, methods, and systems", Computing Surveys, Vol.12, No.4, December 1980.

[SAME 82] H. Samet, "Neighbor finding techniques for images represented by quadtrees", Computer Graphics and Image Processing, Volume 18, 1982, P.37.

[SAME 84] H. Samet, "The quadtree and related hierarchical data structures", Computing Surveys, 16,2, June 1984, 187-260.

[SEDE 86] T. W. Sederberg and S. R. Parry, “Free-form deformation of polygonal data”, Deuxième Colloque Image, Nice, Avril 1986.

[SEDE 86] T. W. Sederberg and S. R. Parry "Free-form deformation of solid geometric models", Computer Graphics, July 1986, pp 151-160.

[SNYD 87] J. Snyder, A. Barr, “Ray tracing complex models containing surface tessellations”, Computer Graphics, vol.21, no.4, July 1987.

[TERZ 87] D. Terzopoulos, J. Platt, A. Barr and K. Fleischer, "Elastically deformable models", Computer Graphics, July 1987.

[TERZ 88] D. Terzopoulos, K. Fleischer, "Modeling ielastic deformation: viscoelasticity, plasticity, fracture", Computer Graphics, August 1988.

[THOM 86] S. W. Thomas, “Scattered thoughts on B-splines”, Computer Graphics Workshop, Monterey, November 1986.

[WITK 87] A. Witkin, K. Fleischer, A. Barr, "Energy constraints on parameterized models", Computer Graphics, July 1987, pp 225-232.

[WATE 87] K. Waters, "A mucle model for animating three-dimensioal facial expression", Computer Graphics, July 1987.

Résumé :

Depuis quelques années, les déformations sont utilisées en synthèse d'images pour la modélisation des solides. La déformation libre est une méthode de transformation qui déforme des objets en déplaçant un maillage de points de contrôle définis dans un repère local et en utilisant une interpolation volumique. Deux extensions ont été développées. La première utilise l'interpolation de type B-spline, au lieu de l'interpolation de type Bézier. La deuxième utilise un repère local cylindrique ou sphérique, au lieu d'un repère local cartésien. Pour visualiser des objets déformés, un algorithme de facettisation adaptative est proposée. Pour déplacer des points de contrôle dans l'espace 3D avec un localiseur 2D, un curseurs 3D est utilisé.

Mots clés :

Synthèse d'images, modélisation géométrique, déformation, déformation libre, facettisation adaptative, interface graphique.