Approximation et intersection des surfaces procédurales utilisées en C.A.O.

Stéphane Chau

UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS - UFR Sciences

École Doctorale Sciences Fondamentales et Appliquées

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'UNIVERSITÉ de Nice-Sophia Antipolis

Spécialité : Mathématiques

présentée et soutenue par

Stéphane Chau

Approximation et intersection des surfaces procédurales utilisées en C.A.O.

Thèse dirigée par André Galligo et Mohamed Elkadi
soutenue au laboratoire J.A. Dieudonné le 10 Juin 2008

Membres du jury :

Mme. Gema Maria Diaz-Toca	Professeur à l'Université de Murcia	Examinateur
M. Mohamed Elkadi	Maître de conférence à l'Université de Nice	Directeur
M. André Galligo	Professeur à l'Université de Nice	Directeur
M. Laureano Gonzalez-Vega	Professeur à l'Université de Cantabria	Rapporteur
M. Bert Jüttler	Professeur à l'Université Johannes Kepler	Rapporteur
M. Bernard Mourrain	Directeur de recherche à l'INRIA	Président
M. Jean-Claude Yakoubsohn	Professeur à l'Université de Toulouse	Rapporteur

Approximation et intersection des surfaces procédurales utilisées en C.A.O.

Stéphane Chau

Remerciements

Il est de coutume de consacrer quelques pages de remerciements dans une thèse et la mienne ne fera pas exception à la règle. Bien entendu, et ce n'est pas par conformité, je voudrais sincèrement remercier mes directeurs de thèse **André Galligo** et **Mohamed Elkadi** qui m'ont consacré beaucoup de temps et m'ont énormément apporté que ce soit sur le plan scientifique ou humain. Mon expérience dans le milieu universitaire m'a fait comprendre que les relations doctorant-directeur(s) peuvent parfois être difficiles en raison d'incompatibilité de caractères. Ce constat me permet de réaliser que je fais partie d'une minorité qui a eu la chance de côtoyer des encadrants avec lesquels je me suis particulièrement bien entendu. Cela a fortement contribué à mon épanouissement tout au long du projet de thèse dont le manuscrit présent ne reflète qu'une infime partie.

Cette dernière remarque me paraît importante dans la mesure où il faut bien réaliser qu'une multitude de facteurs contribue au bon déroulement d'une thèse et que l'auteur ne fait pas tout. Parmi eux, il y a évidemment les personnes qui sont autour du projet et qui permettent d'enrichir son contenu. Je pense tout d'abord aux membres du projet GALAAD de l'INRIA qui m'ont beaucoup apporté grâce à leurs compétences spécifiques. En particulier je voudrais remercier **Bernard Mourrain** pour son regard expert à tous les niveaux, **Julien** pour tous les conseils qu'il m'a apporté sur le plan informatique et **Jean-Pascal** qui m'a fait découvrir, avec son sens de la communication absolument unique, les problèmes de la CAO. Il y a également **Laurent**, **Marc**, **Lionel** et **Jean-Pierre** avec qui j'ai eu souvent des discussions fructueuses. Je remercie aussi les personnes qui ont accepté

de faire partie de mon **jury** de thèse et en particuliers les **rapporteurs** qui ont eu la lourde tâche de me relire.

De même que la technologie avancée d'une Formule 1 ne serait rien sans des pneus de qualité pour exploiter son potentiel, une thèse doit son existence aussi en partie à l'entourage proche de son auteur. Cette métaphore doit en faire sourire plus d'un et je saisis cette opportunité pour citer toutes les personnes du laboratoire Dieudonné avec lesquelles j'ai beaucoup ri. Je veux remercier **Pierre**, qui a partagé mon bureau et mes amphis il y a quelques années en arrière, pour son look inimitable, **Mig** pour ses formules de politesse qui riment avec bus, **Fabien** pour sa bonne humeur permanente et bien sûr tous les autres : **Delphine**, **Mimi**, **Thi Ha**, **Marcello**, **Olivier**, **La maman de Josué et Lucie**, **You**, **Nico**, **Thomas**, **Joan**, **Xavier**...

Les moments de détente ne se résumant pas qu'au bureau, je me dois aussi de nommer les copains qui ne font pas partie du labo mais avec qui j'ai passé de très bons moments : **Laura** qui m'a beaucoup soutenu et fait découvrir des perles musicales, **Marie** qui me fait toujours beaucoup rire avec ses anecdotes, **Elise** chez qui on a passé des soirées mémorables, **Claire**, **Vincent**, **Julie**, **Pascale**,...

J'ai aussi une pensée toute particulière pour **Coralie** et **Kiwi** qui partagent ma vie et qui ont bien du mérite à supporter un énergumène comme moi surtout après certaines journées de travail où je n'ai eu comme seul interlocuteur que mon écran d'ordinateur.

Enfin, je terminerai en remerciant ma **famille** et ma **belle famille**.

Table des matières

Introduction

Contexte du projet de recherche. Dans le contexte de la conception et de la réalisation de produits manufacturables actuel, l'outil informatique est de plus en plus utilisé. Des domaines variés tels que la conception mécanique, l'électronique, le bâtiment ou la confection de vêtements font désormais fréquemment usage de ce que l'on appelle la conception assistée par ordinateur (CAO en abrégé). Faisant l'objet d'une forte demande, une multitude de logiciels de CAO ont fait leur apparition. L'enjeu économique est important car l'utilisation d'un tel programme informatique fait considérablement baisser les coûts de réalisation. En effet, la conception de tout système technique implique des interactions ainsi que des incompatibilités éventuelles entre les fonctions qui la constituent. La connaissance et la maîtrise de ces paramètres font partie du savoir de l'ingénieur. Cependant, lorsque ces variables sont trop nombreuses, la complexité de ce système dépasse la capacité de l'être humain. En faisant appel à la conception virtuelle, on peut apprécier le comportement global d'un objet avant même que celui-ci n'existe. On se met ainsi à l'abri de la perte onéreuse d'un prototype confectionné ne satisfaisant pas toutes les contraintes demandées et qui, de ce fait, serait inexploitable. Le développement d'un logiciel de CAO demande de bonnes connaissances en informatique bien sûr, mais également en mathématique car les objets manipulés à l'intérieur d'une telle application sont de nature géométrique puisque ce sont les courbes et les surfaces. Cette discipline est aujourd'hui mature mais, malgré son succès et un engouement important au niveau de la recherche et du développement, de nombreux problèmes subsistent. Parmi eux, le problème d'intersection, qui fait apparaître des situations singulières,

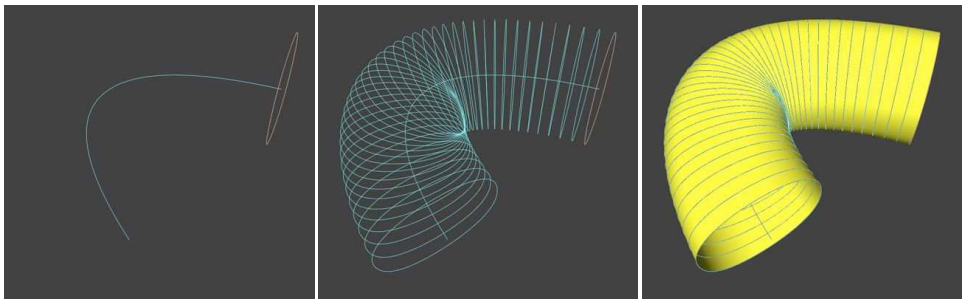


FIG. 1: Création d'un tube présentant une auto-intersection.

sur lequel se concentrent des subtilités et des difficultés.

Problème. Dans la pratique, les logiciels de CAO utilisent des surfaces dites « procédurales » *i.e.* données uniquement par évaluations. Ce choix répond à une contrainte essentielle. En effet, dans une suite de constructions, à travers les diverses fonctionnalités de CAO, l'utilisateur peut à tout moment vouloir changer certains paramètres dans l'une des étapes. Cela peut provenir de contraintes esthétiques, mécaniques, aérodynamiques... Or, cette possibilité ne peut lui être offerte que si le logiciel conserve un historique de toutes les constructions et cela est impossible si les équations correspondantes sont calculées au fur et à mesure.

Les surfaces sont essentiellement utilisées pour délimiter des volumes. Par conséquent, leurs courbes d'intersection font nécessairement partie de leur description. Par exemple, en conception mécanique, si on fabrique deux pièces qui doivent s'emboîter, il est primordial de savoir faire l'intersection des surfaces qui enveloppent ces deux objets. Mais il se peut également que des intersections involontaires se produisent lors de la conception virtuelle et que ces dernières posent des problèmes lors de la fabrication effective de l'objet confectionné. La figure 1 illustre bien ce phénomène. Il s'agit de la conception d'un tube à partir d'une courbe (ici un cercle) qui se déplace le long d'une deuxième courbe (ici une parabole). Ce tube présente une intersection (on parle « d'auto-intersection ») en raison d'une courbure trop importante de cette seconde courbe. Il est essentiel que le logiciel détecte ce genre de phénomènes.

Etant donné son importance, on pourrait s'attendre à ce que le problème

FIG. 2: Approximation d'un offset par une B-spline.

d'intersection soit l'un des mieux résolus du domaine, or il n'en est rien car les exigences des utilisateurs, et donc les surfaces utilisées, évoluent constamment en complexité. La façon la plus ancienne de considérer le problème est de représenter chacune des surfaces par un maillage (approximations locales par des triangles) et de procéder à leur intersection. Malgré les efforts de la communauté scientifique pour proposer des alternatives, elle demeure, encore aujourd'hui, une des plus employées du fait de sa simplicité de manipulation. L'inconvénient significatif est que le résultat peut être de très mauvaise qualité lorsque l'on est en présence de triangles « presque » parallèles, situation qui correspond à des cas d'intersections « presque » tangentes.

Une autre approche consiste à utiliser des représentations globales des surfaces par des B-splines et traiter alors le problème d'intersection entre de tels modèles ([19]). Cependant, les procédures usuelles de CAO ne conservent pas toujours ce modèle. L'offset, qui consiste à translater point par point une surface suivant la normale correspondante, en est un bon exemple car ce n'est pas une opération rationnelle. Comme il a été dit précédemment, on utilise plutôt des surfaces procédurales pour représenter une suite de constructions (dans notre cas l'offset) et une approximation par B-spline est alors calculée pour des opérations futures. Cela est illustré sur la figure 2. Il s'agit d'une surface B-spline (à gauche) sur laquelle un offset est appliqué et dont le résultat est approximé par une autre B-spline (figure du milieu). On s'aperçoit que, dans ce contexte, l'approximation par une B-spline nécessite une représentation relativement dense par rapport à la surface de départ (les points de contrôle apparaissent sur les figures de gauche et du mi-

lieu). L'avantage ici vient du fait que les intersections tangentes peuvent être détectées contrairement à l'approche par maillage. Cependant, l'approximation est globale et par conséquent est, en quelque sorte, séparée du procédé d'intersection. Dans ce contexte, l'intersection est déterminée également de manière globale et peut donc présenter localement une qualité insuffisante.

Approche. Dans le cadre de ce projet de thèse, on s'intéresse à une méthode d'approximation des surfaces en utilisant des approximants plus fins et surtout moins rigides que les triangles. Les formes complexes étant difficiles à approcher avec des triangles, les systèmes de CAO sont contraints d'utiliser des structures de maillages très lourdes (besoin de beaucoup de triangles) ce qui augmente d'autant plus la difficulté du problème d'intersection. Nous privilégions une approche utilisant un modèle d'approximation par des carreaux de surfaces biquadratiques. Cela apporte une plus grande variété de formes géométriques avec une représentation plus compacte que celle que l'on aurait avec des triangles. De plus, pour le problème d'intersection, cela représente une alternative intéressante. En effet, c'est un compromis entre la méthode utilisant des triangles et celle utilisant des B-splines dans la mesure où cette approximation est locale (comme dans le cas des triangles) et permet de détecter des cas d'intersection tangente (comme dans le cas des B-splines).

Pour mettre en oeuvre une telle approche, il faut savoir maîtriser la géométrie des entités élémentaires utilisées. Il s'agit de mettre en place une méthode pour approximer les surfaces procédurales et ensuite de traiter le problème d'intersection de manière efficace. Les difficultés se concentrent principalement sur ce dernier point, il sera donc traité plus en détails. Plusieurs méthodes seront proposées et illustrées tant du point de vue théorique, c'est à dire mathématiquement, que pratique, c'est à dire informatiquement.

Structure de la thèse. La thèse est divisée en quatre parties. Le chapitre 1 présente des outils et résultats algébriques qui nous seront utiles. Dans le but de ne pas surcharger inutilement le contenu, on s'est contenté d'aller directement à l'essentiel en formulant les résultats dans le contexte qui est le notre.

Le chapitre 2, quant à lui, traite de l'approximation des surfaces procédurales

par des carreaux de surfaces biquadratiques et l'application de ce modèle au problème d'intersection issu de la CAO. Plus précisément, on met en place une structure de données efficace qui détecte rapidement les sous-domaines où aucune intersection n'est présente. Ensuite, l'algorithme se concentre sur des intersections « pertinentes » entre surfaces paramétrées polynomiales.

Le chapitre 3 traite des configurations d'intersection entre surfaces paramétrées polynomiales. Si on recense brièvement les différentes manières dont ce sujet a déjà été abordé, on peut essentiellement distinguer les techniques algébriques ([9, 28, 36]), de *marching* ([4, 5, 54]) et de subdivision ([19, 42]). Dans cette thèse, les contributions apportées sur la question portent sur plusieurs approches. Parmi elles, on peut mentionner la méthode utilisant les résultants qui fait appel à des projections autres que celles rencontrées chez de nombreux auteurs. Il s'agira de considérer des espaces de paramètres correspondant à des variables séparées et on verra que ce choix est plus efficace que celui des projections « naturelles » dans les espaces de paramètres de chaque carreau de surface. On peut aussi mentionner la méthode dite par représentation semi-implicite qui utilise la structure particulière des carreaux biquadratiques. Dans ce cas, toujours dans un objectif d'efficacité, la forme spécifique des équations nous permettra d'exprimer explicitement certaines données primordiales pour le calcul d'intersection. Enfin, contrairement aux approches habituelles d'intersection par subdivision qui utilisent des projections se ramenant à $\mathbb{R}^2$, on étudiera la courbe d'intersection dans $\mathbb{R}^4$ de deux carreaux de surfaces. En fait, on donnera non seulement une méthode de calcul de topologie de cette courbe dans $\mathbb{R}^4$ mais on traitera également les problèmes de noeud qui se posent lorsqu'on désire « remonter » cette topologie dans $\mathbb{R}^3$. Ce dernier point est important et est souvent omis, comme par exemple dans [40].

Pour finir, le chapitre 4 traitera de certains aspects, aussi bien pratiques que techniques, du logiciel Axel ([12]) qui est l'environnement dans lequel la plupart des algorithmes décrits sont implémentés et qui a servi à créer une bonne partie des illustrations de cette thèse.

Chapitre 1

Outils algébriques et numériques

On se propose d'introduire dans ce chapitre les outils algébriques et numériques qui seront utilisés. Dans la mesure où ces notions sont pour la plupart largement connues, on ne rentrera pas dans les détails notamment en ce qui concerne les démonstrations. Toutefois, on s'efforcera de donner des références pour compléter les sujets abordés.

1.1 Base de Bernstein

Les objets algébriques que l'on manipulera seront les polynômes. En algèbre, ces derniers sont exprimés dans la base monomiale, qui peut-être qualifiée de base canonique. Cette représentation est naturelle et a l'avantage d'être parfois compacte (notamment lorsque le polynôme est de degré « élevé » et qu'un nombre important de monômes n'apparaissent pas). De plus, la plupart des résultats d'algèbre commutative utilisent les polynômes en tant qu'objets intrinsèques et ne font donc jamais de changement de représentation. Cependant, la base monomiale n'est pas vraiment adaptée dès lors que l'on désire avoir des informations numériques ou géométriques sur les fonctions polynômes restreintes sur un domaine fixé. Dans ce contexte, la base de Bernstein est une alternative intéressante notamment car elle est reconnue pour être numériquement plus stable que la base monomiale (voir [27]) et également car elle offre des propriétés utiles pour la modélisation de formes.

Dans les sections qui suivent, on donnera les définitions de cette base dans les cas univarié et multivarié ainsi que des propriétés remarquables. On verra également les conséquences pratiques qui découlent de ces propriétés.

1.1.1 Base de Bernstein univariée

Définition 1.1 (Base de Bernstein univariée) : Soit $f(x) \in \mathbb{R}[x]$ un polynôme univarié de degré $d \in \mathbb{N}^*$ et $[\alpha, \beta] \subset \mathbb{R}$ avec $\alpha < \beta$. Alors, il existe une unique famille $(b_0, \ldots, b_d) \in \mathbb{R}^{d+1}$ telle que f s'écrive sous la forme :

$$f(x) = \sum_{i=0}^{d} b_i B_i^d(x)_{[\alpha,\beta]}$$
$$B_i^d(x)_{[\alpha,\beta]} = \binom{d}{i} \frac{(\beta - x)^{d-i}(x - \alpha)^i}{(\beta - \alpha)^d}$$

Les polynômes $B_i^d(x)_{[\alpha,\beta]}$ sont appelés les polynômes de Bernstein de degré d associés à $[\alpha, \beta]$ et la famille $\left(B_i^d(x)_{[\alpha,\beta]}\right)_{0 \leq i \leq d}$ est appelée la base de Bernstein du $\mathbb{R}$-espace vectoriel des polynômes de degrés inférieurs ou égaux à d (noté $\mathbb{R}_d[x]$) associée à $[\alpha, \beta]$.

Remarque 1.1 :
- Si le contexte précise les valeurs de α et β, alors on se permettra de les omettre dans la notation des polynômes de Bernstein. En d'autres termes on notera de manière plus concise $B_i^d(x)$.
- Si $[\alpha, \beta] = [0, 1]$, alors les polynômes de Bernstein s'écrivent sous la forme $B_i^d(x) = \binom{d}{i}(1 - x)^{d-i}x^i$. Dans la pratique, on se place souvent sur $[0, 1]$ pour simplifier les formules et ce, sans restreindre la généralité. •

Proposition 1.1 (Enveloppe convexe) : Sous les mêmes notations que la définition 1.1, le graphe de f sur $[\alpha, \beta]$ *i.e.* l'ensemble $\{(x, f(x)) \mid x \in [\alpha, \beta]\}$ est entièrement contenu dans l'enveloppe convexe de la famille de points $(\frac{i}{d}, b_i)_{0 \leq i \leq d}$

Propriété 1.1 (Subdivision de De Casteljau) : Soit $f(x) \in \mathbb{R}[x]$ un polynôme univarié de degré $d \in \mathbb{N}^*$ écrit dans la base de Bernstein associée

à $[0,1]$:

$$f(x) = \sum_{i=0}^{d} b_i B_i^d(x).$$

Fixons un $x_0 \in [0,1]$. Alors, on peut définir f par morceaux sur $[0,x_0] \cup [x_0,1]$ par :

$$\begin{aligned}
f^{(0)}(x) &= \sum_{i=0}^{d} b_0^{(i)} B_i^d(x), \text{ si } x \in [0,x_0] \\
f^{(1)}(x) &= \sum_{i=0}^{d} b_i^{(d-i)} B_i^d(x), \text{ si } x \in [x_0,1] \\
b_i^0 &= b_i, \text{ pour } 0 \leq i \leq d \\
b_i^{(k)} &= (1-x_0) b_i^{(k-1)} + x_0 b_{i+1}^{(k-1)} \text{ pour } \begin{cases} 1 \leq k \leq d \\ 0 \leq i \leq d-k \end{cases}
\end{aligned}$$

Remarque 1.2 (Subdivision de De Casteljau dans le cas général) : On peut généraliser la propriété 1.1 sur un intervalle quelconque non vide $[\alpha,\beta]$. Si $\gamma \in [\alpha,\beta]$, alors, il existe un $x_0 \in [0,1]$ tel que $\gamma = (1-x_0)\alpha + x_0\beta$. Pour tout $t \in [\alpha,\beta]$, il existe $x \in [0,1]$ tel que $t = (1-x)\alpha + x\beta$ et on peut alors définir $f(t)$ par morceaux sur $[\alpha,\gamma] \cup [\gamma,\beta]$ en utilisant les formules de la propriété 1.1. •

Définition 1.2 (Changements de signes d'une séquence) : Soit $k \in \mathbb{N}^*$ et $\mathbf{a} = a_1, \ldots, a_k$ une séquence de k nombres réels. Alors, on définit le nombre de changements de signes de la séquence $\mathbf{a}$ par :

$$V(\mathbf{a}) = \#\left\{i \in \{0,\ldots,k-1\} \,|\, a_i a_{i+1} < 0\right\}.$$

Propriété 1.2 (Règle de Descartes) : Soit $f(x) \in \mathbb{R}[x]$ un polynôme univarié de degré $d \in \mathbb{N}^*$ et $[\alpha,\beta] \subset \mathbb{R}$ avec $\alpha < \beta$. On écrit le polynôme f dans la base de Bernstein associée à l'intervalle $[\alpha,\beta]$. Soit $f(x) = \sum_{i=0}^{d} b_i B_i^d(x)$. On note $\mathbf{b}$ la séquence $b_0,\ldots,b_d$. Alors le nombre N de racine(s) de f sur $]\alpha,\beta[$ est inférieur ou égal à $V(\mathbf{b})$. Plus précisément, on a : $N \equiv V(\mathbf{b}) \mod 2$.

Corollaire 1.1 : Une conséquence immédiate de la propriété 1.2 est que :

– si $V(\mathbf{b}) = 0$, alors le nombre de racines réelles de f dans $]\alpha,\beta[$ est 0.

– si $V(\mathbf{b}) = 1$, alors le nombre de racines réelles de f dans $]\alpha, \beta[$ est 1.

Par ailleurs, en prenant également en compte la proposition 1.1, on déduit immédiatement que si tous les coefficients de f dans la base de Bernstein associée à $[\alpha, \beta]$ sont strictement positifs (respectivement strictement négatifs), alors f est strictement positive (respectivement strictement négative) sur $]\alpha, \beta[$.

1.1.2 Base de Bernstein multivariée

On peut généraliser la base de Bernstein univariée dans le cas multivarié.

Définition 1.3 (Base de Bernstein multivariée) : Soit n un entier non nul et $f(\mathbf{x}) \in \mathbb{R}[\mathbf{x}] = \mathbb{R}[x_1, \ldots, x_n]$ un polynôme à n variables de multidegré $(d_1, \ldots, d_n) \in (\mathbb{N}^*)^n$ (*i.e.* f est de degré d_i en la variable x_i pour tout i dans $\{1, \ldots, n\}$). Soit $[\alpha_i, \beta_i]_{1 \leq i \leq n}$ une famille d'intervalles non vides de $\mathbb{R}$. Alors, on peut écrire de manière unique le polynôme $f(\mathbf{x})$ sous la forme :

$$f(\mathbf{x}) = f(x_1, \ldots, x_n) = \sum_{k_1=0}^{d_1} \ldots \sum_{k_n=0}^{d_n} b_{k_1,\ldots,k_n} B_{k_1}^{d_1}(x_1) \ldots B_{k_n}^{d_n}(x_n)$$

Avec, les $b_{k_1,\ldots,k_n}$ dans $\mathbb{R}$ et $(B_j^{d_i}(x_i))_{0 \leq j \leq d_i}$ la base de Bernstein univariée de $\mathbb{R}_{d_i}[x_i]$ associée à l'intervalle $[\alpha_i, \beta_i]$ pour tout i dans $\{1, \ldots, n\}$.

Remarque 1.3 : La subdivision de De Casteljau de la propriété 1.1 procède de la même manière dans le cas multivarié que dans le cas univarié. Il suffit de faire la subdivision de manière indépendante sur chacune des variables x_i. •

Proposition 1.2 : Le corollaire 1.1 s'applique aussi dans le cas multivarié. Explicitement, en reprenant les notations de la définition 1.3, si tous les coefficients de $f(\mathbf{x})$ sont strictement positifs (respectivement strictement négatifs), alors f est strictement positive (respectivement strictement négative) sur $]\alpha_1, \beta_1[\times \cdots \times]\alpha_n, \beta_n[$.

1.2 Solveurs polynomiaux

L'étude des courbes et des surfaces algébriques fait appel à la résolution de systèmes polynomiaux. Par exemple, la recherche de singularité d'une courbe implicite donnée par un polynôme $f(x, y) \in \mathbb{R}[x, y]$ revient à résoudre le système $f(x, y) = \partial_x f(x, y) = \partial_y f(x, y) = 0$. La question de la résolution des systèmes polynomiaux est largement traitée dans [26]. On peut essentiellement distinguer deux types de solveurs à savoir ceux qui sont locaux (*e.g.* [39] et [49]) et ceux qui sont globaux (*e.g.* [38], [51] ou [64]).

Dans la mesure où les problèmes auxquels on s'intéresse sont locaux et réels, l'utilisation d'un solveur de systèmes polynomiaux, se contentant de chercher des solutions dans un domaine (réel) donné, nous sera d'une grande utilité. On donne ici une esquisse de l'élaboration d'un tel solveur utilisant la base de Bernstein. Pour compléter ce qui suit, on renvoie le lecteur à [49].

Le principe de base qui est utilisé est la subdivision. On cherche les solutions dans un domaine initial (borné) et on le subdivise en sous domaines. Un critère est alors utilisé pour exclure les sous-domaines ne contenant aucune solution et un paramètre $\epsilon > 0$ contrôle la taille des domaines générés (cela permet d'arrêter la subdivision et de donner une approximation des solutions relativement par rapport à la précision ϵ demandée). Par ailleurs, un autre critère d'isolation des solutions (*i.e.* qui certifie la présence d'une seule solution par domaine généré) est utilisé.

1.2.1 Solveur univarié

Soit $f(x) \in \mathbb{R}[x]$ un polynôme univarié et I un intervalle non vide et borné de $\mathbb{R}$. Il s'agit ici de donner une méthode pour trouver les racines de f, si elles existent, sur l'intervalle I. Dans la pratique, on peut se placer sur $\mathbb{Q}[x]$. Ainsi, on peut se ramener au problème qui consiste à chercher les racines d'un polynôme de $f(x) \in \mathbb{Z}[x]$. L'objectif d'un solveur univarié consiste à isoler les racines réelles de f dans I, *i.e.* de calculer des sous-intervalles de I ayant des extrémités rationnelles ne contenant chacun qu'une unique racine du polynôme. Cela suffit à donner une approximation des racines relativement par rapport à une tolérance fixée au départ. Le schéma général de fonctionnement d'un tel solveur est donné par l'algorithme 1.1. Ce dernier

nécessite une fonction $V(f, I)$ qui permet de borner le nombres de racines de f sur un intervalle I donné. Pour cela, on peut utiliser la règle de Descartes (propriété 1.2) *i.e.* on peut prendre pour $V(f, I)$ le nombre de changements de signes de la séquence formée par les coefficients de f écrit dans la base de Bernstein associée à I.

Algorithme 1.1 : Isolation de racines réelles

Données : Un polynôme univarié $f \in \mathbb{Z}[x]$ et un intervalle I non vide et borné.

Résultat : Une liste de sous intervalles de I ayant des extrémités rationnelles, ne contenant chacun qu'une unique racine de f.

Calculer la partie sans facteur carré de f ;
Initialiser une liste L d'intervalles avec I ;
tant que $L \neq \emptyset$ **faire**
    Retirer le premier élément J de L et calculer $v := V(f, J)$;
    si $v = 0$ **alors** rejeter J ;
    si $v = 1$ **alors** retourner J dans la liste des solutions ;
    si $v \geq 2$ **alors** subdiviser J en J_g et J_d et les rajouter à la liste L ;
fin

1.2.2 Solveur multivarié

Soient $n, s \in \mathbb{N}^*$ avec $s \leq n$ et $f_1, \ldots, f_s \in \mathbb{R}[x_1, \ldots, x_n]$. On considère le système polynomial suivant :

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ \vdots \\ f_s(x_1, \ldots, x_n) = 0 \end{cases} \tag{1.1}$$

On notera de manière concise $\mathbf{f}(\mathbf{x}) = 0$ avec $\mathbf{x} = x_1, \ldots, x_n$.

On cherche les solutions de ce système dans une boîte donnée $I := [\alpha_1, \beta_1] \times \cdots \times [\alpha_n, \beta_n] \subset \mathbb{R}^n$. La méthode d'approximation des solutions de ce système qu'on va décrire à présent utilise la représentation de Bernstein, le solveur univarié décrit dans la section 1.2.1 et des études de signes. L'objectif de ce solveur multivarié est de retourner une liste de sous boîtes de I suffisamment petites (relativement à une certaine tolérance fixée au départ)

contenant chacune une unique solution du système. Ce solveur par subdivision est détaillé dans [49, 55]. C'est une amélioration de l'algorithme *Interval Projected Polyhedron* de Sherbrooke et Patrikalakis qui est décrit dans [58].

Définition 1.4 : Soit $n \in \mathbb{N}^*$, $f(\mathbf{x}) \in \mathbb{R}[\mathbf{x}] = \mathbb{R}[x_1, \ldots, x_n]$ un polynôme de multidegré $(d_1, \ldots, d_n) \in (\mathbb{N}^*)^n$ et une boîte $I = [\alpha_1, \beta_1] \times \cdots \times [\alpha_n, \beta_n] \subset \mathbb{R}^n$. On suppose que l'écriture de f dans la base de Bernstein est la suivante :

$$f(x_1, \ldots, x_n) = \sum_{i_1=0}^{d_1} \cdots \sum_{i_n=0}^{d_n} b_{i_1,\ldots,i_n} \; B_{i_1}^{d_1}(x_1)_{[\alpha_1,\beta_1]} \cdots B_{i_n}^{d_n}(x_n)_{[\alpha_n,\beta_n]}.$$

Pour tout $j \in \{1, \ldots, n\}$, on pose :

$$\begin{aligned} m_j(f; x_j) &= \sum_{i_j=0}^{d_j} \min_{\{0 \le i_k \le d_k, k \ne j\}} b_{i_1,\ldots,i_n} \; B_{i_j}^{d_j}(x_j)_{[\alpha_j,\beta_j]} \\ M_j(f; x_j) &= \sum_{i_j=0}^{d_j} \max_{\{0 \le i_k \le d_k, k \ne j\}} b_{i_1,\ldots,i_n} \; B_{i_j}^{d_j}(x_j)_{[\alpha_j,\beta_j]} \end{aligned}$$

Lemme 1.1 (Lemme de projection) : Sous les mêmes hypothèses et notations que la définition 1.4, pour tout $\mathbf{u} = (u_1, \ldots, u_n) \in I$ et pour tout $j \in \{1, \ldots, n\}$, on a :

$$m_j(f; u_j) \le f(\mathbf{u}) \le M_j(f; u_j) \qquad \lozenge$$

Le lemme 1.1 implique alors une conséquence directe qui est le corollaire suivant.

Corollaire 1.2 : Toujours sous les mêmes hypothèses et notations que la définition 1.4, pour toute racine $\mathbf{u} = (u_1, \ldots, u_n) \in I$ du polynôme $f(\mathbf{x})$ et pour tout $j \in \{1, \ldots, n\}$, on a $\underline{\mu}_j \le u_j \le \overline{\mu}_j$ avec :

1. $\underline{\mu}_j$ (respectivement $\overline{\mu}_j$) qui est soit une racine de $m_j(f; x_j)$ (respectivement de $M_j(f; x_j)$) sur $[\alpha_j, \beta_j]$ soit égal à α_j (respectivement β_j) dans le cas où $m_j(f; x_j)$ (respectivement $M_j(f; x_j)$) n'a pas de racine dans $[\alpha_j, \beta_j]$,
2. $m_j(f; u) \le 0 \le M_j(f; u)$ pour tout $u \in [\underline{\mu}_j, \overline{\mu}_j]$.

Le solveur procède par étapes de la manière suivante : 1. appliquer une étape de pré-conditionnement aux équations, 2. réduire le domaine, 3. subdiviser le domaine jusqu'à ce qu'il soit de taille inférieure à une tolérance donnée.

Il nous faut donc introduire les fonctionnalités suivantes qui sont indispensables à l'algorithme.

Stratégie de pré-conditionnement. Il s'agit en fait de transformer le système polynomial initial 1.1 en un système ayant un « meilleur » comportement numérique. Résoudre le système $\mathbf{f} = 0$ est équivalent à résoudre le système $M\,\mathbf{f} = 0$, où M est une matrice inversible de taille $s \times s$.

Une telle transformation peut augmenter le degré (relativement par rapport à certaines variables) de certaines équations. Elle peut donc avoir un coût non négligeable dans certains cas. Par ailleurs, si dans chaque polynôme du système, toutes les variables n'apparaissent pas (système creux), alors un tel pré-conditionnement peut faire perdre au système son caractère creux. Dans ce cas, on préférera alors un pré-conditionnement partiel sur un sous-ensemble d'équations qui partagent toutes une partie des variables. On considère des transformations globales, qui réduisent au minimum la distance entre les équations vu comme des vecteurs d'un espace affine de polynômes, et également des redressement locaux (si $s = n$) qui transforment localement le système $\mathbf{f} = 0$ en le système $(J\mathbf{f}_\mathbf{u})^{-1}\mathbf{f} = 0$, où $J\mathbf{f}_\mathbf{u} = (\partial_{x_i} f_j(\mathbf{u}))_{1\leq i,j\leq s}$ est la matrice jacobienne de $\mathbf{f}$ en un point $\mathbf{u} \in I$ où cette matrice est inversible.

Stratégie de réduction. Il s'agit de la technique utilisée pour réduire le domaine initial I pour chercher les solutions du système. On peut utiliser la propriété d'enveloppe convexe comme dans [58] ou alors une localisation des solutions qui est une amélioration de la réduction par enveloppe convexe. Elle consiste à calculer la première (respectivement la dernière) racine du polynôme $m_j(f_k; x_j)$ (respectivement $M_j(f_k; x_j)$), dans l'intervalle $[\alpha_j, \beta_j]$ pour $j \in \{1, \ldots, n\}$ et $k \in \{1, \ldots, s\}$. L'implémentation de cette étape permet de considérer la réduction par enveloppe convexe comme une étape d'itération de ce procédé de réduction.

Stratégie de subdivision. Il s'agit de la technique utilisée pour subdiviser le domaine, afin de simplifier les étapes suivantes qui se chargent de chercher les solutions du système. Certaines règles simples peuvent être utilisées pour subdiviser un domaine et réduire sa taille. L'approche qui est adoptée ici est la bissection : un domaine $D = [\gamma_1, \delta_1] \times \cdots \times [\gamma_n, \delta_n]$ est découpé en deux dans une direction $j \in \{1, \ldots, n\}$ pour laquelle $|\delta_j - \gamma_j|$ est maximal. Mais au lieu de choisir la taille de l'intervalle pour critère de direction suivant laquelle on divise, on peut en choisir un autre en fonction aussi des valeurs de m_j, M_j ou f_j (par exemple quand $M_j - m_j$ est maximal).

Une borne pour la complexité de cette méthode est donnée dans [49].

1.3 Décomposition en valeurs singulières

Soient $m, n \in \mathbb{N}^*$, on note $\mathcal{M}_{m,n}(\mathbb{R})$ le $\mathbb{R}$-espace vectoriel des matrices à m ligne(s) et n colonne(s) à coefficients dans $\mathbb{R}$, si $m = n$ alors on note $\mathcal{M}_m(\mathbb{R})$ plutôt que $\mathcal{M}_{m,m}(\mathbb{R})$.

Définition 1.5 (Valeurs singulières) : Soit $A \in \mathcal{M}_{m,n}(\mathbb{R})$. Les valeurs propres de la matrice $A^T A \in \mathcal{M}_n(\mathbb{R})$ sont positives (car on peut voir $A^T A$ comme la matrice d'une forme quadratique définie positive). Les valeurs singulières de A sont par définition les racines carrées des valeurs propres de $A^T A$.

Remarque 1.4 : Dans la pratique, les valeurs singulières sont souvent notées $(\sigma_i)_{1 \leq i \leq n}$ avec $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$ (écriture avec répétitions éventuelles). •

Proposition 1.3 (Décomposition en valeurs singulières) : Soit $A \in \mathcal{M}_{m,n}(\mathbb{R})$ et $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$ les valeurs singulières de A. Alors on peut écrire A sous la forme $A = UDV^T$ avec $(U, V) \in \mathcal{M}_m(\mathbb{R}) \times \mathcal{M}_n(\mathbb{R})$ un couple de matrices orthogonales (*i.e.* $U^T U = I_m$ et $V^T V = I_n$) et $D \in \mathcal{M}_{m,n}(\mathbb{R})$ de la forme :

$$\begin{pmatrix} \sigma_1 & & (0) \\ & \ddots & \\ (0) & & \sigma_n \\ & (0) & \end{pmatrix} \text{ si } m > n\,; \quad \begin{pmatrix} \sigma_1 & & (0) \\ & \ddots & \\ (0) & & \sigma_n \end{pmatrix} \text{ si } m = n\,;$$

$$\begin{pmatrix} \sigma_1 & & (0) & \\ & \ddots & & (0) \\ (0) & & \sigma_n & \end{pmatrix} \text{ si } m < n.$$

Propriété 1.3 : Sous les mêmes hypothèses et notations que la proposition 1.3, on pose $k = \min(m, n)$. On note $(u_i)_{1 \leq i \leq k}$ (respectivement $(v_i)_{1 \leq i \leq k}$) les k premiers vecteurs colonne de U (respectivement de V). Alors, pour tout $i \in \{1, \ldots, k\}$, on a $Av_i = \sigma_i u_i$ et $A^T u_i = \sigma_i v_i$.

Dans la pratique, la matrice A représente un système linéaire surdéterminé (donc avec $m > n$). La solution du système qu'on considérera sera le dernier vecteur colonne de V (à savoir v_n). En effet, dans ce contexte on a, pour tout $i \in \{1, \ldots, n\}$, $Av_i = \sigma_i u_i$ et donc $\|Av_i\| = \sigma_i$ (car $\|u_i\| = 1$). Vu que σ_n est la plus petite valeur singulière de A, il s'en suit que v_n est la « meilleur » solution du système.

Chapitre 2

Approximation et localisation d'intersections

2.1 Introduction

Dans la mesure où, en CAO, les surfaces paramétrées sont utilisées pour délimiter des volumes, le calcul de la courbe d'intersection entre deux telles surfaces est primordial pour décrire les objets fabriqués dans ce contexte.

Dans l'idéal, l'approximation des surfaces ne doit pas être en marge du procédé d'intersection. Une alternative aux approches classiques par triangles ou B-splines (décrites dans l'introduction) consisterait à approximer les surfaces données par des maillages constitués d'entités élémentaires plus complexes que les triangles. Cela permet d'avoir des lieux d'intersection plus précis. Un choix intéressant consisterait à utiliser des approximations par des carreaux de surfaces de Bézier de petit degré. Dans ce cas de figure, il est alors essentiel de savoir intersecter efficacement de telles surfaces paramétrées polynomiales (c'est l'objet du chapitre 3).

Un exemple de maillage biquadratique. Soit $\mathcal{S}$ une surface paramétrée générale donnée par évaluations (*i.e.* une surface procédurale). On considère une grille de points sur $\mathcal{S}$ de taille $(2m+1) \times (2n+1)$ (avec m et n deux

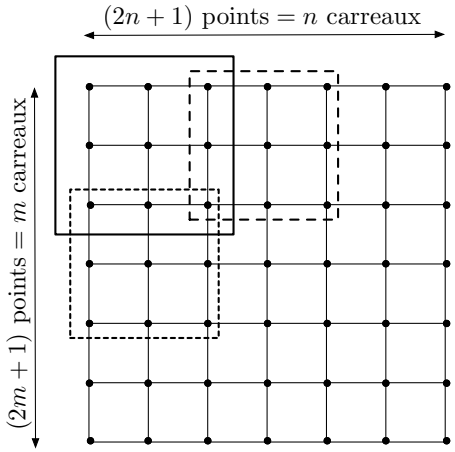


FIG. 2.1: Grille de carreaux biquadratiques.

entiers non nuls) permettant de construire une grille de carreaux de surfaces biquadratiques de taille $m \times n$. Sur la figure 2.1, on a représenté une telle grille de manière schématique.

La construction se fait de manière à ce que deux carreaux adjacents se recollent de façon continue. On a illustré sur la figure 2.2 un exemple d'approximation réalisée de cette manière. Il s'agit d'une forme (représentée à gauche sur la figure 2.2) composée de trois surfaces de type B-spline sur lesquelles on applique un offset. Une fois cette transformation appliquée, le nouvel objet ne peut être représenté par des B-splines. On applique alors une approximation de cet offset par une grille de 144 carreaux biquadratiques (le résultat est représenté à droite sur la figure 2.2). Pour mieux apprécier le résultat, on a également réalisé une image en coupe (figure 2.3). Un autre exemple est illustré sur la figure 2.4, il s'agit d'une surface B-spline sur laquelle on a appliqué plusieurs offsets dont les résultats sont approximés par des grilles de 128 carreaux biquadratiques.

Considérons maintenant deux grilles de carreaux de surfaces qui correspondent à deux surfaces procédurales $\mathcal{S}_1$ et $\mathcal{S}_2$ qu'on désire intersecter. Pour chacune de ces grilles, on construit une autre grille, de même taille, constituée de boîtes de $\mathbb{R}^3$ (*i.e.* de la forme $[a_1, b_1] \times [a_2, b_2] \times [a_3, b_3] \subset \mathbb{R}^3$)

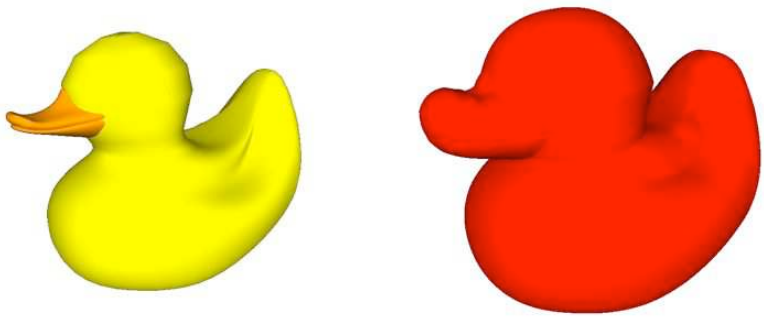


FIG. 2.2: Approximation d'un offset par une grille de 144 carreaux biquadratiques.

FIG. 2.3: Approximation d'un offset par une grille de 144 carreaux biquadratiques (vue en coupe).

construites à partir des points de contrôle de chacun des carreaux de surface et de manière à ce que chaque boîte contiennent entièrement le carreau correspondant (une description de cette construction est donnée dans la section 2.4.2). On construit un quadtree pour chacune de ces deux grilles de boîtes afin de rechercher toutes les paires de boîtes qui s'intersectent. Ces paires de boîtes correspondent alors à des paires de carreaux de surfaces qu'il faut intersecter (bien qu'il se peut que l'intersection soit vide).

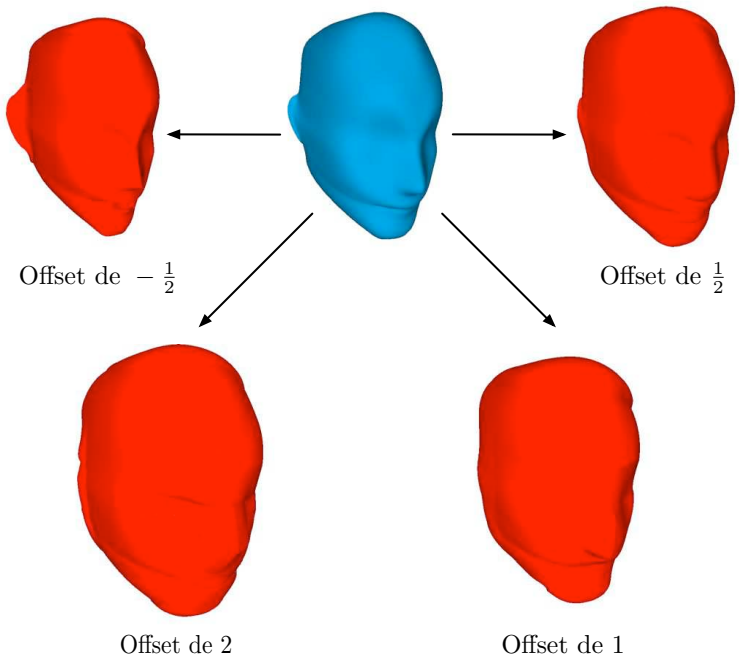


FIG. 2.4: Approximation de différents offsets par des grilles de 128 carreaux biquadratiques.

Les sections qui suivent détaillent les procédés d'approximation et de détection d'intersection probables.

2.2 Notations et représentations

Afin de clarifier le discours, on se propose d'introduire certaines notations générales ainsi qu'une représentation schématique des carreaux de surfaces biquadratiques.

2.2.1 Notations

1. Soient $\overrightarrow{v}$ et $\overrightarrow{w}$ deux vecteurs de $\mathbb{R}^2$ (respectivement de $\mathbb{R}^3$) muni de sa structure euclidienne canonique. On note $\overrightarrow{v} \cdot \overrightarrow{w}$ le produit scalaire de $\overrightarrow{v}$ et $\overrightarrow{w}$, et $\|\overrightarrow{v}\| = \sqrt{\overrightarrow{v} \cdot \overrightarrow{v}}$.

2. Soient $m, n \in \mathbb{N}^*$, on note $\mathcal{M}_{m,n}(\mathbb{R})$ l'ensemble des matrices à m ligne(s) et n colonne(s) à coefficients dans $\mathbb{R}$, si $m = n$ alors on note $\mathcal{M}_m(\mathbb{R})$ plutôt

que $\mathcal{M}_{m,m}(\mathbb{R})$. Soit $M \in \mathcal{M}_{m,n}(\mathbb{R})$, alors pour $1 \leq i \leq m$ et $1 \leq j \leq n$, on note $M[i,j]$ l'élément de M qui se situe à la $i^{\text{ème}}$ ligne et à la $j^{\text{ème}}$ colonne.

3. Soient $k = 2$ ou 3, $n \in \mathbb{N}^*$ et $x_1, \ldots, x_n \in \mathbb{R}^k$. Alors $L := [x_1, \ldots, x_n]$ représente la liste formée par les éléments $x_1, \ldots, x_n$. De plus, on notera $L[i] = x_i$ (pour $1 \leq i \leq n$) et $L[i..j] = x_i, \ldots, x_j$ (pour $1 \leq i < j \leq n$). On généralise cette notation pour des tableaux (listes de listes).

2.2.2 Polynôme de bidegré $(2,2)$ dans la base de Bernstein

Un polynôme $F \in \mathbb{R}[s,t]$ est de bidegré $(2,2)$ si F est de degré 2 en chacune des deux variables s et t. On choisira la base de Bernstein pour représenter les polynômes qui composent nos paramétrisations. Explicitement, se donner un polynôme $F \in \mathbb{R}[s,t]$ de bidegré $(2,2)$ exprimé dans la base de Bernstein revient à se donner une matrice $A \in \mathcal{M}_3(\mathbb{R})$ en posant :

$$F(s,t) = \sum_{i=0}^{2} \sum_{j=0}^{2} A[i+1, j+1] B_2^i(s) B_2^j(t)$$

avec $B_2^0(s) = (1-s)^2$, $B_2^1(s) = 2s(1-s)$ et $B_2^2(s) = s^2$ (ce sont les trois polynômes de Bernstein de degré 2).

Ainsi, se donner une surface polynomiale S de bidegré $(2,2)$ paramétrée sur $[0,1]^2$ (qu'on appellera carreau de surface biquadratique), revient à se donner trois matrices X, Y et $Z \in \mathcal{M}_3(\mathbb{R})$ en posant :

$$\forall (s,t) \in [0,1]^2, S(s,t) = \sum_{i=0}^{2} \sum_{j=0}^{2} P_{i,j} B_2^i(s) B_2^j(t)$$

avec $\forall (i,j) \in \{0,1,2\}^2, P_{i,j} = \begin{pmatrix} X[i+1, j+1] \\ Y[i+1, j+1] \\ Z[i+1, j+1] \end{pmatrix}$.
Ce sont les points de contrôle de S.

Remarque 2.1 : Le graphe dont les sommets sont les points $(P_{i,j})_{0 \leq i,j \leq 2}$ et dont les arêtes sont les segments de la forme $([P_{i,j}, P_{i+1,j}])$ et $([P_{i,j}, P_{i,j+1}])$ est appelé le polyèdre de contrôle de S. Ainsi, on peut représenter schématiquement les surfaces biquadratiques comme sur la figure 2.5. •

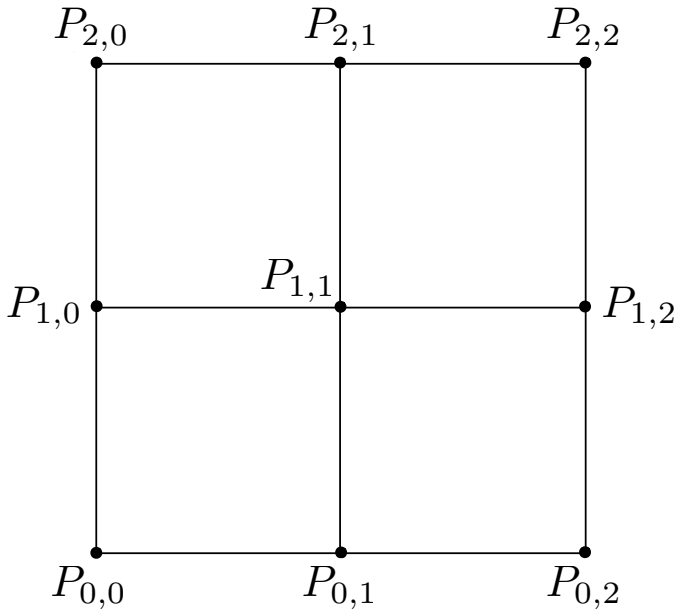


FIG. 2.5: Schéma d'un polyèdre de contrôle pour un carreau de surface biquadratique.

Proposition 2.1 : Le carreaux de surface $S([0,1]^2)$ est entièrement contenue dans l'enveloppe convexe de la famille de points $(P_{i,j})_{0\leq i,j\leq 2}$.

Démonstration : C'est une conséquence directe de la proposition 1.1. □

2.3 Interpolation

Dans cette section, on décrit plus en détails la méthode d'interpolation (locale) utilisée pour l'approximation (globale).

Etant donnée une surface procédurale $\mathcal{S}(u,v)$ définie sur $[a,b]\times[c,d]\subset\mathbb{R}^2$. On met en place une approximation de $\mathcal{S}$ par une grille $m\times n$ de carreaux de surfaces biquadratiques (avec m et n deux entiers non nuls). On échantillonne les paramètres (u,v) en une famille $(u_i,v_j)_{i,j}$ pour $0\leq i\leq 2m$

et $0 \leq j \leq 2n$ définie par :

$$\begin{aligned}
u_0 &= a \\
u_i &= u_0 + i\,\delta_u \text{ pour } i \in \{1, \ldots, 2m\} \\
v_0 &= c \\
v_j &= v_0 + j\,\delta_v \text{ pour } j \in \{1, \ldots, 2n\} \\
\delta_u &= \frac{b-a}{2m} \\
\delta_v &= \frac{d-c}{2n}.
\end{aligned}$$

La famille $(u_i, v_j)_{i,j}$ donne une famille de $(2m+1)(2n+1)$ points $(\mathcal{S}(u_i, v_j))_{i,j}$ de $\mathbb{R}^3$. Cette famille de points donne une famille de mn carreaux de surfaces biquadratiques $(S_{k,l})_{k,l}$ avec pour tout $k \in \{0, \ldots, m-1\}$ et tout $l \in \{0, \ldots, n-1\}$, $S_{k,l}$ interpolée par la famille de points $(\mathcal{S}(u_i, v_j))_{i,j}$ avec $i \in \{2k, 2k+1, 2k+2\}$ et $j \in \{2l, 2l+1, 2l+2\}$.

Plus précisément, pour tout $(k, l) \in \{0, \ldots, m-1\} \times \{0, \ldots, n-1\}$, on a :

$$S_{k,l} : \left(\begin{array}{ccc} [u_{2k}, u_{2k+2}] \times [v_{2l}, v_{2l+2}] & \longrightarrow & \mathbb{R}^3 \\ (u_i, v_j) & \longmapsto & \mathcal{S}(u_i, v_j) \end{array} \right)$$

$$\forall (i, j) \in \{2k, 2k+1, 2k+2\} \times \{2l, 2l+1, 2l+2\}.$$

Dans la mesure où la construction qui précède ne fait qu'interpoler des points, elle ne permet pas d'assurer la continuité entre chaque carreau de surface. Ainsi, on ajuste chacun d'entre eux de manière à ce qu'il partage les mêmes points de contrôle que son voisin sur le bord concerné. Concrètement, il ne faut ajuster qu'un seul point de contrôle par bord. On a illustré schématiquement sur la figure 2.6 une situation de deux carreaux (un en bleu et un en rouge) adjacents construits par interpolation et représentés par leurs points de contrôle respectifs. Pour imposer le recollement, on modifie un point de contrôle sur les bords correspondants. Cette manipulation est effectuée sur tous les carreaux construits par interpolation et on continuera à les noter $(S_{k,l})_{k,l}$. Cela permet d'avoir une approximation de la surface procédurale $\mathcal{S}$.

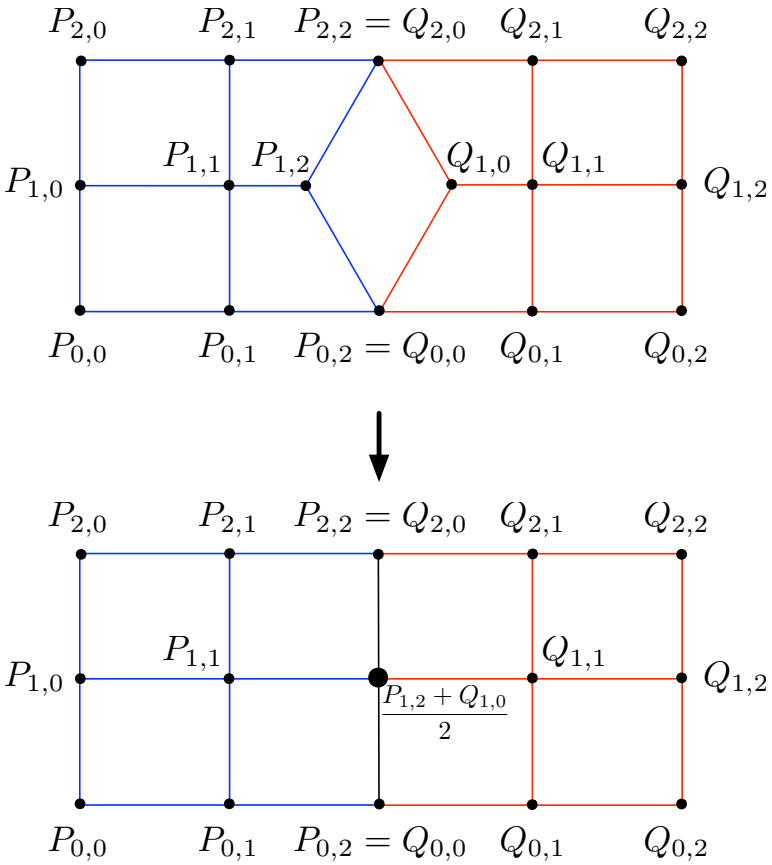


FIG. 2.6: Recollement de deux carreaux adjacents.

2.4 Boîtes englobantes et raffinement

Les boîtes englobantes, qu'on définira dans la section 2.4.2, nous seront utiles pour mettre en place un critère qui permet d'exclure certaines configurations de non intersection entre deux carreaux de surfaces. On donnera, dans la section 2.4.3.2, un algorithme détaillé de ce procédé qui se base sur l'intersection de ces boîtes ainsi que sur l'algorithme de De Casteljau.

2.4.1 Algorithme de De Casteljau

2.4.1.1 Cas d'une courbe de Bézier de degré 2

Soient $k = 2$ ou 3, P_0, P_1 et $P_2 \in \mathbb{R}^k$ et $\mathcal{C} : \begin{pmatrix} [0,1] & \longrightarrow & \mathbb{R}^k \\ t & \longmapsto & \sum_{i=0}^{2} P_i B_2^i(t) \end{pmatrix}$ une courbe de Bézier de degré 2.

Propriété 2.1 : Soit $t_0 \in]0,1[$. On note $P_{0,1}$ (respectivement $P_{1,2}$) le barycentre des deux points pondérés $(P_0, 1-t_0)$ et (P_1, t_0) (respectivement $(P_1, 1-t_0)$ et (P_2, t_0)). Alors $P_0, P_{0,1}$ et $\mathcal{C}(t_0)$ (respectivement $\mathcal{C}(t_0), P_{1,2}$ et P_2) sont les points de contrôle de la courbe $\mathcal{C}([0,t_0])$ (respectivement $\mathcal{C}([t_0,1])$) (voir la figure 2.7).

Pour des raisons de commodité, on notera par la suite :

$$\text{Casteljau}(P_0, P_1, P_2, t_0) := [P_0, P_{0,1}, \mathcal{C}(t_0), P_{1,2}, P_2]$$

Cette construction, que l'on peut appliquer récursivement, permet de subdiviser la courbe. Il faut également remarquer que plus on subdivise plus les points de contrôle se rapprochent de la courbe initiale.

2.4.1.2 Cas d'une surface de Bézier de bidegré $(2,2)$

On considère une famille de points de $\mathbb{R}^3$ $(P_{i,j})_{0 \leq i,j \leq 2}$ et une surface de Bézier de bidegré $(2,2)$ (carreau de surface biquadratique) :

$$S : \begin{pmatrix} [0,1]^2 & \longrightarrow & \mathbb{R}^3 \\ (s,t) & \longmapsto & \sum_{i=0}^{2}\sum_{j=0}^{2} P_{i,j} B_2^i(s) B_2^j(t) \end{pmatrix}.$$

Propriété 2.2 : Soit $(s_0, t_0) \in]0,1[^2$, on pose :

$$\begin{cases} \forall i \in \{0,1,2\}, q_i := \text{Casteljau}(P_{i,0}, P_{i,1}, P_{i,2}, s_0) \\ \forall j \in \{1,\ldots,5\}, p_j := \text{Casteljau}(q_0[j], q_1[j], q_2[j], t_0) \end{cases}. \tag{2.1}$$

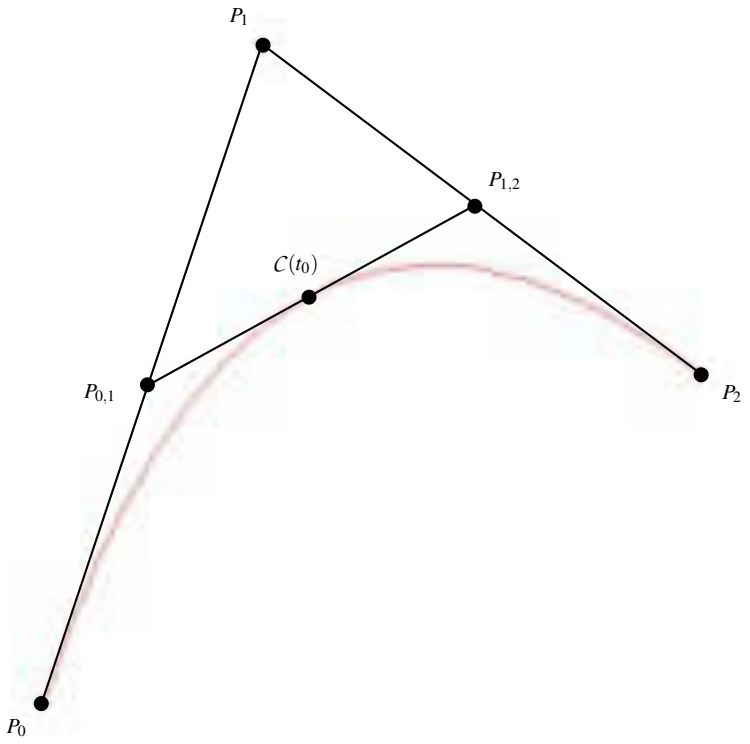


FIG. 2.7: Algorithme de De Casteljau sur une courbe de Bézier de degré 2.

Ainsi, chaque q_i et chaque p_j est une liste de 5 points de $\mathbb{R}^3$ (voir la construction sur la figure 2.8). On construit alors un tableau M de dimension 5×5 de points de $\mathbb{R}^3$ de la manière suivante : $\forall j, k \in \{1, \ldots, 5\}, M[j,k] := p_j[k]$. Alors :

– $M[1..3, 1..3]$ sont les points de contrôle de $S([0, s_0] \times [0, t_0])$.
– $M[1..3, 3..5]$ sont les points de contrôle de $S([0, s_0] \times [t_0, 1])$.
– $M[3..5, 1..3]$ sont les points de contrôle de $S([s_0, 1] \times [0, t_0])$.
– $M[3..5, 3..5]$ sont les points de contrôle de $S([s_0, 1] \times [t_0, 1])$.

Rappelons que $M[1..3, 1..3], M[1..3, 3..5], M[3..5, 1..3]$ et $M[3..5, 3..5]$ sont des sous tableaux de M de dimension 3×3.

Comme dans le cas des courbes, cette construction peut être appliquée récursivement pour subdiviser la surface (plus on subdivise et plus les points

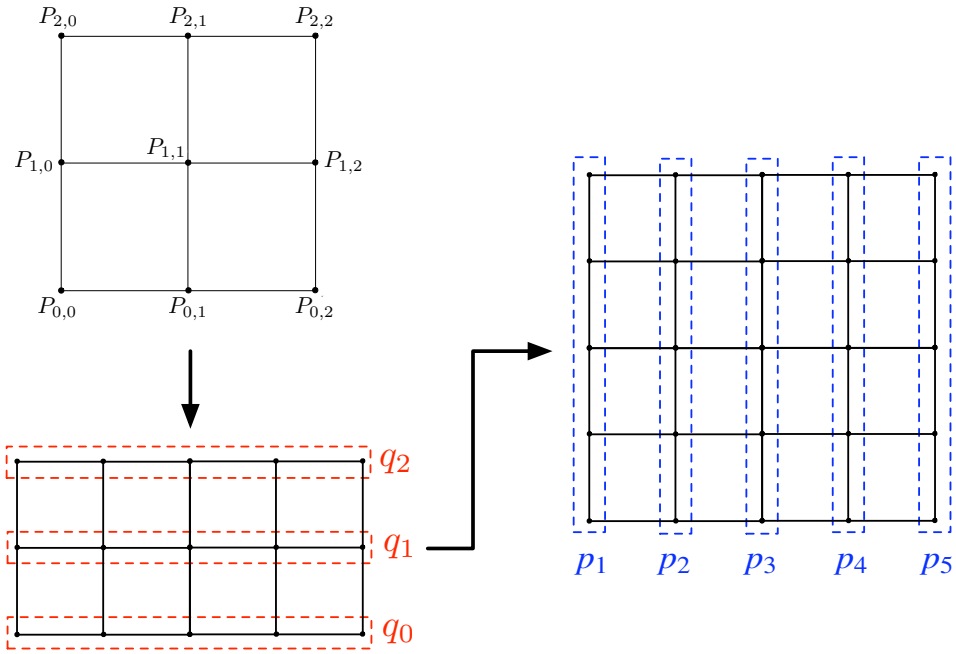


FIG. 2.8: Algorithme de De Casteljau sur un carreau de surface biquadratique.

de contrôle se rapprochent de la surface initiale).

Remarque 2.2 : Dans (2.1), on peut permuter les rôles de s_0 et t_0. •

2.4.2 Boîtes englobantes orientées et intersections

Dans cette section on définit les boîtes englobantes orientées suivant des directions données ainsi qu'un critère d'intersection de telles boîtes.

2.4.2.1 Cas du plan

Définition 2.1 : Soient $p_1, \ldots, p_k$ (avec $k \geq 3$) des points de $\mathbb{R}^2$ non alignés et $(\overrightarrow{v_1}, \ldots, \overrightarrow{v_l})$, (avec $l \geq 2$) une famille génératrice de vecteurs de $\mathbb{R}^2$. Pour tout $i \in \{1, \ldots, l\}$, on note : $m_i = \min_{1 \leq j \leq k} (\overrightarrow{v_i} \cdot \overrightarrow{p_j})$ et $M_i = \max_{1 \leq j \leq k} (\overrightarrow{v_i} \cdot \overrightarrow{p_j})$. Alors on appellera boîte englobante des points $(p_j)_{1 \leq j \leq k}$ orientée par les vecteurs $(v_i)_{1 \leq i \leq l}$ l'ensemble :

$$\{p \in \mathbb{R}^2 / \forall i \in \{1, \ldots, k\}, m_i \leq (\overrightarrow{v_i} \cdot \overrightarrow{p}) \leq M_i\}$$

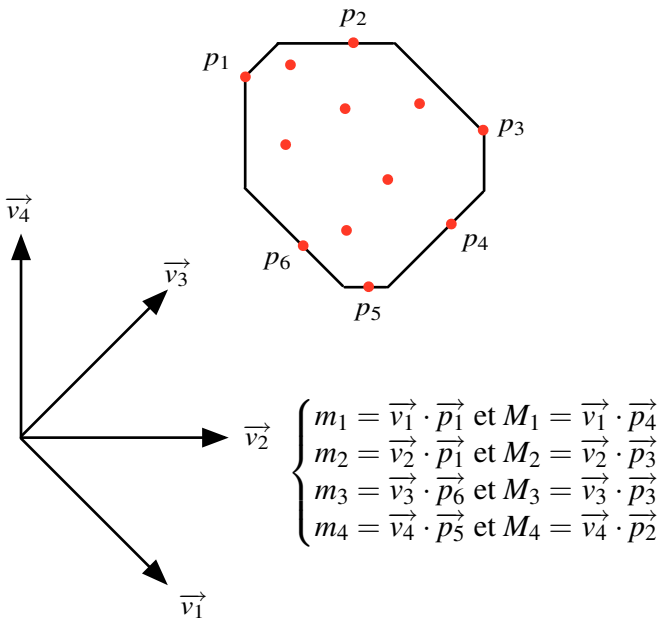


FIG. 2.9: Exemple de boîte englobante dans le plan.

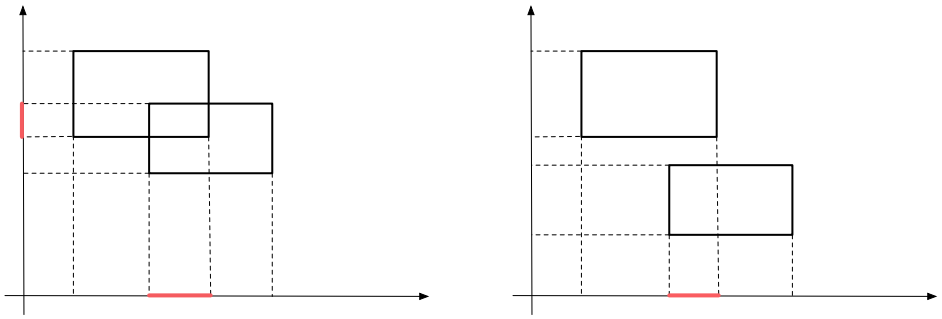
Sur la figure 2.9, on a pris un exemple de boite englobante de 12 points orientée par 4 vecteurs.

Propriété 2.3 (Intersection) : Soient $(p_i)_{1\leq i\leq k}$ et $(q_j)_{1\leq j\leq l}$ deux familles de points non alignés de $\mathbb{R}^2$ (avec k et $l \geq 3$) et $(\overrightarrow{v_1},\ldots,\overrightarrow{v_n})$, (avec $n \geq 2$) une famille génératrice de vecteurs de $\mathbb{R}^2$. On pose :

$$\forall i \in \{1,\ldots,n\}, \begin{cases} I_i := \left[\min\limits_{1\leq j\leq k}(\overrightarrow{v_i}\cdot\overrightarrow{p_j}), \max\limits_{1\leq j\leq k}(\overrightarrow{v_i}\cdot\overrightarrow{p_j})\right] \\ J_i := \left[\min\limits_{1\leq j\leq l}(\overrightarrow{v_i}\cdot\overrightarrow{q_j}), \max\limits_{1\leq j\leq l}(\overrightarrow{v_i}\cdot\overrightarrow{q_j})\right] \end{cases}.$$

Si les deux boîtes englobantes des points $(p_i)_{1\leq i\leq k}$ et $(q_j)_{1\leq j\leq l}$ orientées par les vecteurs $\overrightarrow{v_1},\ldots,\overrightarrow{v_n}$ s'intersectent, alors pour tout $i \in \{1,\ldots,n\}$, $I_i \cap J_i \neq \emptyset$.

On a illustré sur la figure 2.10 le principe de ce test d'intersection de deux boîtes.



(a) Cas avec intersection. (b) Cas sans intersection.

FIG. 2.10: Intersection de deux boîtes.

2.4.2.2 Cas de l'espace

La notion de boîte englobante dans $\mathbb{R}^3$ est similaire au cas du plan. Il suffit de remplacer, dans la section 2.4.2.1, $\mathbb{R}^2$ par $\mathbb{R}^3$, « non alignés » par « non coplanaires » et enfin le nombre de points à considérer doit être au moins de 4 et celui de vecteurs au moins de 3. De plus, le critère d'intersection de deux boîtes englobantes reste le même que celui qui précède (si on fait les changements de contexte comme il a été dit).

2.4.3 Critère de discrimination

Afin d'éviter un maximum de calculs d'intersection entre deux carreaux biquadratiques, il semble judicieux de mettre en place un critère de discrimination pour éliminer rapidement les cas où il n'y a aucune intersection. Soient $S_1 : \begin{pmatrix} [0,1]^2 & \longrightarrow & \mathbb{R}^3 \\ (s,t) & \longmapsto & S_1(s,t) \end{pmatrix}$ et $S_2 : \begin{pmatrix} [0,1]^2 & \longrightarrow & \mathbb{R}^3 \\ (u,v) & \longmapsto & S_2(u,v) \end{pmatrix}$, deux carreaux biquadratiques *i.e.* donnés par des polynômes de bidegrés $(2,2)$ exprimés dans la base de Bernstein. On notera abusivement $S_1 = S_1([0,1]^2)$ et $S_2 = S_2([0,1]^2)$. Notons également $(P_{i,j})_{0\leq i,j\leq 2}$ les points de contrôle de S_1 et $(Q_{i,j})_{0\leq i,j\leq 2}$ ceux de S_2.

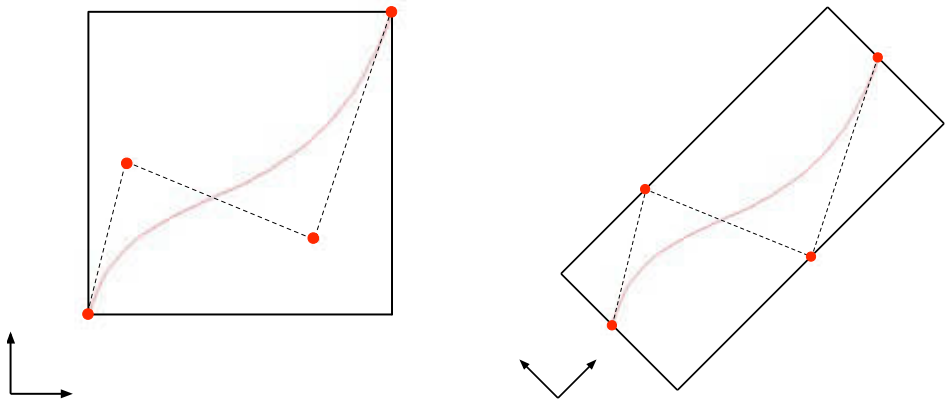
Pour élaborer un critère de discrimination, on « enferme » chaque carreau de surface dans une boite, si les deux boîtes ne s'intersectent pas, alors il en est de même pour S_1 et S_2. Pour se faire, on crée une boite englobante pour les points de contrôle de S_1 et une pour ceux de S_2. Il faut par conséquent

faire un choix de directions pour ces boites englobantes.

2.4.3.1 Choix des directions

Les directions permettent de créer des boîtes englobantes plus ou moins fines pour enfermer chaque carreau de surface (voir la figure 2.11).

Pour choisir les directions des boîtes englobantes, on peut adopter deux types de stratégies. Soit on fixe arbitrairement dès le départ un certain nombre de directions (comme sur la figure 2.12). Soit on fait ce choix judicieusement en fonction de S_1 et S_2 (comme sur le schéma de droite de la figure 2.11) en calculant par exemple des directions principales. La première méthode permet de gagner du temps au niveau du calcul mais ne nous donne aucun contrôle au niveau de la géométrie des objets considérés (choix arbitraire) même s'il peut arriver que les boîtes obtenues soient de bonne qualité (cela dépend évidemment du nombre de directions choisies). En revanche, la seconde demande des calculs préliminaires mais offre un filtrage globalement plus fin. Vu que l'on fera des découpages récursifs de boîtes (voir la section 2.4.3.2), un calcul systématique de directions adaptées à chaque étape peut s'avérer trop coûteux. Toutefois une telle approche serait envisageable si ces calculs de directions adaptées à chaque itération sont réutilisés (par exemple dans le cas où l'on garde ce genre d'information pour traiter l'intersection



(a) Premier choix de directions. (b) Second choix de directions.

FIG. 2.11: Boites englobantes orientées de manières différentes pour une courbe de Bézier.

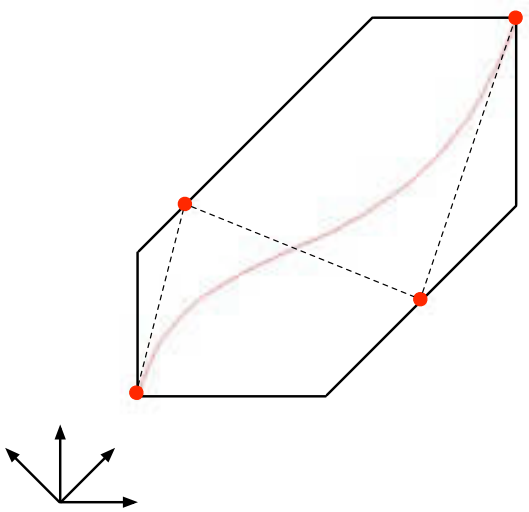


FIG. 2.12: Boite englobante avec un choix arbitraire de directions.

d'une famille de carreaux de surface).

Ainsi, pour traiter intrinsèquement le problème d'intersection de deux carreaux de surfaces, on se contentera plutôt d'adopter la première approche pour ce qui est du choix des directions. On peut par exemple prendre les directions données par les vecteurs : $(1,0,0)$, $(0,1,0)$, $(0,0,1)$, $(1,-1,0)$, $(1,1,0)$, $(1,0,-1)$, $(1,0,1)$, $(0,1,-1)$ et $(0,1,1)$.

Pour savoir si S_1 et S_2 s'intersectent, on peut donc commencer par se demander si les deux boites englobantes des points $(P_{i,j})_{0\leq i,j\leq 2}$ et $(Q_{i,j})_{0\leq i,j\leq 2}$ suivant les directions choisies s'intersectent. Si ce n'est pas le cas, alors la proposition 2.1 permet d'affirmer que $S_1 \cap S_2 = \emptyset$. En revanche, si c'est le cas, cette approximation peut s'avérer trop grossière et donc il peut arriver que le filtrage soit peu efficace dans la mesure où l'on peut rencontrer beaucoup de configurations où les deux boîtes s'intersectent sans qu'il en soit de même pour les carreaux de surfaces correspondants. C'est pourquoi l'algorithme de De Casteljau est d'une grande utilité car il permet précisément « d'affiner les boîtes ». C'est ce qui est expliqué dans la section suivante.

2.4.3.2 Découpage récursif de boîtes

Il s'agit ici de donner une construction pour affiner au fur et à mesure les boîtes décrites précédemment en utilisant l'algorithme de De Casteljau. Pour résumer la situation, on dispose au départ de deux boîtes englobantes B_1 et B_2, si elles sont disjointes, alors on a terminé (et on conclut que $S_1 \cap S_2 = \emptyset$) sinon on peut découper chaque boîte en sous boîtes (plus fines) et recommencer le test d'intersection sur ces sous boîtes et ainsi de suite. Pour ce faire, il nous faut un critère d'arrêt et une stratégie de découpe. Concrètement, au bout d'un certain degré de découpage, si les boîtes continuent à s'intersecter, alors on arrête cette manipulation (il faudra traiter le problème de l'intersection car visiblement il sera alors fort probable que $S_1 \cap S_2 \neq \emptyset$). Pour décrire l'algorithme principal (pour affiner les boîtes de manière récursive), on introduit quelques hypothèses et procédures préliminaires.

Directions des boîtes englobantes : notons $(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$ les directions des boîtes englobantes (choisies dès le départ et fixées une bonne fois pour toute). Pour simplifier (notamment pour le test d'arrêt), on supposera que ces directions sont unitaires.

Informations contenues dans une boîte : on suppose que pour les boîtes englobantes, on a fait un choix de structure de données qui contient les points de contrôle $(P_{i,j})_{0 \leq i,j \leq 2}$, l'espace des paramètres $[a, b] \times [c, d] \subset [0, 1]^2$ du carreau de surface correspondant à ces points de contrôle et également les intervalles $I_k = \left[\min_{0 \leq i,j \leq 2} (\overrightarrow{v_k} \cdot \overrightarrow{P_{i,j}}), \max_{0 \leq i,j \leq 2} (\overrightarrow{v_k} \cdot \overrightarrow{P_{i,j}}) \right]$ pour $k \in \{1, \ldots, n\}$.

Test d'arrêt : on doit disposer d'une procédure d'arrêt `arret`(B_1, B_2, ϵ) (avec $\epsilon > 0$) à valeurs booléennes qui nous informe si les boîtes passées en paramètre sont suffisamment fines (si tel est le cas, cela signifie que ce n'est plus la peine de continuer à subdiviser). On peut par exemple utiliser une des deux fonction de taille décrites ci dessous.

Fonctions de taille : on dispose de deux fonctions de tailles $\texttt{taille}_h$ et $\texttt{taille}_v$. Etant donné une boite B correspondante à un espace de paramètres $[a, b] \times [c, d]$, $\texttt{taille}_h(B) := b - a$ et $\texttt{taille}_v(B) := d - c$.

Les découpes d'une boîte : si B est une boîte contenant les points de contrôle $(P_{i,j})_{0\leq i,j\leq 2}$ et l'espace des paramètres $[a,b]\times[c,d]$, alors on suppose que l'on dispose de $(s_0,t_0)\in]a,b[\times]c,d[$ (on peut par exemple prendre $s_0=\frac{b-a}{2}$ et $t_0=\frac{d-c}{2}$). On définit alors (voir la figure 2.13) : $\texttt{gauche}(B)$, la boîte composée par $(\text{Casteljau}(P_{i,0},P_{i,1},P_{i,2},s_0)[1..3])_{0\leq i\leq 2}$
$\texttt{droite}(B)$, la boîte composée par $(\text{Casteljau}(P_{i,0},P_{i,1},P_{i,2},s_0)[3..5])_{0\leq i\leq 2}$
$\texttt{haut}(B)$, la boîte composée par $(\text{Casteljau}(P_{0,j},P_{1,j},P_{2,j},t_0)[1..3])_{0\leq j\leq 2}$
$\texttt{bas}(B)$, la boîte composée par $(\text{Casteljau}(P_{0,j},P_{1,j},P_{2,j},t_0)[3..5])_{0\leq j\leq 2}$.

Signalons qu'au niveau des structures de données, $\texttt{gauche}(B)$, $\texttt{droite}(B)$, $\texttt{haut}(B)$ et $\texttt{bas}(B)$ contiennent les espaces de paramètres respectifs $[a,s_0]\times[c,d]$, $[s_0,b]\times[c,d]$, $[a,b]\times[c,t_0]$ et $[a,b]\times[t_0,d]$.

Tout cela permet d'introduire l'algorithme 2.1 qui teste de manière récursive si deux carreaux biquadratiques sont grossièrement disjoints.

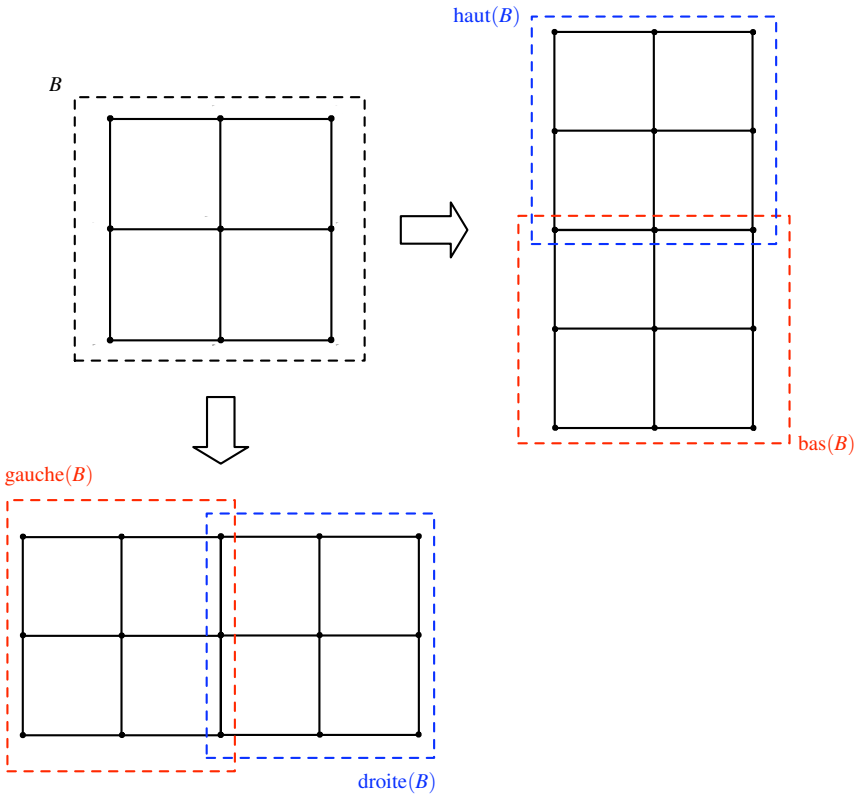


FIG. 2.13: Découpes d'une boîte.

Algorithme 2.1 : Algorithme de test par découpages récursifs.

`testIntersection`(B_1, B_2, ϵ, k)
Données : B_1 et B_2 les deux boîtes englobantes de S_1 et S_2, $\epsilon > 0$ (qui contrôle la finesse des boîtes) et $k \in \{0, 1\}$ (artifice propre à l'algorithme).
Résultat : Un booléen (soit $S_1 \cap S_2 = \emptyset$ soit il faut traiter l'intersection $S_1 \cap S_2$).

```
k := k + 1[mod 2];
si B1 ∩ B2 ≠ ∅ alors
    si arret(B1, B2, ε) alors
        On s'arrête (il faut traiter l'intersection);
    si k = 0 alors
        si taille_h(B1) ≥ taille_h(B2) alors
            testIntersection(gauche(B1), B2, ε, k);
            testIntersection(droite(B1), B2, ε, k);
        sinon
            testIntersection(B1, gauche(B2), ε, k);
            testIntersection(B1, droite(B2), ε, k);
        fin
    sinon
        si taille_h(B1) ≥ taille_h(B2) alors
            testIntersection(haut(B1), B2, ε, k);
            testIntersection(bas(B1), B2, ε, k);
        sinon
            testIntersection(B1, haut(B2), ε, k);
            testIntersection(B1, bas(B2), ε, k);
        fin
    fin
sinon
    Les deux boîtes ne s'intersectent pas et donc S1 ∩ S2 = ∅.
fin
```

2.5 Intersection de deux grilles de boîtes englobantes

Etant donné que, dans le problème d'intersection de deux surfaces procédurales, on met en place une approximation de chacune d'elles, il nous faut intersecter efficacement deux grilles de boîtes englobante. On propose dans cette section un algorithme pour traiter ce type d'intersection.

2.5.1 Structure de données

Soit $\mathcal{S}$ une surface procédurale et $m, n \in \mathbb{N}^*$, la famille de mn carreaux de surfaces $(S_{k,l})_{k,l}$ construite par le procédé décrit dans la section 2.3 permet de définir une famille $\mathcal{B}$ de mn boîtes englobantes $(B_{k,l})_{k,l}$ correspondantes (voir la section 2.4.2). Afin de simplifier les structures de données et l'algorithme de détection d'intersection de boîtes, on suppose que $m = n = 2^p$ pour un certain $p \in \mathbb{N}^*$ (voir la figure 2.14). Si tel est le cas, alors on peut regrouper les boîtes par paquets de quatre boîtes adjacentes (cela permettra d'accélérer l'algorithme de la section 2.5.2).
En effet, pour tout $(r, s) \in \{0, \dots, 2^{p-1} - 1\}^2$, les boîtes $B_{2r,2s}$, $B_{2r+1,2s}$, $B_{2r,2s+1}$ et $B_{2r+1,2s+1}$ sont adjacentes. On construit alors, par les feuilles, un quadtree de la manière suivante (voir l'illustration sur la figure 2.15) :

1. Les feuilles sont constituées par les mn boîtes $(B_{k,l})_{k,l}$.

2. Pour tout $(r, s) \in \{0, \dots, 2^{p-1} - 1\}^2$, on construit le père $\tilde{B}_{r,s}$ de $B_{2r,2s}$, $B_{2r+1,2s}$, $B_{2r,2s+1}$ et $B_{2r+1,2s+1}$ comme étant la plus petite boîte contenant l'union de ces quatre boîtes (que l'on appellera, de manière plus concise, l'union de $B_{2r,2s}$, $B_{2r+1,2s}$, $B_{2r,2s+1}$ et $B_{2r+1,2s+1}$).

3. On réitère le processus précédent en prenant l'union de $\tilde{B}_{2r,2s}$, $\tilde{B}_{2r+1,2s}$, $\tilde{B}_{2r,2s+1}$ et $\tilde{B}_{2r+1,2s+1}$ jusqu'à arriver à la racine de l'arbre (cette racine est

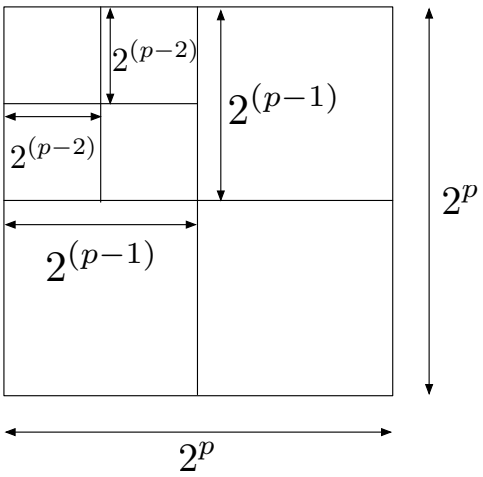


FIG. 2.14: Grille de boîte de taille $2^p \times 2^p$.

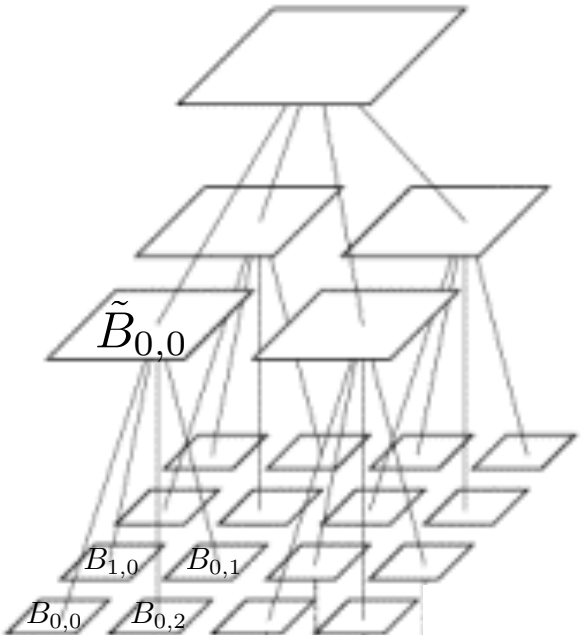


FIG. 2.15: Arbre de boîtes.

alors une boîte qui contient toutes les boites englobantes initiales).

2.5.2 Intersection

Supposons maintenant que l'on ait deux surfaces procédurales $\mathcal{S}_1$ et $\mathcal{S}_2$ qu'on désire intersecter. La première étape consiste à faire une approximation de $\mathcal{S}_1$ (respectivement $\mathcal{S}_2$) par une grille de $m_1 \times n_1$ (respectivement $m_2 \times n_2$) carreaux de surfaces biquadratiques construits par le procédé décrit dans la section 2.3. Cela permet alors de définir une famille $\mathcal{B}_1$ (respectivement $\mathcal{B}_2$) de $m_1 n_1$ (respectivement $m_2 n_2$) boîtes englobantes correspondantes. Afin d'exclure toutes les situations de non intersection grossières (*i.e.* les cas où les boîtes englobantes sont disjointes) entre carreaux de surfaces de $\mathcal{S}_1$ et $\mathcal{S}_2$, on commence par détecter toutes les configurations où $B_1 \cap B_2 \neq \emptyset$ pour $B_1 \in \mathcal{B}_1$ et $B_2 \in \mathcal{B}_2$. Pour cela, on construit deux arbres $\mathcal{A}_1$ et $\mathcal{A}_2$ correspondants respectivement à $\mathcal{B}_1$ et $\mathcal{B}_2$ (comme décrit dans la section

2.5.1) et on utilise l'algorithme de détection d'intersection 2.2 sur les racines respectives de $\mathcal{A}_1$ et $\mathcal{A}_2$. Pour un noeud N d'un arbre, on notera `fils(0,`N`)`, `fils(1,`N`)`, `fils(2,`N`)` et `fils(3,`N`)` les quatre fils de N.

Algorithme 2.2 : Détection d'intersections de boîtes.

`intersection(`N_1, N_2`)`

Données : Deux noeuds N_1 et N_2 des quadtrees respectifs $\mathcal{A}_1$ et $\mathcal{A}_2$ ainsi qu'une liste vide de paires de noeuds $\mathcal{L}$.

Résultat : La liste $\mathcal{L}$ contient les paires de boites (B_1, B_2) avec $B_1 \in \mathcal{B}_1$ et $B_2 \in \mathcal{B}_2$ qui s'intersectent.

si $N_1 \cap N_2 = \emptyset$ **alors**
| **retourner** $\mathcal{L}$;
fin
si N_1 *et* N_2 *sont des feuilles* **alors**
| **si** $N_1 \cap N_2 \neq \emptyset$ **alors**
| | $\mathcal{L} \longleftarrow (N_1, N_2)$;
| **fin**
sinon si N_1 *est une feuille mais pas* N_2 **alors**
| **pour** $j \in \{0,1,2,3\}$ **faire**
| | $\mathcal{L} \longleftarrow$ `intersection(`N_1, `fils(`j, N_2`))` ;
| **fin**
sinon si N_2 *est une feuille mais pas* N_1 **alors**
| **pour** $i \in \{0,1,2,3\}$ **faire**
| | $\mathcal{L} \longleftarrow$ `intersection(fils(`i, N_1`)`, N_2`)` ;
| **fin**
sinon
| **pour** $(i,j) \in \{0,1,2,3\}^2$ **faire**
| | $\mathcal{L} \longleftarrow$ `intersection(fils(`i, N_1`)`, `fils(`j, N_2`))` ;
| **fin**
fin
retourner $\mathcal{L}$;

La détection des boites englobantes qui s'intersectent combinée au raffinement décrit dans la section 2.4 permettent de résumer le problème d'intersection entre deux surfaces procédurale à une famille de problèmes d'intersection entre deux carreaux de surfaces polynomiales qui ne sont pas grossièrement disjointes. Cette question est traitée dans le chapitre 3.

Chapitre 3

Intersection des surfaces polynomiales

Ce chapitre traite la question de l'intersection de deux surfaces parmamétrées polynomiales que l'on supposera, sans restreindre la généralité, paramétrées sur $[0,1]^2$. Sauf mention explicite du contraire, on notera donc :

$$F : \begin{pmatrix} [0,1]^2 & \longrightarrow & \mathbb{R}^3 \\ (s,t) & \longmapsto & F(s,t) \end{pmatrix} \tag{3.1}$$

$$G : \begin{pmatrix} [0,1]^2 & \longrightarrow & \mathbb{R}^3 \\ (u,v) & \longmapsto & G(u,v) \end{pmatrix}. \tag{3.2}$$

Avec $F(s,t)$ (respectivement $G(u,v)$) un vecteur composé de trois polynômes en (s,t) (respectivement en (u,v)). Signalons également que F (respectivement G) désignera abusivement aussi bien l'application que son image *i.e.* le carreaux de surface $Im(F)$ (respectivement $Im(G)$). Enfin, on supposera que l'intersection $F \cap G$ est une courbe :

$$\mathcal{C} = \left\{ (s,t,u,v) \in [0,1]^4 \mid F(s,t) - G(u,v) = 0 \right\}.$$

3.1 Méthode basée sur les résultants

On propose ici une approche du problème d'intersection entre F et G basée sur les résultants. Pour cela, on commence par introduire quelques généralités sur cet outil dans le cas bivarié.

3.1.1 Généralités sur les résultants bivariés

Définition 3.1 : Soient f_1, f_2 et f_3 trois polynômes bivariés et N le nombre de termes dans les trois supports monomiaux. En d'autres termes, N est la somme du nombre de monômes de f_1, de f_2 et de f_3. Pour tout $i \in \{1, 2, 3\}$, on note coeffs(f_i) la séquence des coefficients de f_i. Le résultant de f_1, f_2 et f_3 est un polynôme irréductible R à N variables ayant la propriété suivante :

$$R(\text{coeffs}(f_1), \text{coeffs}(f_2), \text{coeffs}(f_3)) = 0$$

si et seulement si f_1, f_2 et f_3 ont une racine commune dans un domaine spécifique $\mathcal{D}$.

Pour une description plus précise des résultants, on renvoie par exemple à [7, 14, 15]. Dans le cas du problème d'intersection, le résultant peut-être utilisé comme un opérateur de projection. En effet, si f_1, f_2 et f_3 sont les trois composantes de $F(s,t) - G(u,v)$ considérés comme des polynômes bivariés en (s,u), alors le résultant de f_1, f_2 et f_3 est un polynôme en $R(t,v)$ qui définit une courbe implicite plane correspondante à la projection de $\mathcal{C}$ dans l'espace des paramètres (t,v). Plus précisément, si f_1, f_2 et f_3 sont génériques, alors les deux ensembles suivants sont identiques :

$$\left\{(t,v) \in [0,1]^2 \mid R(t,v) = 0\right\}$$

$$\left\{(t,v) \in [0,1]^2 \mid \exists (s,u) \in \mathcal{D} : F(s,t) = G(u,v)\right\}$$

Les résultants multivariés ont été largement étudiés et de nombreuses implémentations existent (voir [11, 45]).

3.1.2 Application au problème d'intersection

Une stratégie pour décrire l'intersection entre F et G consiste à étudier la projection de $\mathcal{C}$ dans un plan en utilisant le résultant. La plupart des auteurs ont l'habitude de projeter $\mathcal{C}$ dans le plan (s,t) ou (u,v) et utilisent un algorithme de tracé (*tracing algorithm*) pour étudier la courbe plane obtenue (voir [31] et [43] pour ce qui concerne les algorithmes de tracé) et « remontent » dans $\mathbb{R}^3$ en utilisant la paramétrisation adéquate. Notons cependant que ce genre de méthode peut donner lieu à des composantes indésirables (dites « composantes fantômes ») qui ne sont pas dans $F([0,1]^2) \cap G([0,1]^2)$. Ainsi, on a besoin d'une étape d'exclusion des branches indésirables. Pour cela, on peut utiliser un solveur polynomial (comme décrit dans la section 1.2 ou plus précisément dans [49]) ou alors une inversion de paramétrisation (voir [8]).

On propose une solution alternative consistant à projeter $\mathcal{C}$ dans le plan (s,u). Ce choix est intéressant dans la mesure où les deux variables s et u sont séparées dans les équations $F(s,t) = G(u,v)$. Ainsi, on peut éliminer ces variables via un calcul simple de résultant. Dans ce cas précis, calculer le résultant revient à calculer un bézoutien (voir [9, 25]).

Notons $F(s,t) - G(u,v) = (f_0, f_1, f_2)$ avec f_0, f_1 et f_2 dans $\mathbb{R}[s,u][t,v]$ *i.e.* on considère que f_0, f_1 et f_2 sont des polynômes en les variables t et v dont les coefficients sont dans l'anneau $\mathbb{R}[s,u]$. Alors, le bézoutien de $f_0(t,v)$, $f_1(t,v)$ et $f_2(t,v)$ est un polynôme de $\mathbb{R}[s,u][t,v,t_1,v_1]$ donné par le déterminant suivant :

$$b = \begin{vmatrix} f_0(t,v) & \theta_1(f_0)(t,v) & \theta_2(f_0)(t,v) \\ f_1(t,v) & \theta_1(f_1)(t,v) & \theta_2(f_1)(t,v) \\ f_2(t,v) & \theta_1(f_2)(t,v) & \theta_2(f_2)(t,v) \end{vmatrix}$$

avec, pour tout $i \in \{0,1,2\}$

$$\theta_1(f_i)(t,v) = \frac{f_i(t,v) - f_i(t_1,v)}{t - t_1} \tag{3.3}$$

$$\theta_2(f_i)(t,v) = \frac{f_i(t_1,v) - f_i(t_1,v_1)}{v - v_1} \tag{3.4}$$

Notons que pour tout $i \in \{0,1,2\}$, $\theta_1(f_i)$ (respectivement $\theta_2(f_i)$) est un polynôme car $t = t_1$ (respectivement $v = v_1$) est racine de $f_i(t,v) - f_i(t_1,v)$

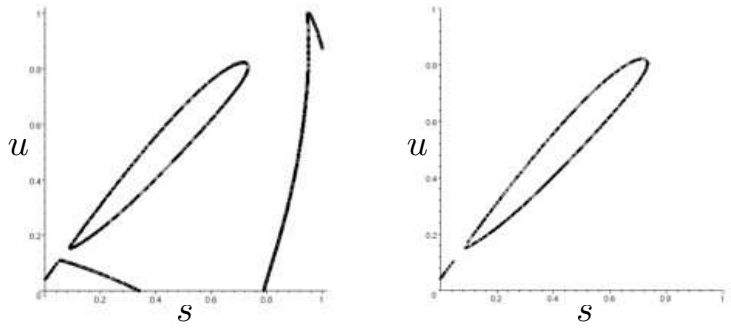


FIG. 3.1: Projection de $\mathcal{C}$ dans l'espace (s, u) avec (à gauche) et sans (à droite) composante fantôme.

(respectivement de $f_i(t_1, v) - f_i(t_1, v_1)$). Explicitement, dans notre cas, le bézoutien est donné par :

$$b = \det\left(F(s,t) - G(u,v), \frac{F(s,t) - F(s,t_1)}{t - t_1}, \frac{G(u,v) - G(u,v_1)}{v - v_1}\right).$$

Le support monomial de b en les variables (t, v) est $\mathcal{S} = \{1, t, v, tv\}$ et celui en les variables (t_1, v_1) est $\mathcal{S}_1 = \{1, t_1, v_1, t_1v_1\}$. Un monôme de b est donc le produit d'un élément de $\mathcal{S}$ et d'un élément de $\mathcal{S}_1$. On forme alors la matrice 4×4 constituée des coefficients de b indexés par le produit cartésien $\mathcal{S} \times \mathcal{S}_1$. Cette matrice, ne contenant que les variables s et u, est appelée la matrice bézoutienne de f_0, f_1 et f_2 et son déterminant est exactement le résultant $R(s, u)$ désiré. Ce résultant donne une courbe implicite plane qui correspond à la projection de $\mathcal{C}$ dans le plan (s, u). On calcul alors la topologie de cette courbe ([34, 61]) et on la trace ([31, 43]) en évitant les composantes fantômes (voir la figure 3.1). On remonte alors les points obtenus dans l'espace pour avoir la courbe d'intersection.

Bien entendu, la méthode qui vient d'être décrite permet également d'obtenir la projection de $\mathcal{C}$ dans le plan (t, v) par des calculs similaires.

3.1.3 Exemples

La méthode basée sur les résultants qui vient d'être décrite ainsi que celles utilisant l'implicitisation approchée et la semi-implicitisation (décrites respectivement dans les sections 3.2 et 3.3) ont été développé dans [13] et testé sur des configurations d'intersection de carreaux de surfaces biquadratiques (*i.e.* paramétrées polynomiales de bidegré $(2,2)$). Il s'agit des exemple 3.1, 3.2 et 3.3. Les résultats obtenus par méthode du résultant sont représentés respectivement sur les figures 3.2, 3.3 et 3.4.

Exemple 3.1 : On considère deux surfaces biquadratiques présentant une courbe d'intersection avec une boucle. Les points de contrôle de chacune des surfaces considérées sont donnés par :

$$\underbrace{\begin{bmatrix} \left(\frac{1}{7},0,\frac{3}{5}\right) & \left(\frac{3}{5},\frac{1}{5},\frac{3}{4}\right) & \left(1,0,\frac{7}{10}\right) \\ \left(\frac{3}{8},\frac{4}{9},\frac{2}{3}\right) & \left(\frac{2}{3},\frac{3}{4},\frac{1}{3}\right) & \left(\frac{6}{7},\frac{3}{8},\frac{5}{7}\right) \\ \left(\frac{1}{5},\frac{6}{7},\frac{4}{7}\right) & \left(\frac{3}{4},\frac{7}{8},\frac{3}{4}\right) & \left(\frac{7}{8},\frac{7}{9},\frac{5}{8}\right) \end{bmatrix}}_{F(s,t)}$$

$$\underbrace{\begin{bmatrix} \left(\frac{2}{7},\frac{1}{7},\frac{2}{5}\right) & \left(\frac{3}{5},\frac{1}{10},\frac{2}{3}\right) & \left(1,0,\frac{4}{5}\right) \\ \left(\frac{3}{8},\frac{4}{9},\frac{2}{3}\right) & \left(\frac{1}{3},\frac{1}{2},1\right) & \left(\frac{5}{7},\frac{3}{8},\frac{2}{7}\right) \\ \left(\frac{1}{5},\frac{6}{7},\frac{3}{7}\right) & \left(\frac{3}{4},\frac{7}{8},\frac{5}{8}\right) & \left(\frac{7}{8},\frac{4}{7},\frac{1}{2}\right) \end{bmatrix}}_{G(u,v)}.$$

■

Exemple 3.2 : Il s'agit d'un exemple présentant non seulement une configuration d'intersection mais également d'auto-intersection (l'auto-intersection de cet exemple est traité dans la section 3.4). Les points de contrôle sont générés par un procédé pseudo aléatoire.

$$\underbrace{\begin{bmatrix} \left(\frac{501}{775},\frac{388}{775},\frac{588}{775}\right) & \left(\frac{347}{775},\frac{276}{775},\frac{479}{775}\right) & \left(\frac{309}{775},\frac{604}{775},\frac{498}{775}\right) \\ \left(\frac{553}{775},\frac{454}{775},\frac{293}{775}\right) & \left(\frac{336}{775},\frac{382}{775},\frac{469}{775}\right) & \left(1,\frac{426}{775},\frac{137}{775}\right) \\ \left(\frac{337}{775},\frac{308}{775},\frac{258}{775}\right) & \left(\frac{517}{775},0,\frac{367}{775}\right) & \left(\frac{533}{775},\frac{492}{775},\frac{564}{775}\right) \end{bmatrix}}_{F(s,t)}$$

$$\underbrace{\begin{bmatrix} \left(\frac{492}{775}, \frac{67}{155}, \frac{522}{775}\right) & \left(\frac{543}{775}, \frac{322}{775}, \frac{117}{775}\right) & \left(\frac{346}{775}, \frac{13}{155}, \frac{4}{5}\right) \\ \left(\frac{113}{155}, \frac{392}{775}, \frac{58}{155}\right) & \left(\frac{632}{775}, \frac{469}{775}, \frac{413}{775}\right) & \left(\frac{307}{775}, \frac{514}{775}, \frac{564}{775}\right) \\ \left(\frac{602}{775}, \frac{129}{775}, \frac{274}{775}\right) & \left(\frac{669}{775}, \frac{692}{775}, \frac{53}{155}\right) & \left(\frac{488}{775}, \frac{219}{775}, \frac{412}{775}\right) \end{bmatrix}}_{G(u,v)}$$

■

Exemple 3.3 : Il s'agit d'un exemple de configuration présentant une intersection tangente.

$$\underbrace{\begin{bmatrix} \left(0, \frac{1}{7}, \frac{4}{5}\right) & \left(\frac{3}{5}, \frac{1}{13}, \frac{1}{3}\right) & \left(1, 0, \frac{4}{5}\right) \\ \left(\frac{1}{8}, \frac{4}{9}, \frac{11}{40}\right) & \left(\frac{1}{3}, \frac{34}{65}, \frac{3}{4}\right) & \left(\frac{6}{7}, \frac{3}{8}, -\frac{16}{35}\right) \\ \left(\frac{1}{5}, \frac{6}{7}, \frac{4}{5}\right) & \left(\frac{3}{4}, \frac{443}{520}, \frac{3}{8}\right) & \left(\frac{7}{8}, 1, \frac{14}{15}\right) \end{bmatrix}}_{F(s,t)}$$

$$\underbrace{\begin{bmatrix} \left(0, \frac{1}{7}, \frac{1}{5}\right) & \left(\frac{3}{5}, \frac{1}{10}, \frac{1}{3}\right) & \left(1, 0, \frac{1}{5}\right) \\ \left(\frac{1}{8}, \frac{4}{9}, \frac{7}{8}\right) & \left(\frac{1}{3}, \frac{1}{2}, \frac{3}{4}\right) & \left(\frac{6}{7}, \frac{3}{8}, \frac{1}{7}\right) \\ \left(\frac{1}{5}, \frac{6}{7}, \frac{1}{5}\right) & \left(\frac{3}{4}, \frac{7}{8}, \frac{3}{8}\right) & \left(\frac{7}{8}, 1, \frac{1}{3}\right) \end{bmatrix}}_{G(u,v)}$$

■

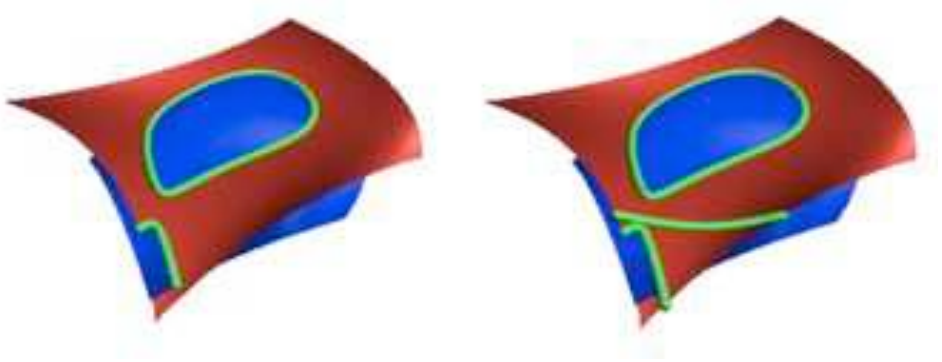


FIG. 3.2: Exemple 3.1 par méthode du résultant avec élimination des composantes fantômes à gauche et sans à droite.

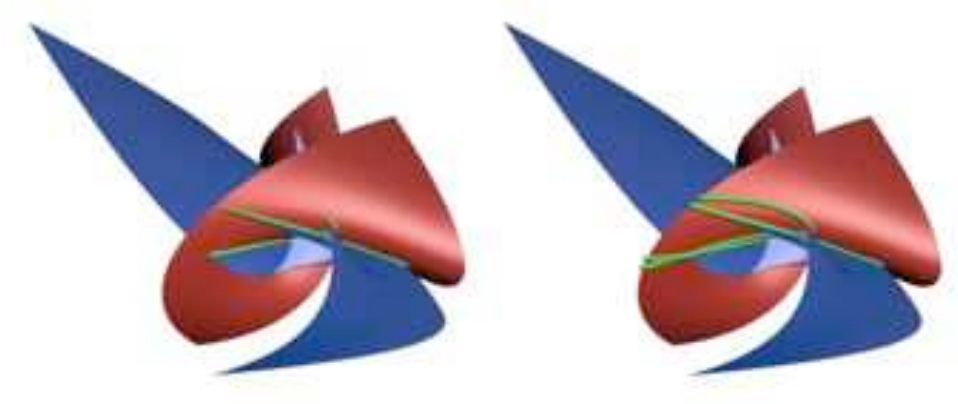


FIG. 3.3: Exemple 3.2 par méthode du résultant avec élimination des composantes fantômes à gauche et sans à droite.

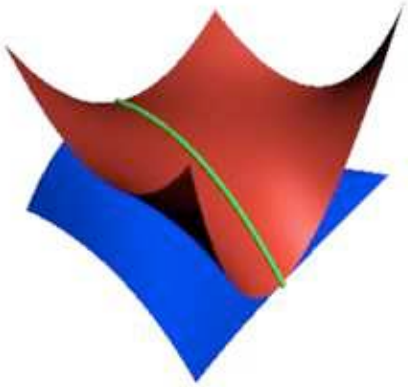


FIG. 3.4: Exemple 3.3 par méthode du résultant.

3.2 Implicitisation approchée et intersection

Dans cette section, on propose une approche du problème d'intersection entre deux carreaux de surfaces biquadratiques par implicitisation approchée. On commence par rappeler ce qu'est le problème d'implicitisation (approchée) et on donnera une méthode spécifique pour le cas biquadratique. L'application au problème d'intersection ainsi que certains exemples seront aussi exposés.

3.2.1 Implicitisation

Le problème d'implicitisation consiste à trouver l'équation implicite (*i.e.* une représentation algébrique) d'une surface paramétrée rationnelle donnée. Il existe de nombreuses techniques pour traiter ce problème. Les plus répandues utilisent les résultants ou les bases de Gröbner (voir [14, 15, 41]). Cependant, en général, l'implicitisation requiert des temps de calculs assez conséquents du fait que le degré de l'équation implicite cherchée est élevé. En effet, si on se donne une surface paramétrée rationnelle générique de bidegré (d_1,d_2), alors l'équation implicite correspondante sera de degré $2d_1d_2$. En général, un polynôme à n variables de degré d contient $\binom{d+n}{n}$ coefficients donc les équations implicites peuvent avoir des tailles considérables. Signalons également que toute courbe ou surface paramétrée rationnelle admet une représentation algébrique alors que la réciproque est fausse. Le lien entre les deux représentations (paramétrée ou algébrique) peut-être complexe. En effet, dans la mesure où un objet paramétré est de nature locale alors qu'un objet algébrique est de nature globale, il se peut que l'on soit confronté à des problèmes dis de « composantes fantômes » *i.e.* que la courbe ou la surface implicite présente des composantes absentes de la paramétrisation correspondante. Ainsi, il semble raisonnable de chercher une représentation algébrique approchée plutôt qu'exacte. Dans ce contexte, les calculs seront plus efficaces et les résultats comporteront, a priori, moins de composantes fantômes. Ce point de vue a déjà été considéré dans [21, 44, 62].

3.2.2 Implicitisation approchée

Le problème auquel on s'intéresse met en jeux des surfaces paramétrées polynomiales. Considérons donc une surface paramétrée polynomiale réelle $S(u, v)$ sur $[0, 1]^2$. Soit $d \in \mathbb{N}^*$ (le degré de l'équation implicite approchée) et $\epsilon \geq 0$ (une tolérance). Au sens de Tor Dokken dans [20], le problème d'implicitisation approchée consiste à trouver un polynôme non nul $P \in \mathbb{R}[x, y, z]$ de degré d tel que :

$$\forall (u, v) \in [0, 1]^2, P\left(S(u, v) + \alpha(u, v)\,\mathbf{g}(u, v)\right) = 0$$

avec $|\alpha(u, v)| \leq \epsilon$ et $||\mathbf{g}(u, v)||_2 = 1$.

En fait, α est la fonction d'erreur et $\mathbf{g}$ est la direction suivant laquelle on mesure cette erreur. Cette dernière peut être, par exemple, la normale unitaire à S.

3.2.3 Implicitisation d'une surface biquadratique

Une question essentielle dans l'implicitisation approchée est le choix du degré. Il semblerait qu'un facteur pertinent pour faire ce choix est la topologie et ce, particulièrement lorsque la surface initiale comporte des singularités ou des auto-intersections. L'utilisation du degré 4 a été suggéré par Tor Dokken. Après plusieurs expérimentations, ce dernier a conclu que les surfaces algébriques de degré 4 (communément appelées quartiques) avaient suffisamment de degrés de liberté pour approximer la plupart des cas rencontrés en modélisation géométrique. Si on considère une surface biquadratique *i.e.* paramétrée polynomiale de bidegré $(2, 2)$, alors son équation implicite exacte est de degré 8 et donc utiliser un degré 4 semble être un bon compromis. On se propose à présent de donner deux méthodes pour approximer une surface biquadratic S par une quartique P. On note l'équation implicite :

$$P(x, y, z) = \sum_{i=0}^{4} \sum_{j=0}^{4-i} \sum_{k=0}^{4-i-j} b_{ijk}\, x^i y^j z^k$$

avec $b = (b_{000}, b_{100}, \ldots, b_{004}) \in \mathbb{R}^{35}$ le vecteur des coefficients de P. Soit $\beta(u, v)$ le vecteur formé par le produit tensoriel des polynômes de Bernstein de bidegré $(8, 8)$ (concrètement $\beta(u, v)$ est un vecteur colonne composé de 81 polynômes en (u, v)).

Méthode de Tor Dokken. Cette méthode qu'on peut retrouver en détails dans [20] procède de la manière suivante :

1. Factoriser $P(S(u, v)) = (Db)^T \beta(u, v)$ où D est une matrice 81×35.
2. Faire une décomposition en valeurs singulières de D.
3. Prendre pour b le vecteur correspondant à la plus petite valeur singulière de D.

Cette méthode est assez générale et ne fait aucune hypothèse sur le bidegré de S. On peut donc adopter une stratégie géométrique spécifique au cas

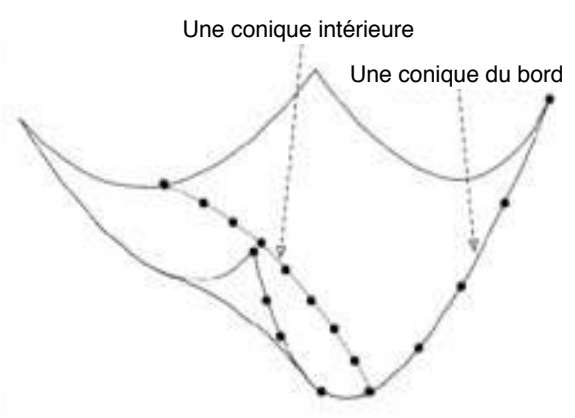


FIG. 3.5: Caractérisation d'une conique par 9 points sur une surface biquadratique.

biquadratique.

Méthode géométrique par évaluations. Dans cette approche, on construit des contraintes géométriques pertinentes afin d'obtenir un système linéaire en les inconnues $b_{000}, b_{100}, \ldots, b_{004}$. On utilise alors une décomposition en valeurs singulières pour la résolution de ce système. On a choisi de caractériser des coniques sur S. En particulier les quatre coniques qui forment le bord de S ainsi que deux coniques intérieures. Il s'agit explicitement de :

$$\begin{aligned} C_1 &= S([0,1] \times \{0\}), & C_2 &= S([0,1] \times \{1\}) \\ C_3 &= S(\{0\} \times [0,1]), & C_4 &= S(\{1\} \times [0,1]) \\ C_5 &= S(\{\tfrac{1}{2}\} \times [0,1]), & C_6 &= S([0,1] \times \{\tfrac{1}{2}\}) \end{aligned}$$

Lemme 3.1 : Soit $i \in \{1, \ldots, 6\}$. Si la quartique d'équation $P = 0$ contient 9 points distincts de C_i alors elle contient C_i. ◊

Démonstration : C_i a une paramétrisation de degré 2 et P est de degré 4. Donc en injectant la paramétrisation de C_i dans P, on obtient un polynôme univarié de degré 8. □

En utilisant l'observation géométrique du lemme 3.1, on peut construire un système linéaire et le résoudre via une décomposition en valeurs singulières. Cela permet alors d'avoir une approximation algébrique de degré 4 de la

surface S.

3.2.4 Application au problème d'intersection

Une manière de traiter le problème d'intersection de deux surfaces paramétrée polynomiales F et G consiste à appliquer l'implicitisation approchée sur F (par exemple), ce qui permet d'obtenir un polynôme $P(x, y, z)$, puis à étudier $P(G(u, v))$ qui donne une représentation implicite de la courbe d'intersection de F et G dans le domaine des paramètres (u, v). On applique alors des méthodes classiques de tracé de courbes implicites planes pour obtenir le lieux d'intersection.

Cependant, bien que ces méthodes d'implicitisation approchée sont relativement efficaces, elles ne fournissent pas toujours des résultats satisfaisants. Quand un carreau de surface paramétrée polynomiale donné est simple (*i.e.* avec une certaine planéité et sans singularité ni auto-intersection), son approximation algébrique est relativement fidèle. Dans le problème d'intersection, de telles méthodes peuvent être combinées avec une subdivision. Cela permet, entre autres, d'exclure des domaines ne présentant aucune intersection (en utilisant des boîtes englobantes comme décrit dans la section 2.4.2 du chapitre 2), et également d'éviter des configurations problématiques comme la présence de boucles dans l'intersection (utilisation du critère d'Hohmeyer donné dans le théorème 3.1 et dans [42]). Toutefois, il faut avouer que dans des configurations avec des singularités, on a, en général, de très mauvais résultats. En effet, même si l'on dispose d'un bon critère dans la subdivision, on peut être confronté au problème des composantes fantômes qui subsistent mais qui sont moins nombreuses que dans une implicitisation exacte. Il convient alors de faire une étape supplémentaire d'élimination de ces composantes fantômes. Par ailleurs, il y a un autre inconvénient provenant de l'approximation algébrique qui est la difficulté à obtenir des points certifiés sur le lieu d'intersection.

Theorème 3.1 (Hohmeyer) : On garde toujours les mêmes notations données par les formules (3.1) et (3.2). On désigne par N_F (respectivement N_G), le champ de vecteurs normaux à F (respectivement à G) *i.e.* $N_F : (s, t) \mapsto \partial_s F(s, t) \times \partial_t F(s, t)$ (respectivement $N_G : (u, v) \mapsto \partial_u G(u, v) \times \partial_v G(u, v)$). S'il existe deux vecteurs $\overrightarrow{w_1}$ et $\overrightarrow{w_2}$ dans $\mathbb{R}^3$ tels que pour tout $(\overrightarrow{n_F}, \overrightarrow{n_G}) \in$

$Im(N_F) \times Im(N_G)$, on ait :

$$\begin{cases} \overrightarrow{w_1} \cdot \overrightarrow{n_F} > 0 \\ \overrightarrow{w_1} \cdot \overrightarrow{n_G} < 0 \\ \overrightarrow{w_2} \cdot \overrightarrow{n_F} > 0 \\ \overrightarrow{w_2} \cdot \overrightarrow{n_G} > 0 \end{cases}$$

alors la courbe d'intersection $F \cap G$ ne comporte ni boucle ni de singularité.

Remarque 3.1 :

- On peut se contenter de satisfaire les inégalités du théorème 3.1 sur les points de contrôle de N_F et N_G grâce au corollaire 1.2.
- La question de l'existence des deux vecteurs $\overrightarrow{w_1}$ et $\overrightarrow{w_2}$ dans le théorème 3.1 est un problème de programmation linéaire.
- Si les deux vecteurs $\overrightarrow{w_1}$ et $\overrightarrow{w_2}$ du théorème 3.1 existent, alors le vecteur $\overrightarrow{w} = \overrightarrow{w_1} \times \overrightarrow{w_2}$ donne une direction dans laquelle la courbe d'intersection est monotone *i.e.* pour tout vecteur tangent $\overrightarrow{T}$ à cette courbe d'intersection dans $\mathbb{R}^3$, on a $\overrightarrow{T} \cdot \overrightarrow{w} > 0$. •

3.2.5 Exemples

Les exemples de configurations d'intersection entre deux surfaces biquadratiques donnés dans la section 3.1.3 ont également été testé avec la seconde méthode d'implicitisation approchée (dite « géométrique par évaluation »). Les exemples 3.1 et 3.2 donnent également des composantes fantômes comme on peut le voir respectivement sur les figures 3.6 et 3.7. Quant à l'exemple 3.3, l'approximation algébrique n'est pas adaptée dans la mesure où l'on obtient un résultat incorrect. Plus précisément, on trouve deux courbes d'intersection assez proches (voir la figure 3.8).

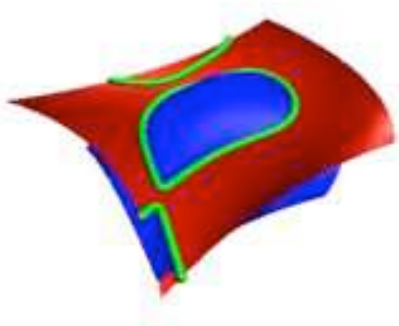


FIG. 3.6: Exemple 3.1 par la méthode d'implicitisation approchée géométrique.

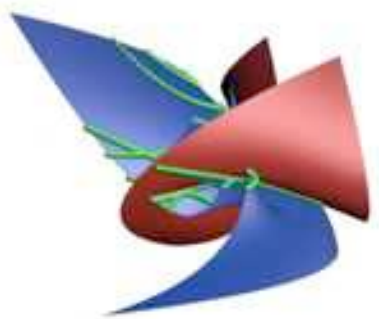


FIG. 3.7: Exemple 3.2 par la méthode d'implicitisation approchée géométrique.

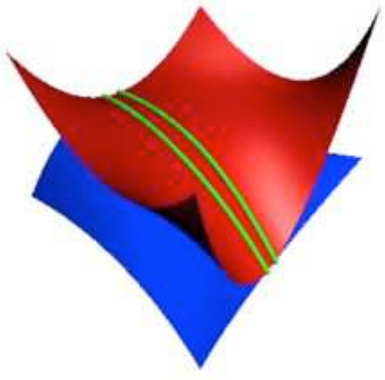


FIG. 3.8: Exemple 3.3 par la méthode d'implicitisation approchée géométrique.

3.3 Approche semi-implicite

Il s'agit d'une méthode élaborée par Oberneder et Jüttler et qui est exposée dans notre article commun [13]. Elle se base sur une représentation semi-implicite des surfaces biquadratiques au sens de [10].

3.3.1 Intersection d'une courbe coordonnée

Définition 3.2 : En gardant toujours les mêmes notations données par les formules (3.1) et (3.2), une courbe coordonnée de $F(s,t)$ pour une valeur constante de paramètre $s = s_0$ est une courbe paramétrée donnée par :

$$\mathbf{p}(t) = F(s_0, t) = \mathbf{a}_0(s_0) + \mathbf{a}_1(s_0)\, t + \mathbf{a}_2(s_0)\, t^2$$

avec $\mathbf{a}_0$, $\mathbf{a}_1$ et $\mathbf{a}_2 \in \mathbb{R}^3$.

Dans notre cas, il s'agit d'une conique (donc plane) qu'on pourra représenter comme l'intersection d'un plan avec un cylindre sur une conique comme on l'a illustré sur la figure 3.9. Dorénavant, on fixe un paramètre s_0 qui permet de définir une courbe coordonnée tracée sur F. Dans la mesure où on ne s'intéresse qu'à l'intersection $F \cap G$ dans une certaine région, on introduit également deux plans π_1 et π_2 pour borner cette région. Enfin, signalons que dans le cas particulier où la courbe coordonnée est une droite, cette dernière est représentée comme intersection de deux plans orthogonaux.

On désire calculer l'intersection de la courbe coordonnée avec la second carreau biquadratique $G(u,v)$. Pour cela, on utilise le schéma suivant :

1. Décrire la courbe coordonnée comme l'intersection d'un plan avec un cylindre.
2. Calculer l'intersection $\mathcal{I}$ du plan avec le second carreau biquadratique $G(u,v)$.
3. Restreindre la courbe d'intersection $\mathcal{I}$ à la région voulue.
4. Intersecter le cylindre avec la courbe restreinte précédente.

Les quatre étapes précédentes vont maintenant être expliquées en détails.

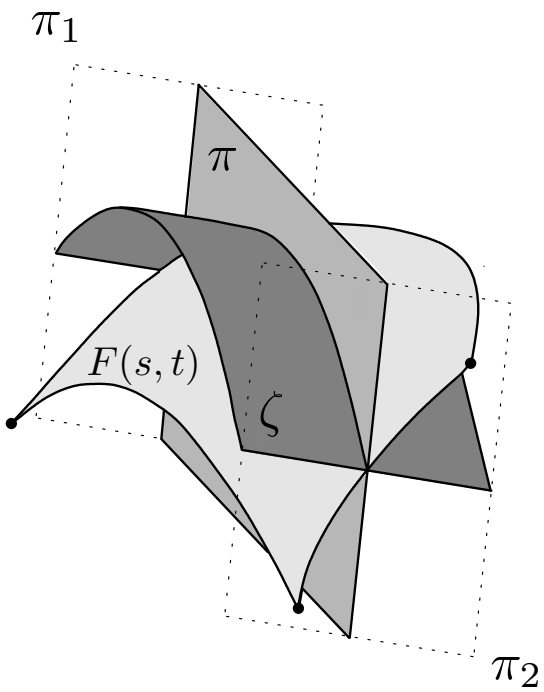


FIG. 3.9: Représentation d'une courbe coordonnée comme l'intersection d'un plan et d'un cylindre sur une conique.

Définition du plan, du cylindre et des deux plans du bord. La courbe coordonnée et ses trois points de contrôle sont coplanaires. Ceci étant dit, pour obtenir un vecteur normal $\mathbf{n}(s_0) = (n_1(s_0), n_2(s_0), n_3(s_0))$ au plan, il suffit de considérer le produit vectoriel de deux vecteurs formés par ces trois points de contrôle. Il s'ensuit que le plan en question est donné par une équation de la forme :

$$\pi(s_0)(x, y, z) = e_0(s_0) + n_1(s_0)\, x + n_2(s_0)\, y + n_3(s_0)\, z. \tag{3.5}$$

En extrudant la courbe coordonnée dans la direction $\mathbf{n}(s_0)$, on obtient une paramétrisation du cylindre qui intersecte orthogonalement le plan.

$$(p, q) \mapsto \mathbf{w}(s_0)(p, q) = F(s_0, p) + q \cdot \mathbf{n}.$$

L'équation implicite de ce cylindre en revanche est plus compliquée. Pour la calculer, on peut utiliser soit un résultant de Sylvester soit une méthode

par coefficients indéterminés. On obtient une équation de la forme :

$$\begin{aligned}\zeta(s_0)(x,y,z) \;:=\; & a_0(s_0) + a_1(s_0)\,x + a_2(s_0)\,y + a_3(s_0)\,z \\ & + a_4(s_0)\,x\,y + a_5(s_0)\,x\,z + a_6(s_0)\,y\,z \\ & + a_7(s_0)\,x^2 + a_8(s_0)\,y^2 + a_9(s_0)\,z^2 \;=\; 0. \end{aligned} \tag{3.6}$$

Dans le cas où la courbe coordonnée est une droite, on choisit deux plans orthogonaux qui s'intersectent le long de cette droite. On crée ensuite les deux plans $\pi_1(x,y,z)$ et $\pi_2(x,y,z)$ qui bornent la courbe coordonnée. Pour cela, on peut, par exemple, choisir les deux plans normaux à la courbe coordonnée en ses extrémités.

Intersection du plan avec $G(u,v)$. En substituant le second carreau biquadratique $G(u,v)$ dans l'équation (3.5) du plan π, on obtient un polynôme de bidegré $(2,2)$ en (u,v) qu'on peut interpréter comme un polynôme de degré 2 en u à coefficients dépendants de v. L'intersection $\pi \cap G$ se traduit alors par l'équation :

$$\pi(G(u,v)) = a(v)\,u^2 + b(v)\,u + c(v) = 0. \tag{3.7}$$

Pour toute valeur de v, on obtient deux solutions $u_1(v)$ et $u_2(v)$ de la forme :

$$u_{1,2}(v) = -\frac{b(v)}{2\,a(v)} \pm \sqrt{d(v)} \quad \text{avec} \quad d(v) = \frac{b(v)^2}{4\,a(v)^2} - \frac{c(v)}{a(v)}. \tag{3.8}$$

Ces solutions permettent de paramétrer les deux branches de la courbe d'intersection $\mathcal{I}$ dans le plan des paramètres (u,v) de la seconde surface G. En résolvant plusieurs équations de degré 2, on détermine les sous intervalles de $[0,1]$ où on a $d(v) \geq 0$ et $0 \leq u_i(v) \leq 1$. Cela donne une liste de domaines candidats pour chaque branche de la courbe d'intersection.

En composant (3.8) avec G on obtient ces deux branches $\mathbf{k}_1(v)$ et $\mathbf{k}_2(v)$ de la courbe d'intersection $\mathcal{I}$.

$$\mathbf{k}_{1,2}(v) = G(u_{1,2}(v), v) = \frac{1}{a(v)^2}\mathbf{h}(v) \pm \frac{\sqrt{d(v)}}{a(v)}\mathbf{l}(v) + d(v)\mathbf{m}(v) \tag{3.9}$$

où les composantes de $\mathbf{h}(v)$, $\mathbf{l}(v)$ et $\mathbf{m}(v)$ sont polynomiales de degrés respectifs 6, 4, et 2.

Restriction du domaine. Dans la mesure où la région à laquelle on s'intéresse est située entre les plans $\pi_1(x, y, z)$ et $\pi_2(x, y, z)$, après un choix d'orientation, les deux inégalités suivantes doivent être satisfaites.

$$\begin{cases} \pi_1(x, y, z) \geq 0 \\ \pi_2(x, y, z) \leq 0 \end{cases}.$$

Il s'agit donc d'imposer :

$$\begin{cases} k_{\pi_1}(v) := \pi_1(G(u(v), v)) \geq 0 \\ k_{\pi_2}(v) := \pi_2(G(u(v), v)) \leq 0 \end{cases}.$$

Cela rajoute donc des contraintes pour les domaines possibles du paramètre v. Pour toute branche de la courbe d'intersection, on crée une liste de domaines possibles qu'on garde en mémoire. Les bords des intervalles peuvent être calculés en résolvant trois systèmes polynomiaux composés chacun de deux équations de bidegrés $(2, 2)$. De manière équivalente, cela revient également à résoudre un système polynomial composé de trois équations de degrés 8 obtenues après élimination du paramètre u. Pour résoudre ce système, on peut, au choix, utiliser le solveur décrit dans la section 1.2.2 (qui est également expliqué dans [49]) ou alors des méthodes telles que celles que l'on peut trouver dans [24] ou [53].

Exemple 3.4 : La figure 3.10 illustre un domaine de paramètres en u et v (correspondant au carreau de surface G). Seule la première branche $u_1(v)$ de la courbe d'intersection apparaît. La condition $0 \leq u \leq 1$ n'impose aucune condition supplémentaire sur v dans ce cas. Cependant, les intersections avec les plans π_1 et π_2 donnent lieu à deux courbes supplémentaires (k_{π_1} et k_{π_2}) qui doivent être intersectées avec la courbe $v = u_1(v)$ ce qui donne deux bornes v_0 et v_1 pour le domaine à considérer. ∎

Intersection du cylindre avec la courbe d'intersection $\mathcal{I}$ restreinte. On dispose de l'équation implicite $\zeta(s_0)(x, y, z)$ du cylindre donnée par (3.6) et de la paramétrisation $\mathbf{k}_{1,2}(v)$ par v de la courbe d'intersection $\mathcal{I} = \pi \cap G$ (cette paramétrisation est donnée par (3.9)). On effectue alors la substitution $\zeta(s_0)(\mathbf{k}_{1,2}(v))$ et on ordonne les termes obtenus suivant les puissances

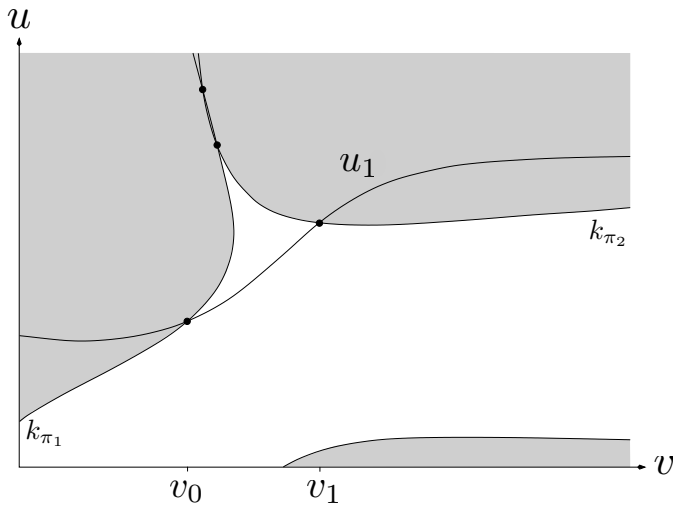


FIG. 3.10: Identification des intervalles à considérer pour les valeurs de v.

croissantes de $\sqrt{d(v)}$ pour obtenir une expression de la forme :

$$\zeta(s_0)(\mathbf{k}_{1,2}(v)) = p_1(v) + p_2(v)\sqrt{d(v)} + p_3(v)\left(\sqrt{d(v)}\right)^2 + \\ + p_4(v)\left(\sqrt{d(v)}\right)^3 + p_5(v)\left(\sqrt{d(v)}\right)^4 \qquad (3.10)$$

avec, pour tout $i \in \{1, \ldots, 5\}$, $p_i(v)$ de degré 12.
Pour résoudre $\zeta(s_0)(\mathbf{k}_{1,2}(v)) = 0$, il faut se débarrasser des racines carrées en séparant les puissances paires et impaires de $\sqrt{d(v)}$. Plus précisément, on écrit $\zeta(s_0)(\mathbf{k}_{1,2}(v)) = A - B$ avec A (respectivement B) contenant toutes les puissances paires (respectivement impaires) de $\sqrt{d(v)}$. On s'intéresse alors plutôt à l'équation $A^2 d(v) - (B\sqrt{d(v)})^2 = 0$ qui, elle, est polynomiale de degré 24 mais qui peut-être ramenée à un degré 16 par factorisation du discriminant. Remarquons au passage que le degré 16 est cohérent dans la mesure où la représentation algébrique d'une surface biquadratique est de degré 8 et que donc son intersection avec une courbe de degré 2 est génériquement composée de 16 points.

Finalement, on résout cette équation de degré 16 dans les intervalles adéquats de v. Cela donne les points d'intersection de la courbe coordonnée de la première surface avec la seconde surface.

3.3.2 Structure globale de la courbe d'intersection

La section 3.3.1 permet de calculer l'intersection de la courbe coordonnée de F avec G. On propose à présent de revenir au problème initial à savoir celui de l'intersection $F \cap G$.

Pour toute valeur $s = s_0$, la courbe coordonnée $F(s_0, t)$ a un certain nombre de points d'intersection avec G. Si s_0 varie continument, alors le nombre de points d'intersection peut changer uniquement dans l'un des deux cas suivants :

1. un des points d'intersection est sur le bord d'un des deux carreaux de surfaces (points du bord),
2. la courbe coordonnée de la première surface est tangente à la seconde surface (points tangents).

Ainsi, pour analyser la structure globale de la courbe d'intersection $F \cap G$, on procède en deux étapes. D'une part, on trouve et on ordonne les valeurs de s_0 pour lesquelles le nombre de points d'intersection change (un des deux cas précités), ce qui donne une séquence de points critiques :

$$0 = s_0^{(0)} < s_0^{(1)} < \ldots < 1 = s_0^{(K)}.$$

Et d'autre part, pour tout $i \in \{0, \ldots, K-1\}$, on intersecte la courbe coordonnée de F correspondante à $s_0 = \frac{s_0^{(i)} + s_0^{(i+1)}}{2}$ avec G en utilisant ce qui a été fait dans la section 3.3.1. Etant donné que le nombre de points d'intersection entre deux valeurs critiques reste constant, on peut utiliser un algorithme de tracé classique (*tracing algorithm*) comme on peut trouver dans [43] ou bien générer plus de points en intersectant plus de courbes coordonnées.

On explicite maintenant les calculs des valeurs critiques s_0.

Points du bord. Ces points correspondent à l'intersection des courbes coordonnées du bord d'un carreau avec l'autre. Explicitement, il y en a huit et on peut les traiter avec la méthode de la section 3.3.1.

Points tangents. On peut interpréter les points tangents de différentes manières.

1. Le vecteur tangent de la courbe coordonnée est dans la plan tangent de G. Alors $F_s \cdot (G_u \times G_v) = 0$. Il faut aussi prendre en considération le fait que l'ont ait $F - G = 0$. Cela donne donc un système polynomial de 4 équations à 4 inconnues (s, t, u, v). Trois des équations sont de multidegré $(2, 2, 2, 2)$ et une de multidegré $(1, 2, 3, 3)$ et on résout en s.

2. Quelques considérations géométriques peuvent permettre d'éliminer la variable t. En effet, si on est en présence d'un point tangent correspondant à une courbe coordonnée de paramètre $s = s_0$, alors le plan π, définit par l'équation (3.5), doit le contenir (car la courbe coordonnée est dans ce plan). Cela donne l'équation :

$$\pi(s_0)(G(u, v)) = 0. \tag{3.11}$$

Par ailleurs, le cylindre (définit par l'équation (3.6)) doit également contenir ce point tangent. D'où l'équation :

$$\zeta(s_0)(G(u, v)) = 0. \tag{3.12}$$

Enfin, le vecteur tangent de la courbe coordonnée doit être dans le plan tangent à G. Dans la mesure où ce vecteur est colinéaire à

$$V := \nabla\pi(s_0)(G(u, v)) \times \nabla\zeta(s_0)(G(u, v)),$$

on aboutit à l'équation :

$$\det(\partial_u G, \partial_v G, V) = 0 \tag{3.13}$$

Finalement, les équations (3.11), (3.12) et (3.13), qui sont de multidegrés respectifs $(6, 2, 2)$, $(16, 4, 4)$ et $(18, 5, 5)$ en (s_0, u, v), nous donnent un autre système pour caractériser les points tangents.

3.3.3 Exemples

On reprend ici encore les exemples donnés dans la section 3.1.3 et on les traite avec cette approche semi-implicite. Les résultats dans les espaces de paramètres respectifs pour l'exemple 3.1 sont représentés sur la figure 3.11 et pour l'exemple 3.2 sur la figure 3.12 (avec dans ce dernier cas également l'auto-intersection obtenue par la méthode décrite en section 3.4.2). On a également représenté les points critiques. L'exemple 3.3 n'a pas été représenté car il donne également de mauvais résultats (comme sur l'illustration de la figure 3.8).

3.4 Auto-intersection des surfaces biquadratiques

On garde toujours les notations données par les formules (3.1) et (3.2) et on adapte ici les méthodes par résultants et semi-implicitisation développées dans les sections 3.1 et 3.3 pour traiter le problème d'auto-intersection des surfaces biquadratiques. Signalons que l'utilisation de l'implicitisation approchée dans le cadre de ce problème a déjà été traité dans [62].

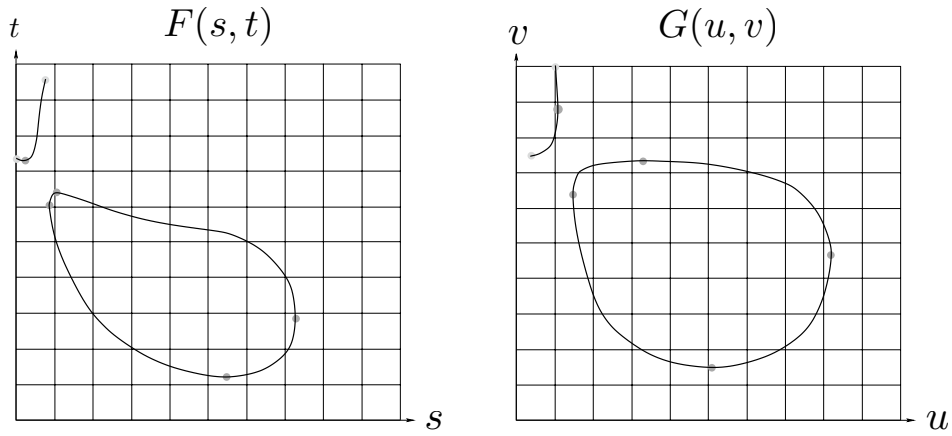


FIG. 3.11: Exemple 3.1 par l'approche semi-implicite.

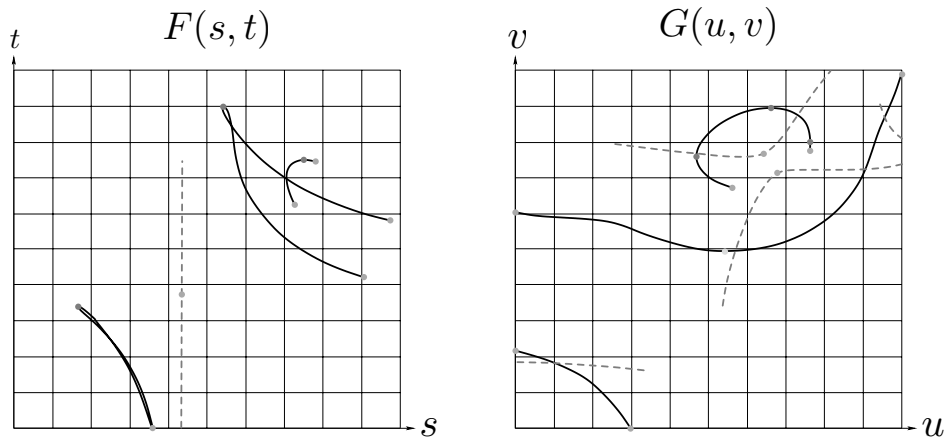


FIG. 3.12: Exemple 3.2 par l'approche semi-implicite (avec l'auto-intersection en pointillés).

3.4.1 Auto-intersection par résultants

Dans le domaine de paramètres $[0,1]^4$, la courbe d'intersection du premier carreau de surface (à savoir F) est de la forme :

$$\left\{(s_1,t_1,s_2,t_2) \in [0,1]^4 \mid (s_1,t_1) \neq (s_2,t_2) \text{ et } F(s_1,t_1) = F(s_2,t_2)\right\}. \quad (3.14)$$

Ce lieu est la trace réelle d'une courbe complexe. On suppose qu'il est soit vide, soit de dimension 0 ou 1. Dans les exemples de la section 3.1.3, le lieu d'auto-intersection est toujours une courbe dans $\mathbb{R}^4$. Afin, d'éliminer les composantes triviales $(s_1,t_1) = (s_2,t_2)$, on utilise un changement de coordonnées. Soient $(s_2,t_1) \in [0,1]^2$ et $(l,k) \in \mathbb{R}^2$. On pose $s_1 = s_2 + l$, $t_2 = t_1 + lk$. Si $(s_1,t_1) \neq (s_2,t_2)$ alors $l \neq 0$. Ainsi, $F(s_1,t_1) = F(s_2,t_2)$ si et seulement si on a :

$$F(s_2 + l, t_1) = F(s_2, t_1 + lk). \quad (3.15)$$

On suppose à présent que (u_2,v_1,l,k) vérifie cette relation (3.15). On introduit également le polynôme suivant :

$$\tilde{T}(s_2,t_1,l,k) = \frac{1}{l}\left[F(s_2+l,t_1) - F(s_2,t_2+lk)\right].$$

Le multidegré de $\tilde{T}(s_2, t_1, l, k)$ en (s_2, t_1, l, k) est $(2, 2, 1, 2)$ et son support monomial en (l, k) ne contient que les termes $k^2 l$, k, l et 1. On peut alors diminuer le degré en considérant :

$$T(s_2, t_1, m, k) = m\tilde{T}(s_2, t_1, \frac{1}{m}, k).$$

Le support monomial de $T(s_2, t_1, m, k)$ en (m, k) ne contient alors plus que 1, m, k^2 et km. On peut donc réécrire T sous forme matricielle :

$$T(s_2, t_1, m, k) = \begin{pmatrix} a_1(s_2, t_1) & b_1(s_2, t_1) & c_1(s_2, t_1) & d_1(s_2, t_1) \\ a_2(s_2, t_1) & b_2(s_2, t_1) & c_2(s_2, t_1) & d_2(s_2, t_1) \\ a_3(s_2, t_1) & b_3(s_2, t_1) & c_3(s_2, t_1) & d_3(s_2, t_1) \end{pmatrix} \begin{pmatrix} 1 \\ m \\ k^2 \\ km \end{pmatrix}$$

L'écriture matricielle du système $T(s_2, t_1, m, k) = 0$ et l'utilisation de la règle de Cramer donnent alors :

$$m = \frac{D_2}{D_1}, \quad k^2 = \frac{D_3}{D_1}, \quad \text{et} \quad km = \frac{D_4}{D_1} \tag{3.16}$$

avec

$$D_1 = \begin{vmatrix} b_1 & c_1 & d_1 \\ b_2 & c_2 & d_2 \\ b_3 & c_3 & d_3 \end{vmatrix}, \; D_2 = \begin{vmatrix} -a_1 & c_1 & d_1 \\ -a_2 & c_2 & d_2 \\ -a_3 & c_3 & d_3 \end{vmatrix},$$

$$D_3 = \begin{vmatrix} b_1 & -a_1 & d_1 \\ b_2 & -a_2 & d_2 \\ b_3 & -a_3 & d_3 \end{vmatrix}, \; D_4 = \begin{vmatrix} b_1 & c_1 & -a_1 \\ b_2 & c_2 & -a_2 \\ b_3 & c_3 & -a_3 \end{vmatrix}.$$

Soit alors $Q(s_2, t_1)$ le polynôme définit par $Q = D_4^2 D_1 - D_2^2 D_3$.

Lemme 3.2 : La courbe implicite $\{(s_2, t_1) \in [0, 1]^2 \mid Q(s_2, t_1) = 0\}$ est la projection du lieu d'auto-intersection (donné par l'ensemble (3.14) mais dans $\mathbb{C}^4$) dans le domaine des paramètres $(s_2, t_1) \in [0, 1]^2$. ◊

Démonstration : $Q(s_2, t_1) = 0$ est la seule relation algébrique (de degré minimal) entre s_2 et t_1 telle que :

$$\forall (s_2, t_1) \in [0, 1]^2, (Q(s_2, t_1) = 0) \Rightarrow \exists (m, k) \in \mathbb{C}^2, T(s_2, t_1, m, k) = 0. \quad \square$$

FIG. 3.13: Auto-intersections des surfaces de l'exemple 3.2 calculées par résultants.

Le lemme 3.2 donne une méthode pour calculer le lieu d'auto-intersection de F. Il suffit d'utiliser un algorithme de tracé sur la courbe implicite $Q(s_2, t_1) = 0$ et, pour tout point (s_2, t_1) sur cette courbe, on obtient, par continuité, le point $(s_1, t_2) \in [0, 1]^2$ correspondant s'il existe.

Cette méthode a été testée sur les deux surfaces de l'exemple 3.2 et les résultats obtenus sont sur la figure 3.13.

3.4.2 Auto-intersection par semi-implicitisation

L'approche par semi-implicitisation décrite dans la section 3.3 permet également de calculer les courbes d'auto-intersection. Il suffit d'intersecter la surface $F(s_1, t_1)$ avec elle-même $F(s_2, t_2)$. Dans ce contexte, les équations (3.7) et (3.10) se factorisent par $(s_2 - s_1)$ et cela permet de se débarrasser des composantes triviales. Les résultats de l'exemple 3.2 apparaissent en pointillés sur la figure 3.12. Il convient de signaler que le calcul des points tangents comme il est décrit dans la section 3.3.2 donne alors soit des points tangents « classiques » soit des cusps comme illustrés sur la figure 3.14.

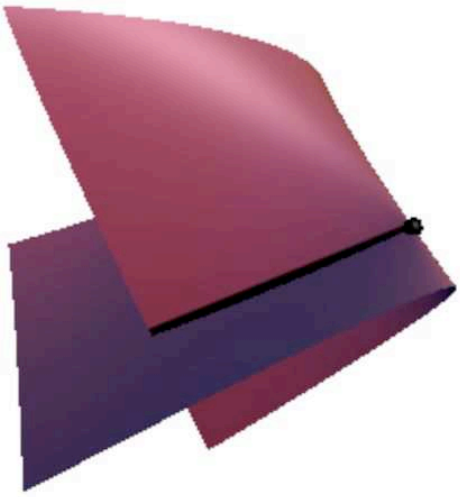


FIG. 3.14: Auto-intersection avec un cusp.

3.5 Topologie d'une courbe implicite

3.5.1 Généralités

Définition 3.3 (Isotopie) : Soient $n \in \mathbb{N}^*$ et $\mathcal{O}_1$ et $\mathcal{O}_2$ deux sous ensembles de $\mathbb{R}^n$. On dit que $\mathcal{O}_1$ et $\mathcal{O}_2$ sont isotopes s'il existe une application $F : \mathcal{O}_1 \times [0,1] \to \mathbb{R}^n$ continue telle que :

- $F(\cdot, 0) = Id_{\mathcal{O}_1}$,
- $F(\mathcal{O}_1, 1) = \mathcal{O}_2$, $\forall t \in [0,1]$,
- $F(\cdot, t)$ réalise un homéomorphisme sur son image.

Définition 3.4 (Topologie dans $\mathbb{R}^2$ ou $\mathbb{R}^3$) : Soit $B \subset \mathbb{R}^2$ ou $\mathbb{R}^3$ (*i.e.* $B = [a_1, b_1] \times [a_2, b_2]$ ou $B = [a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$ avec $a_1, b_1, a_2, b_2, a_3, b_3 \in \mathbb{R}$). La topologie d'une courbe $\mathcal{C}$ dans la boîte B est un graphe de points de $\mathcal{C}$ connectés par des segments et qui est isotope à $\mathcal{C} \cap B$. De même, la topologie d'une surface $\mathcal{S}$ dans B est un complexe simplicial composé de points de $\mathcal{S}$ et qui est isotope à $\mathcal{S} \cap B$.

Dans la pratique, lorsqu'on étudie les courbes et les surfaces dans le but de les visualiser, l'information réellement pertinente est la topologie. A partir de

cette dernière, on peut donner une approximation de l'objet géométrique et affiner la précision de cette approximation selon les besoins. On utilise alors des algorithmes dits de « marching »(voir [4, 5] dans le cas de l'intersection).

Dans ce chapitre on va donc s'intéresser au calcul effectif de la topologie d'une courbe d'intersection entre deux carreaux de surfaces polynomiales. On verra qu'il s'agit en fait d'étudier une courbe implicite dans un espace de dimension 4. L'approche proposée est une méthode par subdivision. Pour plus de clarté, nous commençons par donner une méthode dans le cas d'une courbe implicite du plan.

3.5.2 Courbe implicite plane

Soient $f(x, y) \in \mathbb{R}[x, y]$ un polynôme et $B = [a_1, b_1] \times [a_2, b_2]$ (avec $a_1 \leq b_1$ et $a_2 \leq b_2$) une boîte de $\mathbb{R}^2$. On veut donner un critère pour pouvoir déterminer la topologie de la courbe $\mathcal{C} = \{(x, y) \in B \mid f(x, y) = 0\}$ à partir de $\mathcal{C} \cap \partial B$ (intersection de la courbe avec le bord de la boîte). Ceci est possible si on a, par exemple, pour tout $x_0 \in [a_1, b_1]$ au plus un $y_0 \in [a_2, b_2]$ tel que $f(x_0, y_0) = 0$.
En effet, supposons qu'on se trouve dans une telle situation et qu'on note $p_1, p_2, \ldots, p_{2r-1}, p_{2r}$ la liste des points de $\mathcal{C} \cap \partial B$ triés suivant la coordonnée en x avec répétition d'un point s'il a une intersection multiple avec le bord ∂B. Alors la topologie de $\mathcal{C}$ est donnée par la liste de segments $[p_1, p_2]$, $[p_3, p_4]$, …, $[p_{2r-1}, p_{2r}]$ (voir l'illustration de la situation correspondante sur la figure 3.15).

Proposition 3.1 : Si pour tout $(x, y) \in B$, on a $\partial_y f(x, y) \neq 0$ alors, pour tout $x_0 \in [a_1, b_1]$, on a au plus un $y_0 \in [a_2, b_2]$ tel que $f(x_0, y_0) = 0$.

Démonstration : $\partial_y f(x, y) \neq 0$ pour tout $(x, y) \in B$ donc $\mathcal{C}$ est lisse dans B. Soit $x_0 \in [a_1, b_1]$. Supposons qu'il existe $y_1, y_2 \in [a_2, b_2]$ distincts (avec par exemple $y_1 < y_2$) tels que $f(x_0, y_1) = f(x_0, y_2) = 0$, alors en appliquant le théorème de Rolle à $y \mapsto f(x_0, y)$, on en déduit qu'il existe $y_0 \in]y_1, y_2[$ tel que $\partial_y f(x_0, y_0) = 0$ ce qui contredit l'hypoyhèse. □

Remarque 3.2 :
– Dans la proposition 3.1, on peut permuter les rôles de x et y pour avoir

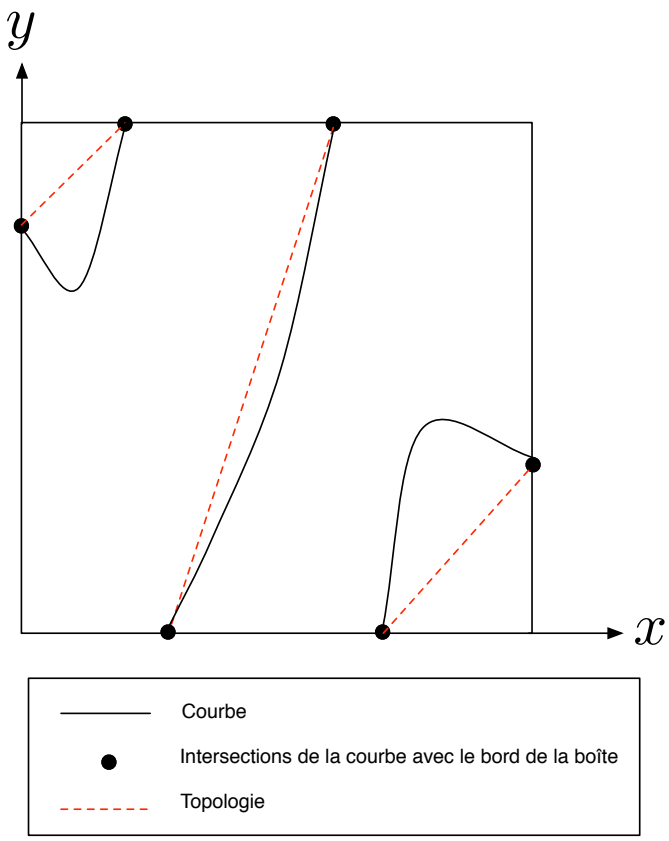


FIG. 3.15: Topologie d'une courbe implicite via son intersection avec le bord de la boîte.

un autre critère permettant de déterminer de manière unique la topologie de la courbe $\mathcal{C}$ à partir de $\mathcal{C} \cap \partial B$.

– Si on n'est pas dans le cadre des hypothèses de la proposition 3.1, alors on peut appliquer (récursivement) une subdivision à la courbe et tester le critère sur les sous-cas générés. On peut donc utiliser ce critère dans le cadre d'une méthode combinée à une subdivision pour avoir la topologie d'une courbe implicite dans une boîte donnée.
– La condition $\partial_y f(x, y) \neq 0$ pour tout $(x, y) \in B$ est assez simple à vérifier. Il suffit de convertir le polynôme $\partial_y f(x, y)$ dans la base de Bernstein associée à B puis de regarder si tous les coefficients de ce polynôme dans cette base ont le même signe. •

3.5.3 Courbe implicite 3D

Le principe précédent peut être étendu au cas du calcul de la topologie d'une courbe implicite dans l'espace. Soient $f(x, y, z)$ et $g(x, y, z) \in \mathbb{R}[x, y, z]$ deux polynômes et $B = [a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$ une boîte. Supposons que les deux surfaces implicites définies respectivement par $f(x, y, z) = 0$ et $g(x, y, z) = 0$ s'intersectent le long d'une courbe dans la boîte B. On désire déterminer la topologie de la courbe $\mathcal{C} = \{(x, y, z) \in B \mid f(x, y, z) = g(x, y, z) = 0\}$ à partir de $\mathcal{C} \cap \partial B$. Comme précédemment, il s'agit de mettre en place un critère d'injectivité. On dispose de la proposition suivante.

Proposition 3.2 : Si $\mathcal{C}$ est lisse dans B et si pour tout $x_0 \in [a_1, b_1]$ on a au plus un $(y_0, z_0) \in [a_2, b_2] \times [a_3, b_3]$ tel que $f(x_0, y_0, z_0) = g(x_0, y_0, z_0) = 0$, alors la topologie de $\mathcal{C}$ est entièrement déterminée à partir de ses points d'intersection avec le bord de B.

Démonstration : Soit $\pi_z : (x, y, z) \mapsto (x, y)$ (projection suivant l'axe z). D'abord, s'il existait p_1 et p_2 deux points distincts de $\mathcal{C}$ tels que $\pi_z(p_1) = \pi_z(p_2) = (x_0, y_0)$, alors p_1 et p_2 seraient tous les deux dans le plan $x = x_0$, ce qui est exclu par hypothèse. Donc $\pi_z(\mathcal{C})$ n'a pas de point double. Par ailleurs, si $x_0 \in [a_1, b_1]$ alors il existe au plus un $(y_0, z_0) \in [a_2, b_2] \times [a_3, b_3]$ tel que $(x_0, y_0, z_0) \in \mathcal{C}$ donc a fortiori au plus un $y_0 \in [a_2, b_2]$ tel que $(x_0, y_0) \in \pi_z(\mathcal{C})$. Ainsi, on se trouve dans la situation des courbes implicites planes décrite dans la section 3.5.2 et donc la topologie de $\pi_z(\mathcal{C})$ est entièrement déterminée à partir de son intersection avec le bord de $[a_2, b_2] \times [a_3, b_3]$. Supposons que cette topologie soit donnée par la liste de segments $[p_1, p_2], [p_3, p_4], \ldots, [p_{2r-1}, p_{2r}]$. Notons pour tout $i \in \{1, 2, \ldots, 2r\}$, (x_i, y_i) les coordonnées de p_i. Alors, par hypothèse, pour tout $i \in \{1, 2, \ldots, 2r\}$, il existe un unique $z_i \in [a_3, b_3]$ tel que le point P_i de coordonnées (x_i, y_i, z_i) appartienne à $\mathcal{C}$. La topologie de $\mathcal{C}$ est donc donnée par la liste de segments $[P_1, P_2], [P_3, P_4], \ldots, [P_{2r-1}, P_{2r}]$. □

A présent, intéressons nous au critère d'injectivité. Soit $x_0 \in [a_1, b_1]$, introduisons les notations $f_{x_0} : (y, z) \mapsto f(x_0, y, z)$ et $g_{x_0} : (y, z) \mapsto g(x_0, y, z)$.

On va s'intéresser à l'application :

$$\phi_{x_0} : \left(\begin{array}{ccc} [a_2, b_2] \times [a_3, b_3] & \longrightarrow & \mathbb{R}^2 \\ (y, z) & \longmapsto & (f_{x_0}(y, z), g_{x_0}(y, z)) \end{array} \right)$$

Proposition 3.3 : Si ϕ_{x_0} n'est pas injective alors on est dans un des deux cas suivants :

1. ϕ_{x_0} admet au moins un point sur $U_{x_0} := [a_2, b_2] \times [a_3, b_3]$ en lequel elle n'est pas localement injective.
2. f_{x_0} ou g_{x_0} ont strictement plus de deux extrema sur le bord de U_{x_0}.

Démonstration : Supposons que ϕ_{x_0} n'est pas injective. Alors, il existe A et B distincts dans $[a_2, b_2] \times [a_3, b_3]$ tels que $\phi_{x_0}(A) = \phi_{x_0}(B)$. Considérons la courbe de niveau $C_k = \{(y, z) \in U_{x_0} \mid f_{x_0}(y, z) = k\}$ avec $k = f_{x_0}(A) = f_{x_0}(B)$. On distingue les deux cas suivants.

Premier cas : A et B sont sur une même composante connexe Γ de C_k.
Alors, on peut construire un chemin de classe C^1 injectif $\gamma : [0, 1] \to C_k$ tel que $\gamma(0) = A$ et $\gamma(1) = B$. Le théorème de Rolle appliqué à $g_{x_0} \circ \gamma$ sur $[0, 1]$ assure qu'il existe $t_0 \in]0, 1[$ tel que $(g_{x_0} \circ \gamma)'(t_0) = 0$. Supposons qu'il existe un voisinage $V_{\gamma(t_0)}$ de $\gamma(t_0)$ dans U_{x_0} tel que la restriction de ϕ_{x_0} à $V_{\gamma(t_0)}$ soit injective (s'il n'existe pas de tel voisinage alors ϕ_{x_0} n'est pas localement injective en $\gamma(t_0)$) et considérons un voisinage V_{t_0} de t_0 dans $]0, 1[$ tel que $\gamma(V_{t_0}) \subset V_{\gamma(t_0)}$. Alors il existe $t_1, t_2 \in V_{t_0}$ distincts tels que $(g_{x_0} \circ \gamma)(t_1) = (g_{x_0} \circ \gamma)(t_2)$. Par ailleurs, $f_{x_0}(\gamma(t_1)) = f_{x_0}(\gamma(t_2)) = k$ donc finalement $\phi_{x_0}(\gamma(t_1)) = \phi_{x_0}(\gamma(t_2))$ avec $\gamma(t_1), \gamma(t_2) \in V_{\gamma(t_0)}$ distincts (par injectivité de γ) ce qui contredit l'injectivité de ϕ_{x_0} sur $V_{\gamma(t_0)}$. On en conclut donc que ϕ_{x_0} n'est pas localement injective en $\gamma(t_0)$.

Deuxième cas : A et B sont sur des composantes connexes distinctes de C_k.
Notons C_k^A (respectivement C_k^B) la composante connexe de C_k contenant A (respectivement B).
Si C_k^A est fermée (*i.e.* forme une boucle), alors le domaine $D \subset U_{x_0}$ de frontière C_k^A est compact donc f_{x_0} atteint sur D un maximum absolu M et un minimum absolu m. Si $M = m = k$, alors f_{x_0} est constante égale à k et par suite $\mathrm{Jac}(\phi_{x_0})$ (déterminant de la matrice jacobienne) est nul sur tout U_{x_0} donc ϕ_{x_0} n'est pas localement injective (théorème d'inversion locale). Sinon si $M \neq k$, alors il existe $(y_0, z_0) \in D$ tel que $f_{x_0}(y_0, z_0) = M$. Le point

(y_0, z_0) n'est pas sur le bord de D (car sinon $f_{x_0}(y_0, z_0) = k$) donc (y_0, z_0) est dans l'intérieur de D et c'est donc un point critique (*i.e.* $\partial_y f_{x_0}(y_0, z_0) = \partial_z f_{x_0}(y_0, z_0) = 0$). Par suite, ϕ_{x_0} n'est pas localement injective en (y_0, z_0). Sinon si $m \neq k$ alors le même raisonnement conduit à la non injectivité de ϕ_{x_0} en un certain point de U_{x_0}.

De même, si C_k^B est fermée alors ϕ_{x_0} n'est pas localement injective.

Supposons à présent que C_k^A et C_k^B ne sont pas fermées. Alors C_k^A et C_k^B intersectent chacune ∂U_{x_0} en deux points distincts *i.e.* il existe quatre points distincts M_1, M_2, M_3 et M_4 sur ∂U_{x_0} tels que $f_{x_0}(M_1) = f_{x_0}(M_2) = f_{x_0}(M_3) = f_{x_0}(M_4) = k = f_{x_0}(A) = f_{x_0}(B)$. En paramétrant le bord de U_{x_0} par un chemin C^1 injectif $\tilde{\gamma} : [0, 1] \to \partial U_{x_0}$, on en déduit qu'il existe t_1, t_2, t_3 et $t_4 \in [0, 1]$ distincts tels que $(f_{x_0} \circ \tilde{\gamma})(t_1) = (f_{x_0} \circ \tilde{\gamma})(t_2) = (f_{x_0} \circ \tilde{\gamma})(t_3) = (f_{x_0} \circ \tilde{\gamma})(t_4)$. Donc, par le théorème de Rolle, $f_{x_0} \circ \tilde{\gamma}$ a au moins trois extrema locaux sur $[0, 1]$ *i.e.* f_{x_0} a au moins trois extrema locaux sur ∂U_{x_0}. □

D'après la proposition 3.3, si on impose les conditions :

1. $\mathrm{Jac}(\phi_{x_0}) \neq 0$ sur tout $[a_2, b_2] \times [a_3, b_3]$ (*i.e.* ϕ_{x_0} partout localement injective).
2. f_{x_0} a au plus deux extrema sur ∂U_{x_0}.

Alors ϕ_{x_0} est injective. Cependant la condition 2 est impliquée par « f_{x_0} a au plus deux extrema sur les coins de ∂U_{x_0} (*i.e.* sur $(a_2, a_3), (a_2, b_3), (b_2, b_3)$ ou (b_2, a_3)) ». On peut donc introduire la proposition suivante :

Proposition 3.4 : ϕ_{x_0} est injective si :

1. ϕ_{x_0} est partout localement injective.
2. $\partial_y f_{x_0} \neq 0$ sur $[a_2, b_2] \times \{a_3\}$ et sur $[a_2, b_2] \times \{b_3\}$.
3. $\partial_z f_{x_0} \neq 0$ sur $\{a_2\} \times [a_3, b_3]$ et sur $\{b_2\} \times [a_3, b_3]$ et a le même signe sur ces deux côtés de U_{x_0}.

Démonstration : Les hypothèses 2 et 3 assurent que f_{x_0} a au plus deux extrema sur les coins de ∂U_{x_0} (voir la figure 3.16). □

Remarque 3.3 :

– Dans la proposition 3.4, on peut permuter les rôles de f_{x_0} et g_{x_0} ainsi que les rôles de y et z ce qui donne donc quatre critères d'injectivité de ϕ_{x_0}.

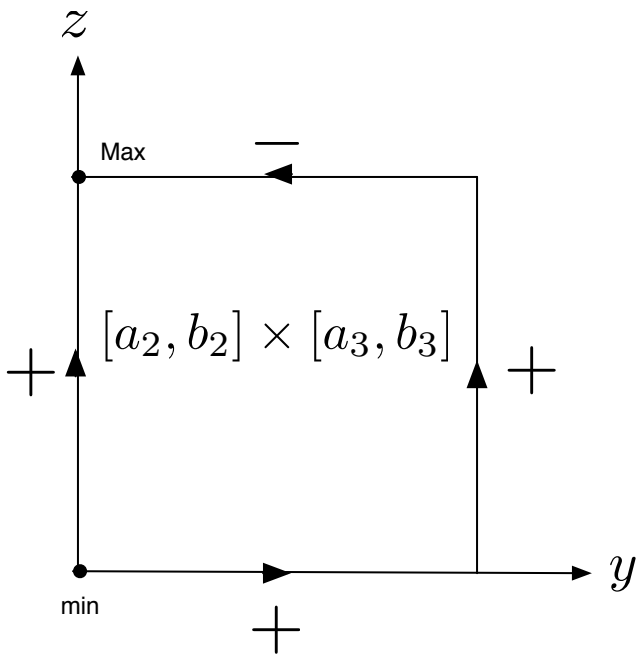


FIG. 3.16: Monotonie de f_{x_0} (exemple de configuration correspondante aux hypothèses de la proposition 3.4).

– Toutes les conditions de non nullité (ainsi que celle de signe) dans la proposition 3.4 concernent des polynômes donc cela se vérifie aisément en travaillant dans la base de Bernstein. •

En résumé, si pour tout $x_0 \in [a_1, b_1]$, ϕ_{x_0} vérifie les hypothèses de la proposition 3.4, alors, en particulier, pour tout $x_0 \in [a_1, b_1]$ on a au plus un $(y_0, z_0) \in [a_2, b_2] \times [a_3, b_3]$ tel que $f(x_0, y_0, z_0) = g(x_0, y_0, z_0) = 0$. La proposition 3.2 permet alors de déterminer la topologie de $\mathcal{C}$ à partir de $\mathcal{C} \cap \partial B$. En combinant ce critère d'injectivité avec une subdivision, on peut déterminer la topologie d'une courbe implicite 3D.

3.5.4 Courbe implicite 4D

A présent, on s'intéresse au cas d'une courbe implicite 4D qui est le cas le plus intéressant car il correspond au cas de la courbe d'intersection de deux carreaux de surfaces paramétrées polynomiales. Dans la suite, on fera l'abus de notation (habituel) qui consiste à confondre une surface avec sa paramétrisation. Soient donc F et G deux carreaux de surfaces paramétrées polynomiales qu'on supposera, sans restreindre la généralité, définies sur $[0,1]^2$.

$$F : \begin{pmatrix} [0,1]^2 & \longrightarrow & \mathbb{R}^3 \\ (s,t) & \longmapsto & F(s,t) \end{pmatrix}$$
$$G : \begin{pmatrix} [0,1]^2 & \longrightarrow & \mathbb{R}^3 \\ (u,v) & \longmapsto & G(u,v) \end{pmatrix}.$$

On suppose que l'intersection $F \cap G$ est une courbe :

$$\mathcal{C} = \left\{ (s,t,u,v) \in [0,1]^4 \mid F(s,t) - G(u,v) = 0 \right\}. \tag{3.17}$$

Si on désire reprendre le même genre de raisonnement que dans les sections 3.5.2 et 3.5.3 pour déterminer la topologie de $\mathcal{C}$, alors il nous faut un critère assurant que, pour tout $s_0 \in [0,1]$, il existe au plus un $(t_0, u_0, v_0) \in [0,1]^3$ tel que $F(s_0,t_0) - G(u_0,v_0) = 0$. Fixons donc un $s_0 \in [0,1]$ et intéressons nous à l'application :

$$\phi_{s_0} : \begin{pmatrix} [0,1]^3 & \longrightarrow & \mathbb{R}^3 \\ (t,u,v) & \longmapsto & F(s_0,t) - G(u,v) \end{pmatrix}.$$

Notons pour la suite :

$$\phi(s,t,u,v) = F(s,t) - G(u,v) = (\phi_1, \phi_2, \phi_3)$$
$$\phi_{s_0}(t,u,v) = (\phi^1_{s_0}, \phi^2_{s_0}, \phi^3_{s_0}).$$

Il s'agit de mettre en place un critère d'injectivité de ϕ_{s_0}. Pour cela, on adopte la même démarche que celle effectuée dans la section 3.5.3. Une

condition nécessaire est donc l'injectivité locale de ϕ_{s_0} *i.e.* :

$$\forall (t,u,v) \in [0,1]^3, \det(\partial_t \phi_{s_0}(t,u,v), \partial_u \phi_{s_0}(t,u,v), \partial_v \phi_{s_0}(t,u,v)) \neq 0$$

Il faut ensuite mettre en place un critère sur les bords de $[0,1]^3$ (analogue des propriétés 2 et 3 de la proposition 3.4). Si ϕ_{s_0} n'est pas injective, alors il existe A et B dans $[0,1]^3$ tels que $\phi_{s_0}(A) = \phi_{s_0}(B)$ *i.e.* tels que pour tout $i \in \{1,2,3\}$, $\phi^i_{s_0}(A) = \phi^i_{s_0}(B)$.
Notons :

$$\begin{aligned} S_1 &:= \left\{ M \in [0,1]^3 \mid \phi^1_{s_0}(M) = \phi^1_{s_0}(A) \right\} \\ C_{1,2} &:= \left\{ M \in [0,1]^3 \mid \phi^1_{s_0}(M) = \phi^1_{s_0}(A) \text{ et } \phi^2_{s_0}(M) = \phi^2_{s_0}(A) \right\}. \end{aligned}$$

Proposition 3.5 : Si A et B sont sur une même composante connexe de $C_{1,2}$, alors ϕ_{s_0} n'est pas localement injective.

Démonstration : Si $C_{1,2}$ admet au moins un point singulier M_0 alors la matrice jacobienne de l'application :
$(t,u,v) \mapsto \left(\phi^1_{s_0}(t,u,v) - \phi^1_{s_0}(A), \phi^2_{s_0}(t,u,v) - \phi^2_{s_0}(A)\right)$ en M_0 n'est pas de rang maximum et donc la matrice jacobienne de ϕ_{s_0} en M_0 non plus. Par suite, ϕ_{s_0} n'est pas localement injective en M_0.
Supposons donc que $C_{1,2}$ est lisse. Alors, on peut construire un chemin de classe C^1 injectif $\gamma : [0,1] \to C_{1,2}$ tel que $\gamma(0) = A$ et $\gamma(1) = B$. Le théorème de Rolle appliqué à $(\phi^3_{s_0} \circ \gamma)$ sur $[0,1]$ assure qu'il existe $t_0 \in]0,1[$ tel que $(\phi^3_{s_0} \circ \gamma)'(t_0) = 0$ *i.e.* tel que $\nabla \phi^3_{s_0}(\gamma(t_0)) \cdot \gamma\prime(t_0) = 0$. Par ailleurs, $\gamma\prime(t_0)$ est non nul et tangent à $C_{1,2}$ en $\gamma(t_0)$ donc $\nabla \phi^1_{s_0}(\gamma(t_0)) \cdot \gamma\prime(t_0) = \nabla \phi^2_{s_0}(\gamma(t_0)) \cdot \gamma\prime(t_0) = 0$. Par suite, le vecteur non nul $\gamma\prime(t_0)$ est dans le noyau de matrice jacobienne de ϕ_{s_0} en $\gamma(t_0)$ donc ϕ_{s_0} n'est pas localement injective en $\gamma(t_0)$. □

Proposition 3.6 : Si A et B sont sur des composantes connexes distinctes de $C_{1,2}$ notées respectivement C_A et C_B et si l'une d'entre elles est fermée (*i.e.* forme une boucle), alors ϕ_{s_0} n'est pas localement injective.

Démonstration : Il suffit de reprendre la démonstration de la proposition 3.3. □

Les propositions 3.5 et 3.6 permettent de ne considérer que le cas où A et B sont sur des composantes connexes distinctes, notées respectivement C_A et C_B, qui coupent chacune deux fois le bord du cube $[0,1]^3$. Dans ce cas, on aurait quatre points distincts P_1, P_2, P_3 et P_4 sur ce bord (constituant $C_{1,2} \cap \partial[0,1]^3$). Il s'agit donc d'élaborer un critère pour éviter ce cas. Pour cela, on va introduire deux types de conditions à savoir des monotonies adaptées de $\phi^1_{s_0}$ sur les arêtes de $[0,1]^3$ et de $\phi^2_{s_0}$ le long de ∂S_1.

Monotonie de $\phi^1_{s_0}$ sur les arêtes de $[0,1]^3$. Pour commencer, on peut imposer des conditions de monotonie de $\phi^1_{s_0}$ sur les arêtes de la boîte $[0,1]^3$. On peut, par exemple, imposer à $\phi^1_{s_0}$ d'être strictement croissante sur toutes les arêtes de $[0,1]^3$ comme indiqué sur la figure 3.17. Si on a une telle configuration, alors, $\phi^1_{s_0}$ ne s'annulera qu'une fois au plus le long de n'importe quel chemin sur les arêtes de $[0,1]^3$ joignant O à I.

Proposition 3.7 : Sous la condition de monotonie décrite précédemment, la surface implicite S_1, (qui a pour équation $\phi^1_{s_0}(t,u,v) = \phi^1_{s_0}(A)$), est connexe.

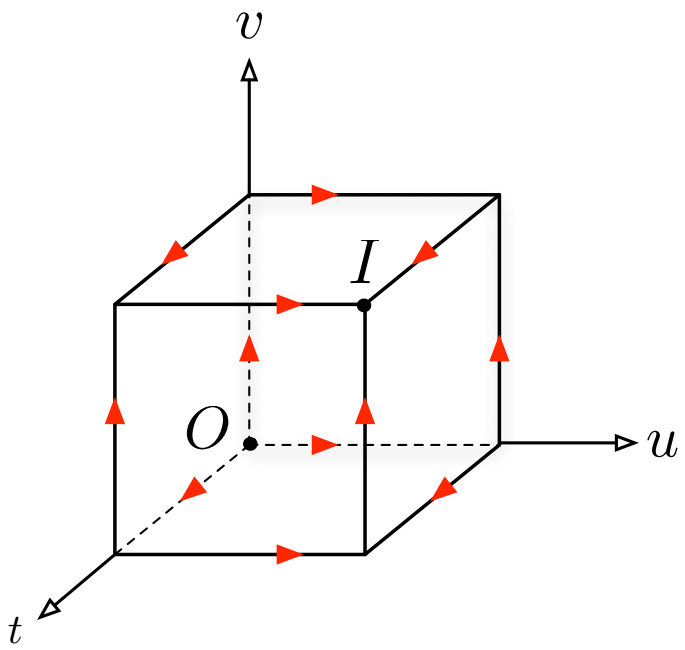


FIG. 3.17: Exemple de monotonie de $\phi^1_{s_0}$ sur les arêtes de $[0,1]^3$.

Démonstration : Les différentes configurations possibles sont illustrées sur la figure 3.18). □

Il convient de signaler que ce n'est pas la seule possibilité de monotonie de $\phi^1_{s_0}$. En fait, on peut se donner un catalogue de 8 configurations qui permettent de s'assurer que S_1 est connexe. Pour les construire, on choisit, pour chaque vecteur de coordonnées, un sens de direction de stricte monotonie (dans le sens du vecteur ou dans le sens opposé) pour les 4 arêtes de même direction que ce vecteur. Cela donne bien en tout $2^3 = 8$ choix possibles.

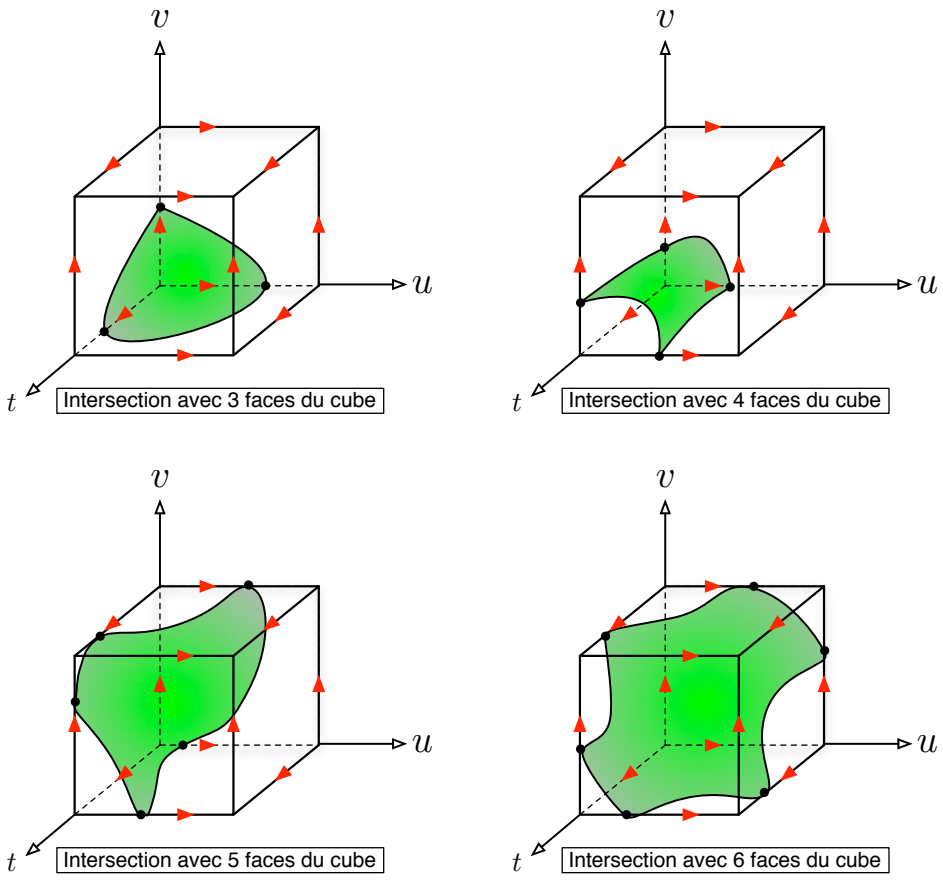


FIG. 3.18: Configurations possibles de la surface S_1 dans $[0,1]^3$ sous contrainte de monotonie $\phi^1_{s_0}$ sur les arêtes de $[0,1]^3$.

Concrètement, toutes ces conditions se résument à tester si :

$$\begin{cases} \partial_t \phi^1_{s_0}(t, u_0, v_0), \text{ avec } u_0, v_0 \in \{0,1\}, \text{ sont tous strictement de même signe.} \\ \partial_u \phi^1_{s_0}(t_0, u, v_0), \text{ avec } t_0, v_0 \in \{0,1\}, \text{ sont tous strictement de même signe.} \\ \partial_v \phi^1_{s_0}(t_0, u_0, v), \text{ avec } t_0, u_0 \in \{0,1\}, \text{ sont tous strictement de même signe.} \end{cases}$$

Remarque 3.4 :

- Vu la forme de $\phi^1_{s_0}$ (variables séparées), les expressions précédentes sont en réalité moins nombreuses qu'il n'y paraît et se simplifient. La première condition ne comporte qu'une seule expression dans laquelle apparaît s_0 et t, la seconde (respectivement la troisième) n'en comporte que deux dans lesquelles apparaissent seulement u (respectivement v) et pas s_0.
- Remarquons également que l'on aurait pu choisir $\phi^2_{s_0}$ ou $\phi^3_{s_0}$ à la place de $\phi^1_{s_0}$ pour imposer la monotonie sur les arêtes de $[0,1]^3$. •

Monotonie de $\phi^2_{s_0}$ le long de ∂S_1. Il s'agit à présent d'étudier les différentes configurations qui précèdent et d'imposer des conditions de monotonie de $\phi^2_{s_0}$ le long du bord de S_1 de sorte que $C_{1,2}$ ne coupe que deux fois ce bord (vu que $C_{1,2} \subset S_1$, si $C_{1,2}$ n'intersecte que deux fois le bord de S_1 alors il n'intersectera que deux fois le bord de $[0,1]^3$). Pour ce faire, on va utiliser le lemme général suivant :

Lemme 3.3 : Soient f une fonction de classe $\mathcal{C}^1$ sur un ouvert convexe $\mathcal{U}$ de $\mathbb{R}^2$ et h un vecteur non nul de $\mathbb{R}^2$. Si pour tout $u \in \mathcal{U}$, on a $\nabla f(u) \cdot h > 0$, alors f est strictement croissante sur $\mathcal{U}$ suivant la direction h (*i.e.* $\forall u \in \mathcal{U}$ et $\forall \epsilon > 0$ tels que $(u + \epsilon\, h) \in \mathcal{U}$, on a $f(u + \epsilon\, h) > f(u)$). ◊

Démonstration : Soit $u_0 \in \mathcal{U}$ tel que $(u_0 + \epsilon h) \in \mathcal{U}$. Alors, $f(u_0 + \epsilon h) - f(u_0) = \int_0^1 \phi(t)\, dt$ avec $\phi(t) = \nabla f(u_0 + t\epsilon h) \cdot h$ qui est positive. □

Remarque 3.5 :

- En remplaçant f par $-f$ dans le lemme 3.3, on en déduit que si, pour tout $u \in \mathcal{U}$, on a $\nabla f(u) \cdot h < 0$, alors f est strictement décroissante sur $\mathcal{U}$ suivant la direction h.
- Une autre remarque, qui va nous servir pour la suite, est que si on se

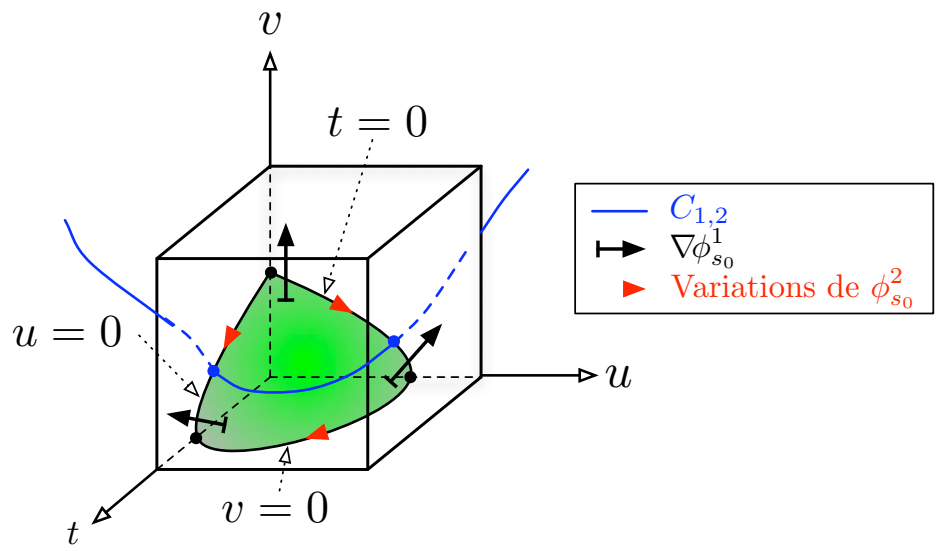


FIG. 3.19: Exemple de monotonie de $\phi^2_{s_0}$ le long du bord de S_1 dans le cas où S_1 intersecte 3 faces du cube.

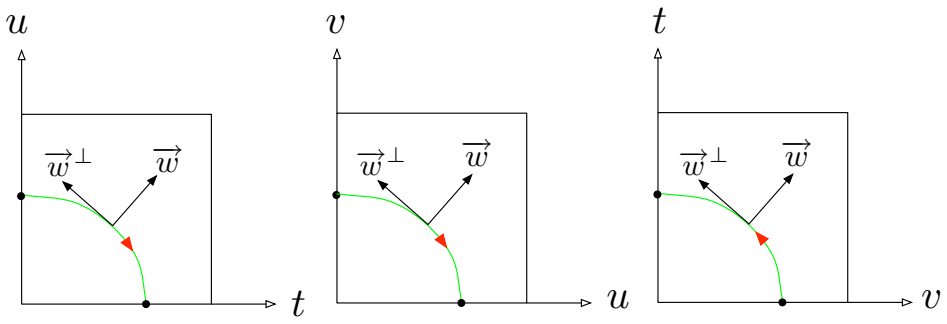


FIG. 3.20: Traces de S_1 sur les faces du cube (cas correspondant à la figure 3.19).

place dans le plan et qu'on se donne un vecteur non nul $\vec{w} = (a, b)$, alors le vecteur $\vec{w}^\perp := (-b, a)$ est normal à $\vec{w}$ et plus précisément l'angle orienté $(\widehat{\vec{w}, \vec{w}^\perp})$ vaut $\pi/2$. •

A présent, on peut imposer une stricte monotonie de $\phi^2_{s_0}$ le long du bord de S_1 de sorte que $C_{1,2}$ ne puisse couper qu'au plus deux fois ce bord comme illustré sur la figure 3.19. Pour cela, on commence par orienter S_1 avec le champ de vecteurs $\nabla\phi^1_{s_0}$. On obtient alors une orientation sur le bord ∂S_1

de S_1 (qui est l'intersection de S_1 avec les faces du cube). Sur chaque face, on projette $\nabla\phi^1_{s_0}$ et on note $\overrightarrow{w}$ ce vecteur. $\overrightarrow{w}$ nous permet alors d'orienter le bord ∂S_1 de S_1 (*i.e.* l'intersection de S_1 avec les faces du cube). Ensuite, on impose une direction de monotonie de la restriction de $\phi^2_{s_0}$ à ∂S_1 sur chaque face du cube. Pour illustrer ce procédé, on a représenté sur la figure 3.20, dans les trois plans de coordonnées, les monotonies de l'exemple de la figure 3.19.

Dans ce cas précis, pour imposer par exemple la monotonie désirée dans le plan (u, v) (qui correspond au dessin du milieu sur la figure 3.20), on commence par projeter le vecteur $\nabla\phi^1_{s_0}$ dans ce plan pour obtenir $\overrightarrow{w} = (\partial_u\phi^1_{s_0}(0,\cdot,\cdot), \partial_v\phi^1_{s_0}(0,\cdot,\cdot))$. Puis on impose la décroissance stricte de $(u, v) \mapsto \phi^2_{s_0}(0, u, v)$ dans la direction $\overrightarrow{w}^\perp$. En appliquant le lemme 3.3, cela revient donc à imposer :

$$\forall (u,v) \in [0,1]^2, \begin{pmatrix} \partial_u\phi^2_{s_0}(0,u,v) \\ \partial_v\phi^2_{s_0}(0,u,v) \end{pmatrix} \cdot \begin{pmatrix} -\partial_v\phi^1_{s_0}(0,u,v) \\ \partial_u\phi^1_{s_0}(0,u,v) \end{pmatrix} < 0.$$

Choix des contraintes de monotonie. On donne à présent les choix des conditions suffisantes telles que $C_{1,2}$ ait au plus deux points d'intersection avec ∂S_1. On considère d'abord, le cas où S_1 intersecte les 6 faces du cube $[0, 1]^3$. Explicitement, on impose, par projection, une monotonie de $\phi^2_{s_0}$ le long de ∂S_1 de sorte que $\#(C_{1,2} \cap \partial S_1) \leq 2$. Dans les autres cas, on se contente d'omettre les conditions correspondantes. La figure 3.21 donne un exemple de type de monotonie possible dans le cas où on a intersection sur les six faces avec les conséquences sur les autres configurations, les projections correspondantes sont représentées sur la figure 3.22.

Bien entendu, l'exemple de la figure 3.21 n'est pas le seul choix possible de monotonie. Pour dresser une liste exhaustive des cas, nous allons introduire une représentation schématique de la situation. Le bord de S_1 est isotope à un hexagone $\{M_1, \ldots, M_6\}$ comme illustré sur la figure 3.23. Plus précisément, $M_1, \ldots, M_6$ sont les points d'intersection de S_1 avec les arêtes du cube $[0, 1]^3$ et les arêtes de l'hexagone représentent chacune une courbe d'intersection de S_1 avec une des 6 faces du cube. Chaque type de monotonie

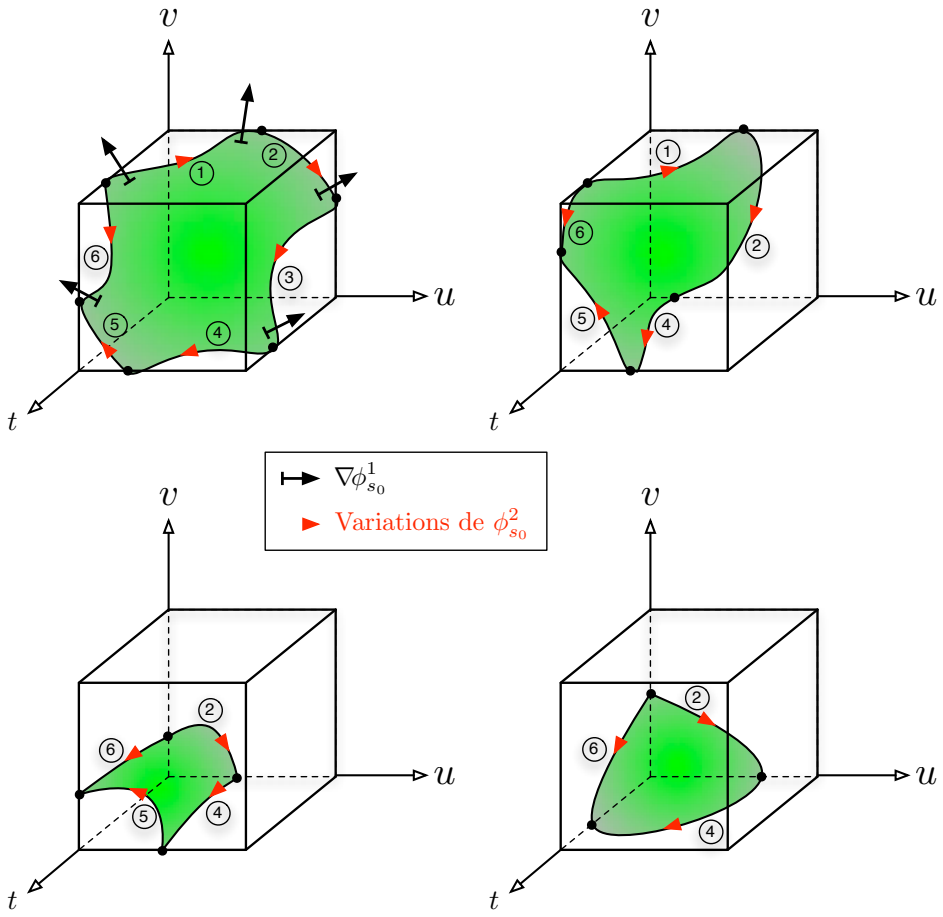


FIG. 3.21: Exemple de monotonie de $\phi^2_{s_0}$ le long du bord de S_1 dans le cas où S_1 intersecte 6 faces du cube avec les conséquences sur les autres cas.

peut alors être décrit sur ce schéma (comme par exemple sur la figure 3.24). Une condition suffisante de monotonie est donnée par le choix d'un point initial M_I et d'un point final M_F parmi $\{M_1, \ldots, M_6\}$, avec éventuellement $M_I = M_F$, tels que $\phi^2_{s_0}$ soit strictement monotone le long du chemin joignant M_I à M_F. Cela implique clairement que $\phi^2_{s_0}$ s'annule au plus deux fois sur ∂S_1.

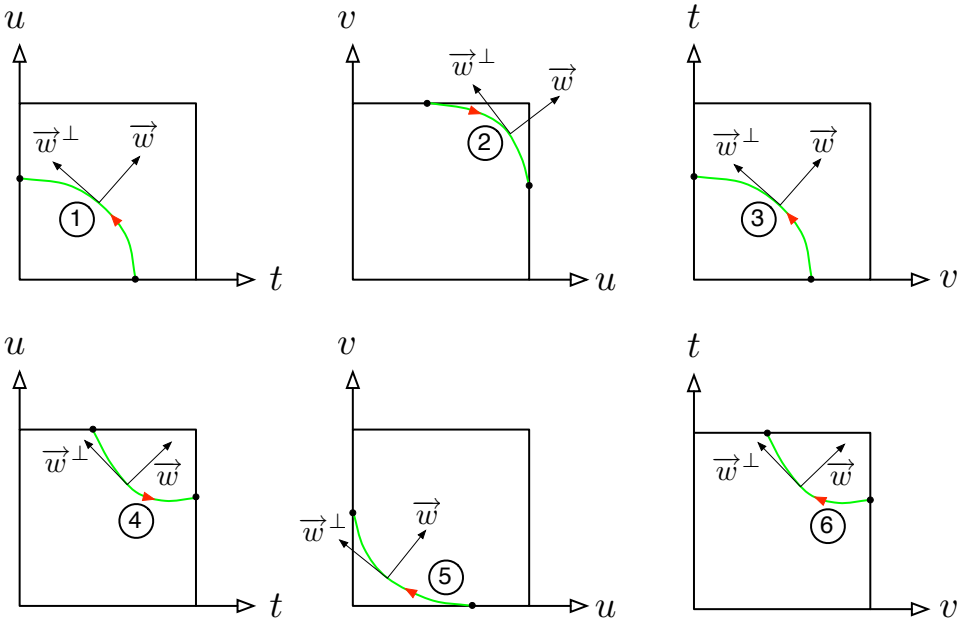


FIG. 3.22: Traces de S_1 sur les faces du cube (cas correspondant à la figure 3.21).

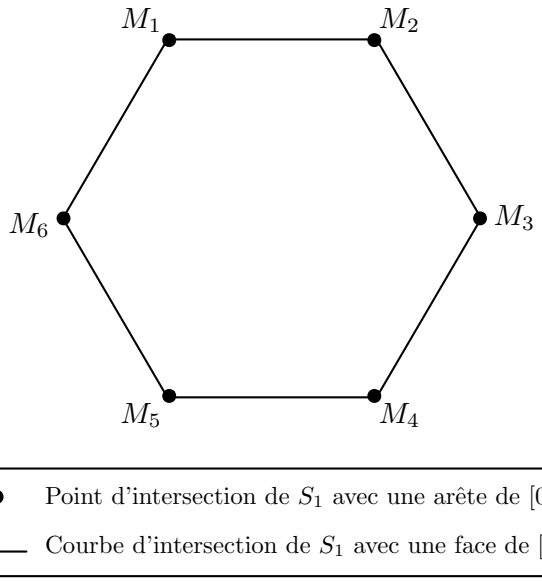


FIG. 3.23: Schéma de la situation géométrique d'intersection de S_1 avec le bord de $[0,1]^3$.

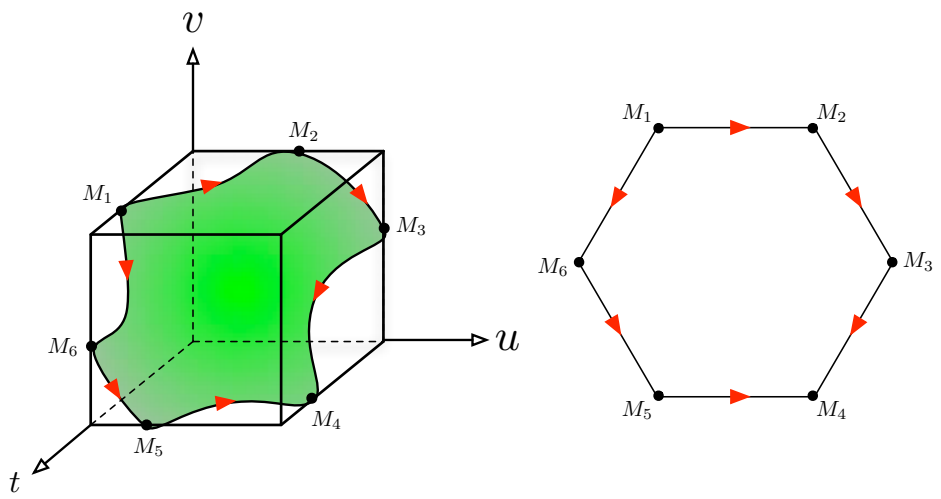


FIG. 3.24: Traduction d'une configuration géométrique de monotonie sur le schéma.

Remarque 3.6 : En résumé, le critère d'injectivité qui vient d'être décrit a été élaboré à partir de plusieurs choix fixés au départ. En effet, on a imposé :

1. L'injectivité locale de ϕ_{s_0}, mais on aurait également pu choisir de fixer la variable t, u ou v à la place de s.
2. La monotonie de $\phi^1_{s_0}$ sur les arêtes de $[0,1]^3$, mais on aurait également pu choisir $\phi^2_{s_0}$ ou $\phi^3_{s_0}$ à la place de $\phi^1_{s_0}$.
3. La monotonie de $\phi^2_{s_0}$ le long de ∂S_1 dans le pire des cas, mais on aurait également pu choisir $\phi^3_{s_0}$ à la place de $\phi^2_{s_0}$.

Cette diversité des choix possibles est intéressante pour rendre l'implémentation plus efficace (voir le chapitre 3.6). •

3.6 Aspects algorithmiques du calcul de topologie

Dans cette section, on apporte quelques explications sur l'implémentation de la méthode de calcul de topologie de la courbe d'intersection expliquée dans la section 3.5.4. L'approche algorithmique est une méthode par subdivision. Les algorithmes de subdivision ont tous un schéma identique (technique de type « diviser pour régner »), c'est la raison pour laquelle, on se contentera d'expliquer dans les détails la méthode pour les courbes implicites 4D. Les

cas 2D et 3D étant sensiblement identiques avec des structures de données et des critères de subdivision plus simples, on les passera sous silence.

3.6.1 Structure de données hexatree

L'objet sur lequel on travaille est la courbe d'intersection $\mathcal{C}$ définie par 3.17. Etant donné qu'on se situe dans un espace à quatre dimensions et que l'on adopte une approche par subdivision, il nous faut partitioner cet espace. L'intuition naturelle est d'utiliser un « hexatree » *i.e.* une structure de données d'arbre dans lequel chaque noeud admet 16 fils. Cette structure permet de généraliser les notions de quadtree et octree qui sont largement utilisés pour partitioner les espaces à respectivement 2 et 3 dimensions. Cet hexatree contiendra au départ (à la racine) une boîte $[a_1, b_1] \times [a_2, b_2] \times [a_3, b_3] \times [a_4, b_4]$, dans laquelle on considère notre courbe d'intersection $\mathcal{C}$, ainsi que les points d'intersection de $\mathcal{C}$ avec le bord de cette boîte (ce sont explicitement des systèmes polynomiaux de 3 polynômes à 3 variables). Puis on créera au fur et à mesure les fils de cette racine en subdivisant à chaque fois au centre de la boîte courante et en distribuant de manière judicieuse (*i.e.* en évitant des calculs superflus) les points d'intersection qui sont hérités et en calculant les nouveaux points d'intersection éventuels apparus avec ces sous-boîtes. Il convient également de mentionner que l'on réserve aussi un champ pour stocker la topologie dans chaque noeud de cet arbre. Concrètement, dans chaque noeud on pourra stocker non seulement des points (d'intersection) mais également une liste de segments constituée par ces points (qui représentera la topologie dans le noeud courant). Au niveau des feuilles de cet arbre, on aura soit une intersection vide (ou alors une taille inférieure à une tolérance ϵ fixée au départ) soit une famille de segments, constitués par des points d'intersection de la courbe avec le bord de cette cellule, isotope à la courbe $\mathcal{C}$ (dans la boîte courante).

Pour encoder judicieusement les 16 fils d'un noeud donné, notons $c_i = \frac{a_i+b_i}{2}$ (pour $i \in \{1, 2, 3, 4\}$). Chaque fils portera un numéro (de 0 à 15) pour l'identifier clairement. Il suffit de regarder ce numéro en binaire. Il s'écrira sous la forme $\alpha_1\alpha_2\alpha_3\alpha_4$ avec les $\alpha_i = 0$ ou 1. Pour $i \in \{1, 2, 3, 4\}$, si $\alpha_i = 0$ alors la i-ème composante de la boîte considérée sera $[a_i, c_i]$ (partie gauche) et si $\alpha_i = 1$ alors ce sera $[c_i, b_i]$ (partie droite). Par exemple le fils numéro 12

sera celui qui correspond à la sous boîte $[c_1, b_1] \times [c_2, b_2] \times [a_3, c_3] \times [a_4, c_4]$ car en binaire 12 s'écrit 1100.

Le critère d'injectivité élaboré dans la section 3.5.4 est implémenté dans une fonction de test (appelé `regulier`). Si cette fonction retourne « faux » sur une cellule, alors cette dernière est subdivisée. On obtient alors l'algorithme de subdivision 3.1. L'implémentation, quant à elle, utilise la librairie Synaps [50] et a été intégrée au modeleur géométrique Axel [12] dont une description est donnée dans le chapitre 4.

3.6.2 Algorithme de subdivision

L'algorithme 3.1 décrit le calcul de la topologie de $\mathcal{C}$, par subdivision, dans une boîte donnée. Certaines autres fonctionnalités sont nécessaires et leurs descriptions sont données dans la suite.

Algorithme 3.1 : Topologie par subdivision en 4D.

`topologie`$(\mathcal{C}, B, \epsilon)$
Données : La courbe $\mathcal{C}$, une boîte de la forme
$[a_1, b_1] \times [a_2, b_2] \times [a_3, b_3] \times [a_4, b_4]$ et une tolérance ϵ.
Résultat : Une liste de segments de $\mathbb{R}^4$ représentant la topologie.
Créer l'hexatree $\mathcal{H}$;
Initialiser la racine de $\mathcal{H}$ avec B et les points d'intersection $\mathcal{C} \cap \partial B$;
Créer une liste de noeuds $\mathcal{L}$;
$\mathcal{L} \longleftarrow$ `racine`$(\mathcal{H})$;
tant que $\mathcal{L} \neq \emptyset$ **faire**
  Prendre le premier élément n de $\mathcal{L}$ (et le retirer de $\mathcal{L}$);
  si `regulier`$(\mathcal{C}, n)$ **alors**
    $n \longleftarrow$ `topologieReguliere`$(\mathcal{C}, n)$;
  sinon si *la boîte courante est de taille* $\geq \epsilon$ **alors**
    $\mathcal{L} \longleftarrow$ `subdivision`$(\mathcal{C}, n)$;
  sinon
    On génère une topologie étoilée par rapport au centre de la boîte (cas où on stop la subdivision);
  fin
fin
retourner `fusion(racine`$(\mathcal{H})$`)`;

Fonction `regulier`. Cette fonction est le critère d'injectivité décrit dans le section 3.5.4. En fait, on dispose de quatre tests différents et chacun d'entre eux correspond au choix de la variable que l'on fixe (s, t, u ou v). Si l'un de ces tests est vérifié, alors on appelle la fonction `topologieReguliere` correspondante. Pour résumer, le fonctionnement de `regulier` est décrit par l'algorithme 3.2.

Algorithme 3.2 : Critère d'injectivité.

`regulier`($\mathcal{C}, n$)
si ϕ *est localement injective dans* n **alors**
  si ϕ_1 *a la monotonie adéquate sur les arêtes de* n **alors**
    si ϕ_2 *ou* ϕ_3 *a la monotonie adéquate sur* ∂S_1 **alors**
      retourner *vrai*;
    sinon
      retourner *faux*;
    fin
  sinon si ϕ_2 *a la monotonie adéquate sur les arêtes de* n **alors**
    si ϕ_1 *ou* ϕ_3 *a la monotonie adéquate sur* ∂S_2 **alors**
      retourner *vrai*;
    sinon
      retourner *faux*;
    fin
  sinon si ϕ_3 *a la monotonie adéquate sur les arêtes de* n **alors**
    si ϕ_1 *ou* ϕ_2 *a la monotonie adéquate sur* ∂S_3 **alors**
      retourner *vrai*;
    sinon
      retourner *faux*;
    fin
  sinon
    retourner *faux*;
  fin
sinon
  retourner *faux*;
fin

Fonction `topologieReguliere`. Si l'un des quatre tests de régularité `regulier` est vérifié, alors la topologie de $\mathcal{C}$ est connue et le rôle de la fonc-

tion `topologieReguliere` est justement de la retourner. Concrètement, il suffit de connecter les points d'intersection de $\mathcal{C}$ avec les bords du noeud courant. En fait, on dispose de quatre fonctions `topologieReguliere` distinctes. Elles correspondent aux choix de la variable fixée (s, t, u ou v). Par exemple, si la variable fixée est $s = s_0$, alors on aura, dans le noeud courant, une liste contenant un nombre pair de points $p_1, p_2, \ldots, p_{2k-1}, p_{2k}$ (on répète un point s'il est de multiplicité paire) sur le bord du noeud. Ces points sont triés par rapport à leur composante en s (si s est la variable fixée) et la topologie est alors donnée par la liste de segments $[p_1, p_2], \ldots, [p_{2k-1}, p_{2k}]$.

Fonction `subdivision`. Le rôle de cette fonction est de subdiviser au milieu du noeud courant en créant ses 16 fils. Elle se charge aussi de distribuer les points d'intersection qui sont hérités et calcule les nouveaux points d'intersection éventuels apparus dans les fils conformément à la description qui a été faite dans la section 3.6.1.

Fonction `fusion`. Cette fonction est appelée lorsque la construction de l'hexatree $\mathcal{H}$ est achevée. Plus précisément, chaque feuille de $\mathcal{H}$ contient la topologie de $\mathcal{C}$ dans la boîte correspondante. Le rôle de `fusion` est de donner la topologie de $\mathcal{C}$ dans la boîte initiale B. L'implémentation de `fusion` consiste juste à assembler récursivement les topologies entre les 16 fils de chaque noeud. Pour un noeud n donné et un entier $i \in \{0, \ldots, 15\}$, on notera `fils(`i`,` n`)` le i-ème fils du noeud conformément à la description qui a été faite dans la section 3.6.1. Par ailleurs, pour deux listes l_1 et l_2 de segments de $\mathbb{R}^4$ données, on notera `assembler`(l_1, l_2) la liste de segments de $\mathbb{R}^4$ composée de tous les segments de l_1 et l_2. L'algorithme 3.3 décrit alors le fonctionnement de `fusion`.

3.6.3 Composantes connexes et boucles

Il convient de remarquer que la topologie de $\mathcal{C}$ produite par l'algorithme 3.1 permet d'identifier facilement les composantes connexes de cette topologie. En effet, le résultat de cet algorithme est une liste de segments de $\mathbb{R}^4$ représentant la topologie de $\mathcal{C}$.

Algorithme 3.3 : Topologie par subdivision en 4D.

```
fusion(n)
Données : Un noeud d'hexatree comme définit dans la section 3.6.1
Résultat : Une liste de segments de ℝ⁴
si n est une feuille alors
  retourner La liste de segments contenue dans n;
sinon
  retourner
  assembler(
    assembler(
      assembler(
        assembler(fusion(fils(0, n)), fusion(fils(1, n)))
        assembler(fusion(fils(2, n)), fusion(fils(3, n))))
      assembler(
        assembler(fusion(fils(4, n)), fusion(fils(5, n)))
        assembler(fusion(fils(6, n)), fusion(fils(7, n)))))
    assembler(
      assembler(
        assembler(fusion(fils(8, n)), fusion(fils(9, n)))
        assembler(fusion(fils(10, n)), fusion(fils(11, n))))
      assembler(
        assembler(fusion(fils(12, n)), fusion(fils(13, n)))
        assembler(fusion(fils(14, n)), fusion(fils(15, n))))));
fin
```

Supposons que cette liste se présente sous la forme $\{[p_1, p_2], \ldots, [p_{2k-1}, p_{2k}]\}$ ($k \in \mathbb{N}^*$). Si pour un certain $i \in \{1, \ldots, k-1\}$, on a $p_{2i} \neq p_{2i+1}$, alors cela signifie que le segment $[p_{2i-1}, p_{2i}]$ appartient à une composante connexe de la topologie de $\mathcal{C}$ et $[p_{2i+1}, p_{2i+2}]$ à une autre. En utilisant ce constat simple, on accède facilement aux différentes composantes connexes voulues. Une analyse similaire sur chacune des composantes connexes permet de détecter celles qui forment des boucles.

3.7 Remontée d'une topologie dans l'espace

La section 3.5.4 donne une méthode pour calculer la topologie d'une courbe $\mathcal{C}$ dans $\mathbb{R}^4$ définie par $F(s,t) = G(u,v)$ (avec $(s,t,u,v) \in [0,1]^4$ par exemple).

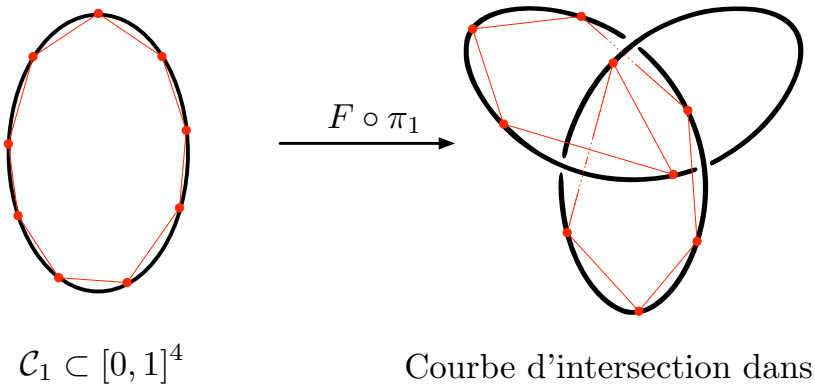


FIG. 3.25: Image d'une boucle avec une structure de noeud.

Fixons les notations suivantes :

$$\pi_1 : \begin{pmatrix} [0,1]^4 & \longrightarrow & \mathbb{R}^2 \\ (s,t,u,v) & \longmapsto & (s,t) \end{pmatrix}$$
$$\pi_2 : \begin{pmatrix} [0,1]^4 & \longrightarrow & \mathbb{R}^2 \\ (s,t,u,v) & \longmapsto & (u,v) \end{pmatrix}.$$

Le lieu d'intersection Γ dans $\mathbb{R}^3$ des deux carreaux de surfaces polynomiales paramétrés par F et G est l'image par $F \circ \pi_1$ (ou $G \circ \pi_2$) de $\mathcal{C}$. La méthode de la section 3.5.4 garantie (relativement par rapport à une tolérance ϵ donnée) la topologie de $\mathcal{C}$. Concrètement, on calcule une famille de segments de $[0,1]^4$ isotope à $\mathcal{C}$. Cela implique que l'image par $F \circ \pi_1$ (ou $G \circ \pi_2$) d'une composante connexe $\mathcal{C}_1$ de $\mathcal{C}$ est connexe. Cependant, si $\mathcal{C}_1$ est une boucle dans $[0,1]^4$, alors cette image est une boucle dans $\mathbb{R}^3$ mais on ne peut connaître sa structure de noeud. Plus précisément, si $\mathcal{C}$ a plusieurs composantes connexes qui forment des boucles dans $[0,1]^4$, alors leurs images par $F \circ \pi_1$ (ou $G \circ \pi_2$) dans $\mathbb{R}^3$ peuvent s'entrelacer (comme dans l'exemple des anneaux olympiques). Si $\mathcal{C}_1$ est donnée par une discrétisation (par des segments) trop grossière, alors la structure de noeud (et les entrelacements) peut être manquée dans l'image par $F \circ \pi_1$ (ou $G \circ \pi_2$) de cette approximation. On peut avoir la situation illustrée sur la figure 3.25.

De même, la topologie de la projection $\widehat{\mathcal{C}}$ de $\mathcal{C} \subset [0,1]^4$ sur $[0,1]^2$ par π_1,

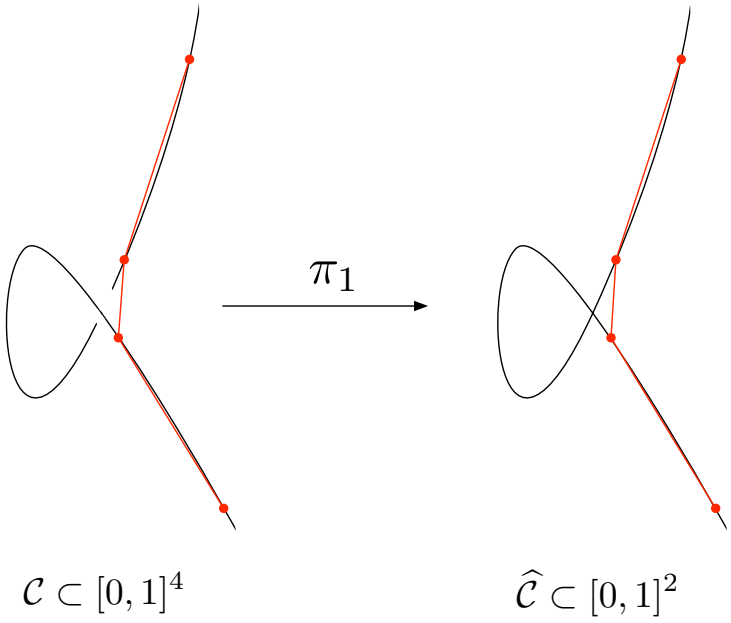


FIG. 3.26: Auto-intersection manquée par projection.

peut ne pas être déterminée par une discretisation de $\mathcal{C}$ et ce, même si elle est suffisamment fine pour déterminer la topologie de $\mathcal{C}$ dans $[0,1]^4$. L'exemple de la figure 3.26 montre un point d'auto-intersection manqué lors de la projection. Pour prendre en considération ces phénomènes, la méthode décrite dans la section 3.5.4 peut-être adaptée avec un critère de subdivision plus fin que l'on va décrire à présent.

La méthode de calcul de topologie décrite dans les sections 3.5.4 et 3.6 permet d'isoler les singularités de Γ dans des boîtes suffisamment petites (relativement par rapport à une tolérance fixée au départ). On se propose de déterminer la topologie de Γ en dehors de ces boîtes singulières. Plus précisément, on considère que des branches entrant dans une telle boîte s'intersectent et forment un point singulier. En dehors de ces cas, tous les autres points sont considérés comme lisses. Supposons que $\mathcal{C}$ admette un certain nombre $k \in \mathbb{N}^*$ de composantes connexes sous forme de boucles. Soient $\mathcal{C}_1, \ldots, \mathcal{C}_k$ ces composantes (qu'on peut détecter grâce à ce qui est

expliqué dans la section 3.6.3) et $\Gamma_1, \ldots, \Gamma_k$ leurs images respectives par $F \circ \pi_1$ (ou $G \circ \pi_2$) dans $\mathbb{R}^3$.

3.7.1 Isolation des branches par des boîtes

Rappelons que chaque noeud de l'hexatree $\mathcal{H}$ (décrit dans la section 3.6.1) contient une boîte de $\mathbb{R}^4$ et la topologie de $\mathcal{C}$ dans cette boîte. Soit n un noeud de $\mathcal{H}$ et $\mathcal{B}_n$ la boîte correspondante. On peut construire une boîte B_n de $\mathbb{R}^3$ contenant entièrement la portion de Γ qui correspond à $\mathcal{B}_n$ *i.e.* l'image de $\mathcal{C} \cap \mathcal{B}_n$ par $F \circ \pi_1$ (ou $G \circ \pi_2$). Pour cela, on exprime $F(s,t)$ (respectivement $G(u,v)$) dans la base de Bernstein associée à $\pi_1(\mathcal{B}_n)$ (respectivement à $\pi_2(\mathcal{B}_n)$). On définit alors $B_n = B_F \cap B_G$ avec B_F (respectivement B_G) la boîte englobante construite à partir des points de contrôle de $F(s,t)$ (respectivement $G(u,v)$) comme décrit dans la section 2.4.2 du chapitre 2. Vu la propriété d'enveloppe convexe dans la base de Bernstein, il est clair que cette construction convient.

On commence par récupérer les composantes connexes qui forment des boucles dans la topologie générée par l'algorithme 3.1. On continue alors à subdiviser toutes les feuilles de $\mathcal{H}$ de manière à ce que chaque boîte intersectant une des boucles $\mathcal{C}_1, \ldots, \mathcal{C}_k$ contienne au plus un seul segment *i.e.* au plus deux points d'intersections entre le bord de cette boîte et $\mathcal{C}$. Après cette étape, certaines ambiguïtés de la structure de noeuds et d'entrelacs de Γ peuvent tout de même subsister. On peut voir, sur la figure 3.27 deux boîtes englobantes dans $\mathbb{R}^3$ partageant des points intérieurs. En joignant les paires de points sur les bords, la courbe rouge peut passer devant ou derrière l'autre courbe (la verte). Il est donc nécessaire de raffiner encore la discrétisation.

Lemme 3.4 : Si deux segments de courbes γ_1, γ_2 de Γ sont disjoints, alors, après un nombre fini de subdivisions des composantes de $\mathcal{C}$ qui correspondent à γ_1 et γ_2, les boîtes englobantes de γ_1 (construites comme il est décrit précédemment) n'intersectent pas celles de γ_2. $\Diamond$

Démonstration : Par subdivision, on peut construire les boîtes englobantes de manière à ce que les distances entre γ_1 et chacune de ses boîtes

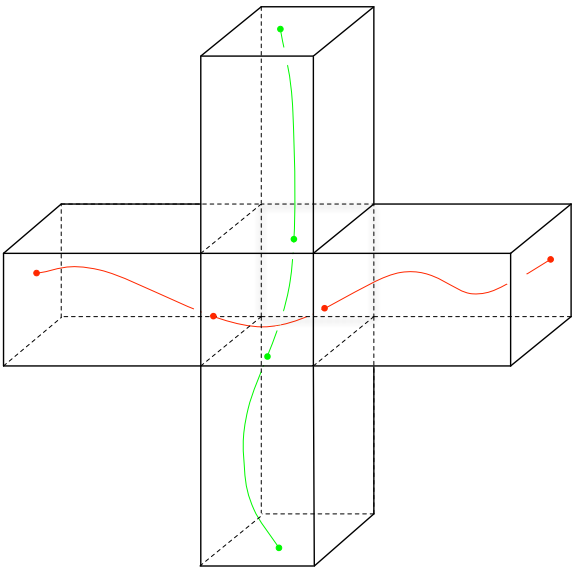


FIG. 3.27: Deux boîtes partageant des points intérieurs.

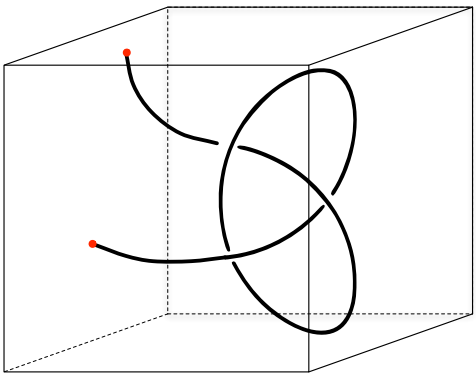


FIG. 3.28: Entrelacement sans boucle.

englobantes soient toutes inférieures à la distance entre γ_1 et γ_2. □

On continue donc la subdivision sur les feuilles de $\mathcal{H}$ jusqu'à ce qu'on se trouve dans le cadre du lemme 3.4. Cela permet alors de lever les ambiguïtés potentielles d'entrelacs entre deux boucles (situation qui correspond au dessin de droite sur la figure 3.25) en évitant la situation de la figure 3.27. Il reste cependant le cas des entrelacs qui ne bouclent pas et qui passent tout de même au travers du critère de subdivision précédent (voir la figure 3.28).

3.7.2 Noeuds et discrétisation

Lemme 3.5 : Soit $\gamma \subset \Gamma$ une portion de courbe contenue dans une boîte englobante issue de la subdivision décrite dans la section 3.7.1. En particulier, le bord de cette boîte englobante ne contient que deux points de γ notés p_1 et p_2. Après un nombre fini de subdivisions dans $\mathbb{R}^4$, on a : $\det(N_F, N_G, \overrightarrow{p_1p_2}) \neq 0$ (dans la boîte correspondante) avec $N_F = \partial_s F \times \partial_t F$ et $N_G = \partial_u G \times \partial_v G$. ◊

Démonstration : La condition $\det(N_F, N_G, \overrightarrow{p_1p_2}) \neq 0$ traduit le fait que le vecteur tangent à γ n'est jamais orthogonal à $\overrightarrow{p_1p_2}$. Vu que γ est lisse par hypothèse, le lemme est une conséquence du théorème de la fonction implicite. □

Si on continue la subdivision sur les feuilles de $\mathcal{H}$ jusqu'à ce qu'on se trouve dans le cadre du lemme 3.5, alors on lèvera les ambiguïtés potentielles d'entrelacs à l'intérieur de chaque boîte englobante générée. Cependant, il reste encore à éviter les entrelacs provenant de deux branches adjacentes.

Proposition 3.8 : Supposons que les conditions des lemmes 3.4 et 3.5 soient vérifiées et que, de plus, $\det(N_F, N_G, \overrightarrow{p_1p_2}) \neq 0$, $\det(N_F, N_G, \overrightarrow{p_2p_3}) \neq 0$ et $\det(N_F, N_G, \overrightarrow{p_1p_3}) \neq 0$ pour des branches adjacentes de la forme $[p_1, p_2]$ et $[p_2, p_3]$. Si l'image (par $F \circ \pi_1$ ou $G \circ \pi_2$) d'une composante connexe de $\mathcal{C}$ qui forme une boucle admet un noeud, alors ce dernier est entièrement caractérisé par sa discrétisation (obtenue par subdivision). En d'autres termes, la liste de segments obtenue par discrétisation décrit un noeud isotope à celui qui est sur Γ.

Démonstration : En fait, on sera dans la situation décrite sur la figure 3.29. Le cas du noeud formé par deux segments adjacents (voir l'illustration sur la figure 3.30) est exclu par la condition de non nullité des trois déterminants de l'énoncé. Il suffit donc de considérer le cas où on a au moins trois segments $[p_1, p_2]$, $[p_2, p_3]$ et $[p_3, p_4]$. Le lemme 3.5 assure que chacun de ces segments de courbe ne s'entrelace pas. Si ces trois segments forment un noeud, alors la boîte englobante contenant $[p_1, p_2]$ intersecte celle qui contient $[p_3, p_4]$ ce qui contredit la propriété du lemme 3.4 vu que $[p_1, p_2]$ et $[p_3, p_4]$ ne sont pas adjacents. □

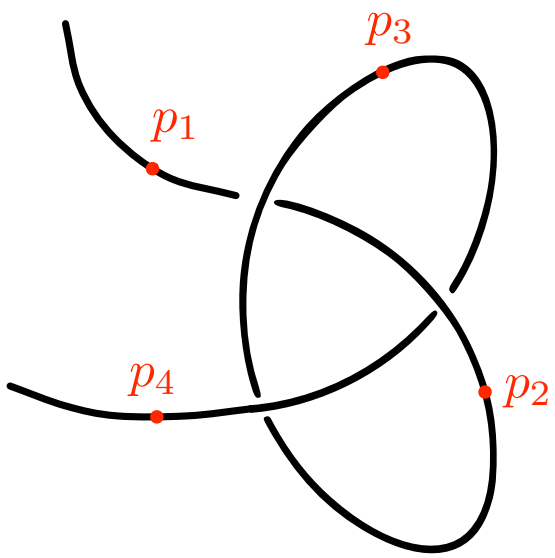


FIG. 3.29: Entrelacement formé par trois segments de courbe.

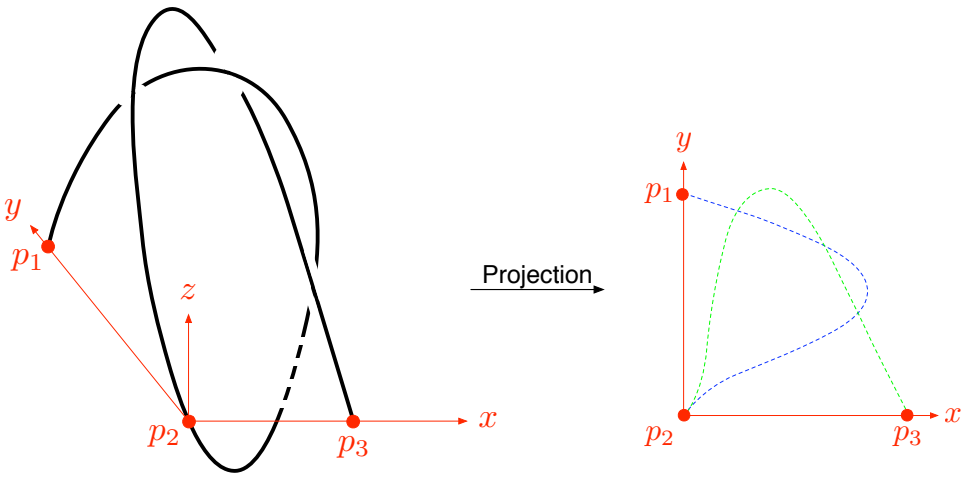


FIG. 3.30: Entrelacement formé par deux segments de courbe adjacents.

3.8 Exemples

On illustre à présent la méthode décrite dans les sections 3.5.4 et 3.6 sur certains exemples. On commence par donner deux cas d'intersection entre deux carreaux de surfaces polynomiales sur les figures 3.31 et 3.32. L'exemple

classique de la théière est également traité. La figure 3.33 montre une approximation de la théière utilisant 32 carreaux de surfaces biquadratiques. Cette approximation possède des lieux d'intersection qui sont illustrés sur la figure 3.34 et le calcul de topologie de ces lieux est représenté sur les figures 3.35 et 3.36.

Ces exemples sont visualisés grâce au logiciel Axel (voir [12] et le chapitre 4) dans lequel la méthode décrite dans les sections 3.5.4 et 3.6 a été implémentée.

FIG. 3.31: Exemple d'intersection de deux carreaux de surfaces polynomiales.

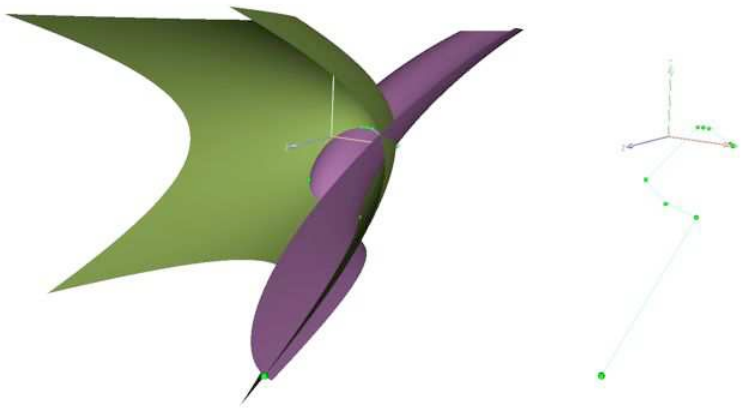


FIG. 3.32: Exemple d'intersection de deux carreaux de surfaces polynomiales.

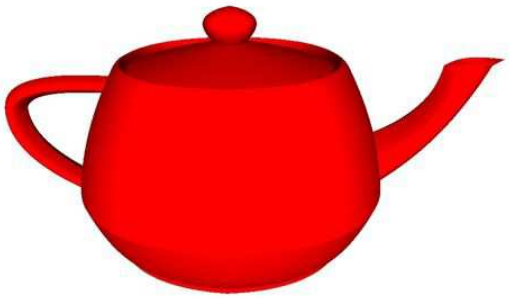


FIG. 3.33: Approximation de la théière par 32 carreaux de surfaces polynomiales.

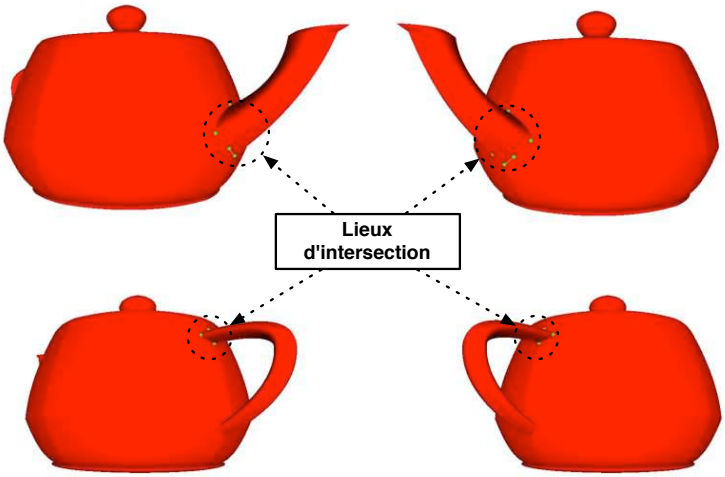


FIG. 3.34: Lieux d'intersection de la théière.

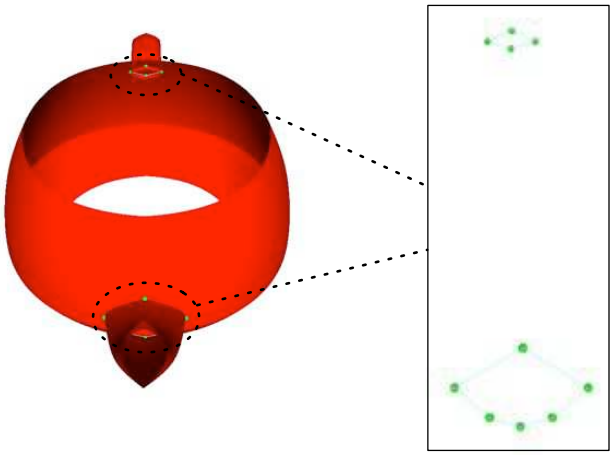


FIG. 3.35: Topologie des lieux d'intersection de la théière.

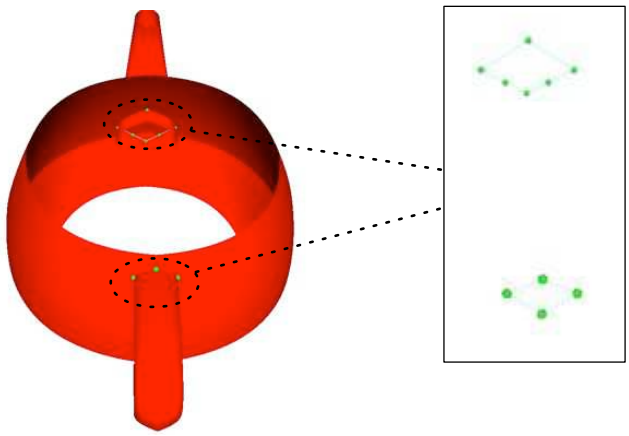


FIG. 3.36: Topologie des lieux d'intersection de la théière.

Chapitre 4

Axel : modeleur algébrique géométrique

Axel [12] est un modeleur algébrique géométrique en ce sens qu'il permet de visualiser et manipuler des objets géométriques ayant des représentations algébriques. Il s'agit essentiellement des courbes et surfaces implicites et paramétrées. Ses principales caractéristiques sont la topologie, l'interpolation, l'approximation, l'intersection, l'auto-intersection et le calcul d'arrangement des courbes et surfaces implicites et paramétrées.

On se propose ici d'exposer les objets et outils qui caractérisent Axel. D'un point de vue utilisateur, on présentera l'interface graphique du logiciel ainsi que les interfaces de fichiers et de script. D'un point de vue développeur, on présentera les hiérarchies internes et les structures de données utilisées dans le but de fournir une implémentation à la fois générique et efficace. Par ailleurs, le système de *plugin* permet d'étendre les fonctionnalités du logiciel ou de *wrapper* une librairie externe pour utiliser l'application comme une interface graphique.

Les modeleurs géométriques courants utilisent essentiellement des représentations discrètes ce qui présente de nombreux avantages : ils sont faciles à appréhender, faciles à manipuler et le rendu est simplifié. Pour toutes ces raisons, le maillage est un modèle de choix pour de nombreux utilisateurs qu'ils soient débutants au confirmés. Bien qu'ils soient adaptés à de nombreuses situa-

tions, ils présentent un point faible : ils ne permettent pas de représenter toutes les surfaces (comme par exemple une sphère) de manière exacte et la quantité de données est étroitement liée à la précision désirée. D'autre part, dans le monde de l'industrie et de la recherche, des modèles géométriques non linéaires sont utilisés pour représenter des formes ou des phénomènes physiques. Par conséquent, ils nécessitent une représentation plus précise et donc des calculs plus complexes mais une quantité de données moindre. Les courbes et surfaces implicites et paramétrées sont des objets géométriques qui répondent à cette exigence et ont une représentation très compacte.

Les systèmes de calcul formel ont évidemment besoin d'un outil de visualisation pour ce type d'objets. Même si certaines solutions existantes permettent d'afficher des objets mathématiques, elles ne proposent pas une réelle interaction. Maple^{TM} et Mathematica^{TM} sont des exemples d'applications intégrant de tels outils. Cependant, ces logiciels donnent des résultats insatisfaisants et ce, principalement à cause de points singuliers ou de courbes singulières. Singular [37] est dédié aux calculs avec des polynômes et effectue des traitements particuliers afin de tenir compte des singularités. Une connexion avec Surf existe et cela permet de fournir des images statiques mais il y a alors un manque d'interaction. Mentionnons également IZIC [30] qui est un des premiers outils indépendants de visualisation des surfaces mathématiques destiné à être connecté à un système de calcul formel externe.

La visualisation d'objets géométriques ayant une représentation mathématique tels que les courbes ou les surfaces est un domaine d'investigation existant depuis plusieurs années et ce, dans de nombreux domaines de recherche tels que l'informatique ou la géométrie algébrique. Les méthodes existantes dépendent en général de la nature des objets. En effet, les modèles paramétrés peuvent être évalués sur des grilles alors que les modèles implicites sont généralement traités de deux manières différentes. La plus répandue est la méthode du lancé de rayons [22, 46] qui est pratique mais dont le principal inconvénient est que les images produites sont statiques *i.e.* qu'elles sont calculées pour un point de vue donné et si on désire changer la position ou l'orientation de l'objet ou de l'utilisateur, alors il faut recalculer une image. L'autre solution consiste à calculer une approximation linéaire par morceaux de l'objet, puis à l'afficher dynamiquement. Les techniques dites de « mar-

ching cube » [48, 56] en font partie mais manquent généralement d'efficacité et peuvent produire des résultats non certifiés. Il existe d'autres techniques qui permettent de garantir la topologie des courbes et des surfaces, même dans les cas singuliers. [2, 3, 23, 34].

Axel vise à fournir un cadre unifié à la fois pour la visualisation et la manipulation des objets géométriques ayant une représentation algébrique. En utilisant des données exactes, on peut fournir des résultats certifiés même lorsque le contexte impose une approximation.

L'application est conçue pour être utilisée soit de manière autonome, soit connectée avec des outils externes. Dans ce deuxième cas, elle devient une interface graphique et son architecture modulaire utilisant les *plugins* permet de le faire facilement. De même, il est aisé de *wrapper* des librairies externes.

Les sections suivantes montrent l'interaction permanente entre les entités géométriques et leur représentation algébrique qui est manipulée dynamiquement dans l'environnement de modélisation.

4.1 Point de vue utilisateur

Dans cette section, on donne un aperçu des objets et des outils fournis dans Axel. Puis, on montrera comment l'utilisateur peut interagir avec eux par le biais de l'interface graphique, de l'interface de script ou d'un fichier.

4.1.1 Objets

Outre les entités géométriques élémentaires telles que les points, les plans, ..., Axel vise à fournir une interface pour la visualisation et la manipulation de courbes et de surfaces avec différents types de représentations : implicite, paramétrée ou linéaire par morceaux.

4.1.1.1 Courbes

Les courbes dans le plan ou dans l'espace sont largement utilisées dans les calculs mathématiques et la modélisation géométrique. Soit pour construire

des surfaces, soit pour en modéliser des sections. C'est par ailleurs l'ingrédient principal des logiciels de dessins techniques.

Courbes implicites. Soit $f(x, y)$ un polynôme bivarié à coefficients dans $\mathbb{R}$. Si f est non nul, alors le lieu des zéros réels de f est une variété algébrique de dimension 0 ou 1 (une courbe plane ou des points isolés). Sa visualisation n'est pas aisée en raison de la difficulté à calculer une famille de points (x_i, y_i) telle que $f(x_i, y_i) = 0$. De plus, détecter et traiter correctement les singularités (*i.e.* des points (x_0, y_0) tels que $f(x_0, y_0) = \partial_x f(x_0, y_0) = \partial_y f(x_0, y_0) = 0$) complique la tâche. Ce sont là les raisons pour lesquelles la plupart des logiciels utilisés par les mathématiciens pour la visualisation des courbes implicites planes donnent des résultats insatisfaisants (en particulier dans les cas singuliers). Considérons à présent deux polynômes à trois variables et à coefficients réels $f(x, y, z)$ et $g(x, y, z)$. Si f et g sont premiers entre eux dans $\mathbb{R}[x, y, z]$, alors l'ensemble $\{(x, y, z) \in \mathbb{R}^3 \mid f(x, y, z) = g(x, y, z) = 0\}$ est une courbe gauche (qu'on peut interpréter comme étant la courbe d'intersection de deux surfaces implicites) ou des points isolés.

Les deux sortes de courbes algébriques qui viennent d'être décrites sont appelées des courbes implicites et peuvent être visualisées dynamiquement dans Axel en utilisant des algorithmes de calcul de topologie (voir la section 4.1.2.1).

La figure 4.1 montre la courbe obtenue à partir du discriminant d'un système polynomial bivarié creux et donne un contre-exemple à la conjecture de Kushnirenko [17]. Cette courbe est de degré 47 en x et y, et la taille maximale des ses coefficients est de l'ordre de 300 bits. La visualisation dans Axel prend moins de 10 secondes.

Courbes paramétrées. Axel peut également être utilisé pour représenter et manipuler les courbes paramétrées. Ces dernières peuvent être classifiées en deux familles.
La première est constituée des courbes paramétrées rationnelles. Une telle courbe est l'image d'une application $\phi : t \mapsto \left(\frac{x(t)}{w(t)}, \frac{y(t)}{w(t)}, \frac{z(t)}{w(t)}\right)$ définie sur un intervalle $[a, b] \subset \mathbb{R}$ où x, y, z et $w \in \mathbb{R}[t]$ avec $w(t) \neq 0$ sur $[a, b]$. Et la seconde étant composée des courbes de type B-spline *i.e.* les images

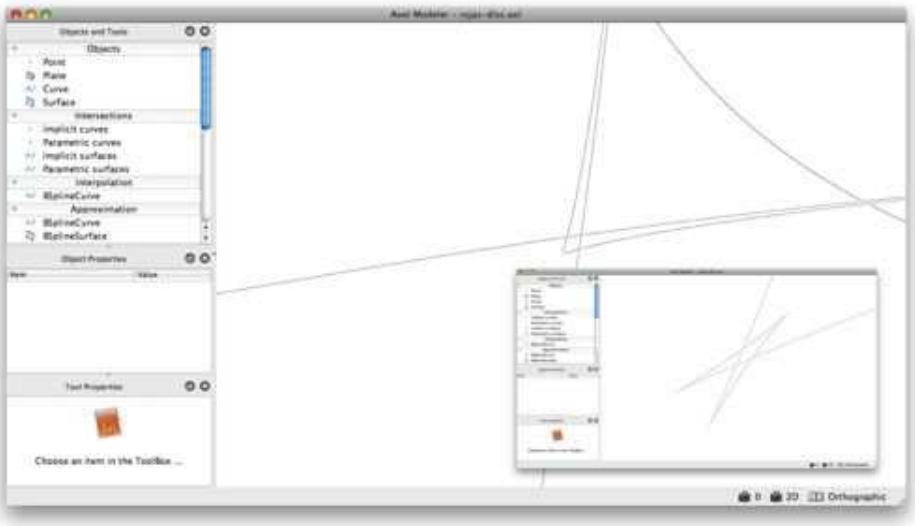


FIG. 4.1: Topologie d'une courbe implicite ayant un cusp caché.

d'applications de la forme $\psi : t \mapsto \sum_{i=1}^{n} p_i B_{i,k,\tau}(t)$ definie sur un intervalle $[a,b] \subset \mathbb{R}$ où : n est le nombre de points de contrôle, $k \in \mathbb{Z}_+^*$ est l'ordre de la B-spline (on suppose que $n \geq k$), pour tout $i \in \{1,\ldots,n\}$, $p_i \in \mathbb{R}^2$ ou $\mathbb{R}^3$ sont les points de contrôle de la B-spline, $\tau = (t_1,\ldots,t_{n+k}) \in [a,b]^{n+k}$ (avec $t_1 = a$ et $t_{n+k} = b$) est le vecteur de noeuds de la B-spline (on suppose que pour tout $i \in \{1,\ldots,n+k-1\}, t_i \leq t_{i+1}$) et que pour tout $i \in \{1,\ldots,n\}, B_{i,1,\tau}(t) = 1$ si $t_i \leq t < t_{i+1}, 0$ sinon, et $B_{i,k,\tau}(t) = \omega_{i,k}(t)B_{i,k-1,\tau}(t)+\omega_{i+1,k}(t)B_{i+1,k-1,\tau}(t)$, où $\omega_{i,k}(t) = \frac{t-t_i}{t_{i+k-1}-t_i}$ si $t_i \neq t_{i+k-1}$, 0 sinon. En supposant de plus que $t_1 = t_2 = \ldots = t_k$ et $t_{n+1} = t_{n+2} = \ldots = t_{n+k}$, p_1 et p_n sont confondus avec les extrémités de la courbe.

La figure 4.2 montre une courbe B-spline dont la définition est donnée par un fichier Axel (qu'on peut voir dans la section 4.1.3.2). On dispose d'un mode d'édition qui permet de modifier dynamiquement le polygone de contrôle de la courbe en déplaçant ses points de contrôle. A chaque courbe paramétrée sélectionnée, un *widget* représentant son espace des paramètres apparaît. Cela permet à l'utilisateur d'interagir avec la courbe en cliquant des paramètres et en visualisant dynamiquement les images correspondantes dans l'espace euclidien.

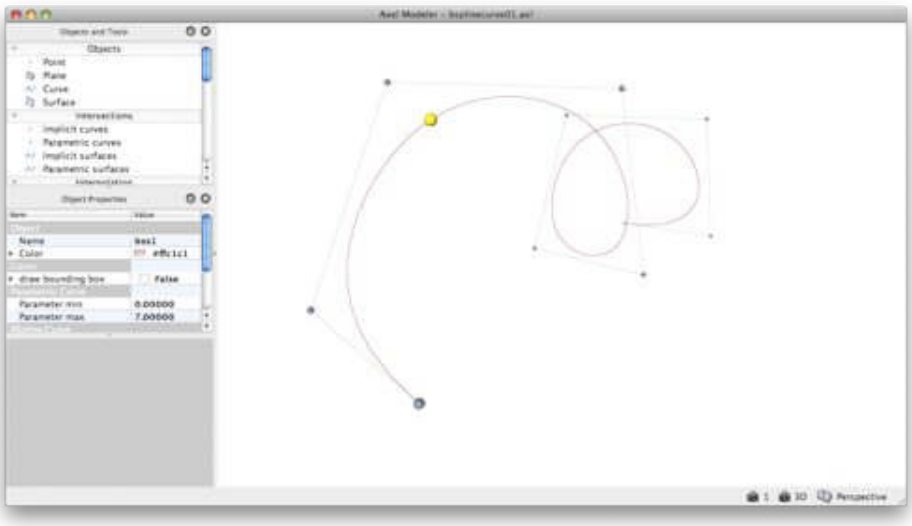


FIG. 4.2: Edition d'une courbe B-spline dans Axel.

4.1.1.2 Surfaces

Les surfaces sont omniprésentes en CAO (Conception Assistée par Ordinateur), FAO (Fabrication Assistée par Ordinateur) et IAO (Ingénierie Assistée par Ordinateur). Elles peuvent également être classées en trois grandes familles : implicites, paramétrées et linéaires par morceaux. Axel est un environnement unifié dont le but essentiel est de rassembler les opérations de base sur les surfaces quelle que soit leur représentation.

Surfaces implicites. Les surfaces implicites (ou surfaces algébriques), sont définies comme étant le lieu des zéros réels d'un polynôme $p(x, y, z)$ à trois variables et à coefficients réels. Signalons que l'on ne fait aucune hypothèse sur le caractère lisse de la surface. Ainsi, même s'il y a des singularités (*i.e.* des points (x, y, z) tels que $p(x, y, z) = \partial_x p(x, y, z) = \partial_y p(x, y, z) = \partial_z p(x, y, z) = 0$), Axel peut, en utilisant des algorithmes de calcul de topologie par subdivision, afficher dynamiquement la surface. Ceci est possible y compris si le lieu singulier n'est pas uniquement composé de points isolés comme le montre l'exemple du parapluie de Whitney sur la figure 4.3.

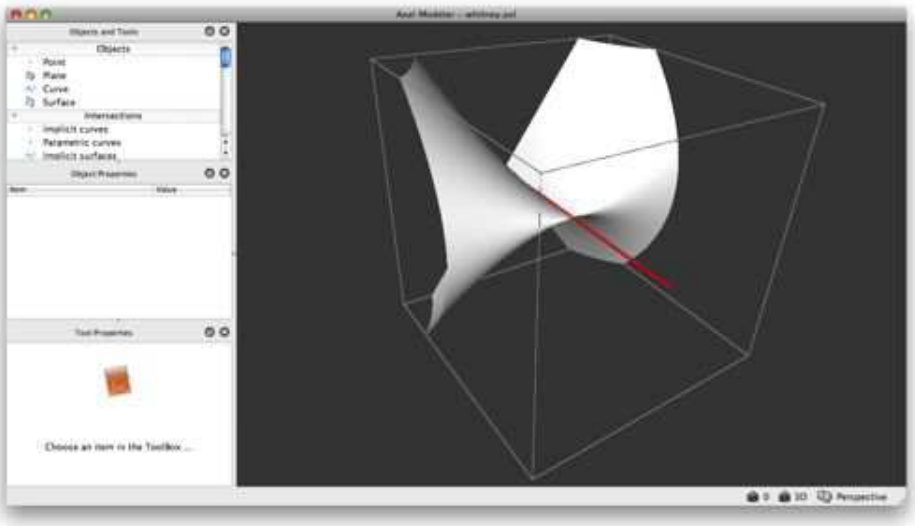


FIG. 4.3: Une surface implicite affichée dynamiquement avec son lieu singulier.

Surfaces paramétrées. Comme dans le cas des courbes, il y a deux sortes de surfaces paramétrées. Les surfaces paramétrées rationnelles qui sont des images d'applications de la forme $\phi : (s,t) \mapsto \left(\frac{x(s,t)}{w(s,t)}, \frac{y(s,t)}{w(s,t)}, \frac{z(s,t)}{w(s,t)}\right)$ définie sur une boîtes $[a,b] \times [c,d] \subset \mathbb{R}^2$ où x, y, z et $w \in \mathbb{R}[s,t]$ avec $w(s,t) \neq 0$ sur $[a,b] \times [c,d]$. Et les surfaces de type B-spline qui sont des images d'applications de la forme $\psi : (s,t) \mapsto \sum_{i=1}^{m} \sum_{j=1}^{n} p_{i,j} B_{i,k,\sigma}(s) B_{j,l,\tau}(t)$ définie sur une boîte $[a,b] \times [c,d] \subset \mathbb{R}^2$. La définition d'une telle surface est similaire au cas des courbes B-splines : m, n, k et $l \in \mathbb{Z}_+^*$ avec $m \geq k$ et $n \geq l$ sont le nombre de points de contrôle et les ordres de la surface dans chaque direction. Pour tout $(i,j) \in \{1,\ldots,m\} \times \{1,\ldots,n\}$, $p_{i,j} \in \mathbb{R}^3$ (ce sont les points de contrôle), $\sigma = (s_1,\ldots,s_{m+k}) \in [a,b]^{m+k}$ et $\tau = (t_1,\ldots,t_{n+l}) \in [c,d]^{n+l}$ sont les vecteurs de noeuds dans chaque direction. Les éléments $B_{i,k,\sigma}(s)$ et $B_{j,l,\tau}(t)$ constituant la base de B-splines sont définis de la même manière que dans le cas des courbes.

4.1.2 Outils

Axel traite, entre autres, les problèmes relatifs aux courbes et aux surfaces suivants : calculs de topologie, d'intersection, d'auto-intersection et d'arrangement. On présente brièvement dans cette section quelques algorithmes qui

sont mis en oeuvre et qui utilisent des résultats de géométrie algébrique.

4.1.2.1 Topologie

Soient $\mathcal{O}_1$ et $\mathcal{O}_2$ deux sous ensembles de $\mathbb{R}^n$ (avec n un entier non nul). On dit que $\mathcal{O}_1$ et $\mathcal{O}_2$ sont isotopes s'il existe une application $F : \mathcal{O}_1 \times [0,1] \longrightarrow \mathbb{R}^n$ telle que F soit continue, $F(\cdot, 0)$ est l'identité sur $\mathcal{O}_1$, $F(\mathcal{O}_1, 1) = \mathcal{O}_2$, et pour tout $t \in [0,1]$, $F(\cdot, t)$ est un homéomorphisme sur son image.

Soit B une boîte de $\mathbb{R}^2$ ou $\mathbb{R}^3$ (*i.e.* $B = [a_1, b_1] \times [a_2, b_2]$ ou $B = [a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$). Alors, la topologie d'une courbe $\mathcal{C}$ dans B est un graphe de points de $\mathcal{C}$ connectés par des segments, qui est isotope à $\mathcal{C} \cap B$. De même, la topologie d'une surface $\mathcal{S}$ dans B est un complexe simplicial, composé de points de $\mathcal{S}$, qui est isotope à $\mathcal{S} \cap B$.

Afin d'obtenir une géométrie correcte des courbes et de surfaces, Axel calcule, dans un premier temps, la topologie en prenant en considération les phénomènes géométriques complexes tels que les singularités ou les auto-intersections. Par la suite, la précision de la visualisation peut être affinée facilement et efficacement.

Les algorithmes de topologie implémentés dans Axel sont décrits en détails dans [2, 3]. Par exemple, en utilisant des techniques de subdivision adaptatives sur les courbes algébriques, Axel est capable de traiter des courbes singulières de degré de plus de 200 avec des tailles de coefficients de plus de 100 bits.

4.1.2.2 Intersection

Considérons deux objets géométriques $\mathcal{O}_1$ et $\mathcal{O}_2$ qui peuvent être de type courbe ou de type surface. Alors, on peut calculer leur lieu d'intersection réel (*i.e.* le lieu commun entre $\mathcal{O}_1$ et $\mathcal{O}_2$ dans $\mathbb{R}^2$ ou $\mathbb{R}^3$) à condition qu'il soit de la dimension espérée *i.e.* un ensemble de points si $\mathcal{O}_1$ et $\mathcal{O}_2$ sont deux courbes ou une courbe et une surface, et une courbe si ce sont deux surfaces.

Plusieurs algorithmes d'intersection sont implémentés dans Axel. Le choix d'une spécialisation ou d'une autre, peut être soit laissé à l'utilisateur soit

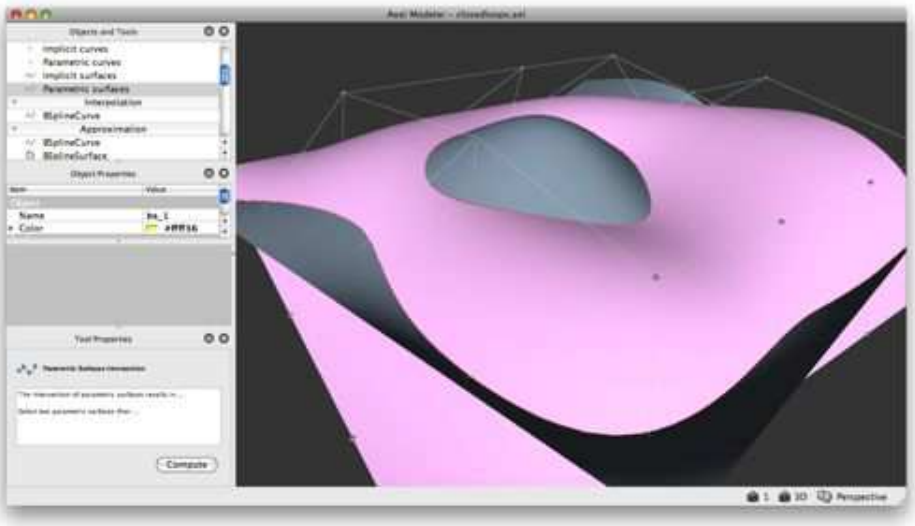


FIG. 4.4: Calcul d'intersection de surfaces paramétrées dans Axel.

effectué de manière automatique en considérant une hiérarchie de classes d'objets et un mécanisme de sélection (voir la section 4.2.1).

Dans le cas des courbes et surfaces implicites, le problème revient soit à résoudre un système polynomial (ce qui peut être fait en utilisant la librairie Synaps [50]), soit à calculer la topologie d'une courbe gauche (voir la section 4.1.1 pour la définition et [47] pour le détail de l'algorithme).

Pour ce qui est des surfaces paramétrées, on dispose encore de plusieurs algorithmes et le choix d'une spécification dépend des informations disponibles pour définir les surfaces. Le cas le plus général considère que les surfaces ne sont données que par évaluations (voir la section 2.1 du chapitre 2 ou [32] pour une description et la figure 4.4 pour une illustration). Un algorithme plus spécifique est mis en oeuvre lorsque les surfaces sont définies par des polynômes (comme les méthodes qui sont décrites dans le chapitre 3, [13] ou [40]).

4.1.2.3 Auto-intersection

Le problème d'auto-intersection est très proche du problème « classique » d'intersection. En fait, il s'agit de déterminer le lieu d'intersection non trivial

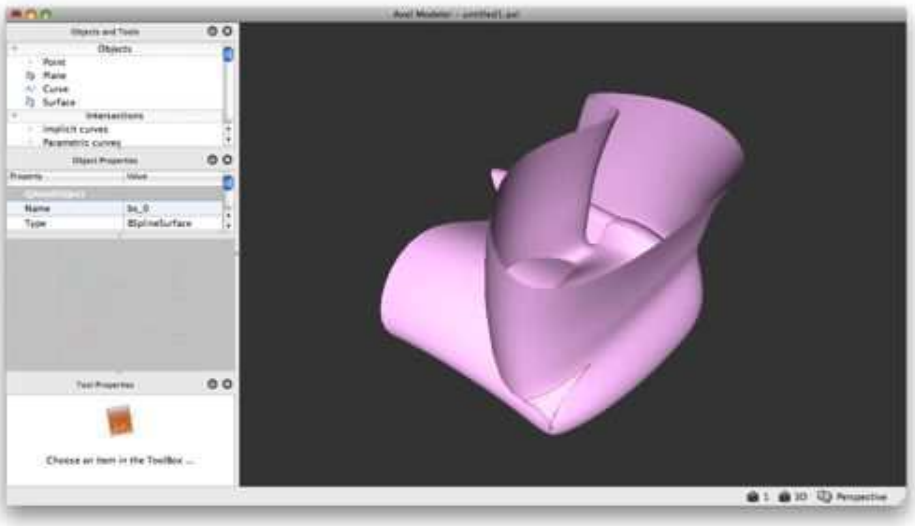


FIG. 4.5: Calcul d'une courbe d'auto-intersection dans Axel.

d'une courbe (ou d'une surface) avec elle-même. Ce lieu est parfois appelé l'ensemble des points multiples.

Axel, traite le calcul des auto-intersections des courbes et des surfaces soit en calculant les lieux singuliers dans le cas des représentations implicites, soit en utilisant des méthodes de segmentation (voir [32]) dans le cas des représentations paramétrées (voir l'illustration 4.5).

4.1.2.4 Arrangement

En modélisation géométrique, la représentation des formes est naturellement fondée sur les représentations semi-algébriques telles que les paramétrisations de type B-spline ou les équations implicites. Un objet géométrique est décrit par l'assemblage de pièces primitives. Lorsque plusieurs objets doivent être manipulés comme, par exemple, dans les opérations (*e.g.* intersection, union, différence entre volumes) en géométrie constructive des solides (*CSG* en anglais pour *Constructive Solid Geometry*), la structure topologique de la forme résultante doit être déterminée. Cela se fait grâce à une description géométrique des différents éléments qui constituent cette structure.

Le calcul d'arrangement permet de résoudre de tels problèmes. La première

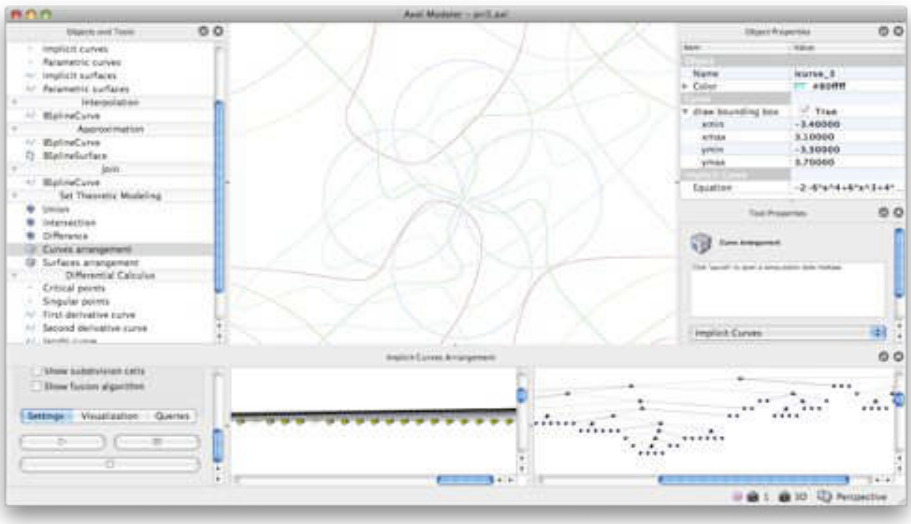


FIG. 4.6: Un calcul d'arrangement d'un ensemble de courbes implicites.

étape est de déterminer la topologie d'un objet géométrique, *i.e.* analyser comment cet objet se décompose dans l'espace ambiant en composantes connexes et comment ces régions, ainsi que leurs limites, sont liées. Ensuite, il faut examiner plusieurs objets, soit de manière incrémentale, soit de manière globale. Le calcul d'arrangement décrit la structure topologique définie par l'union de toutes ces primitives géométriques. Cette décomposition est faite sur les différentes régions (les composantes connexes).

Axel calcule des arrangements pour les courbes implicites, paramétrées et linéaires par morceaux. L'arrangement peut être utilisé soit de manière statique (pour calculer l'arrangement d'un ensemble d'objets) soit de manière dynamique en maintenant la solution lorsque de nouveaux objets sont créés ou lorsque des objets existants sont retirés. Pour une description détaillée de l'algorithme, on renvoie à [66]. La figure 4.6 montre le calcul d'un arrangement d'un ensemble de 15 courbes implicites de degrés supérieurs à 9 dans une configuration dégénérée *i.e.* plusieurs courbes ont un point singulier commun.

4.1.2.5 Calcul différentiel et approximation

D'autres outils sont également fournis. On ne donnera pas une liste exhaustive, toutefois, mentionnons qu'Axel permet de visualiser certains résultats de calcul différentiel, comme le calcul des points critiques ou singuliers. Les calculs d'interpolation et d'approximation d'un ensemble de points donnés sont aussi implémentés. Le premier consistant à produire une courbe ou une surface qui contient tous les points donnés alors que le second donne un objet qui épouse la forme globale du nuage de points. On dispose aussi d'algorithmes pour créer des surfaces cintrées (*lofting surfaces*) à partir de courbes.

4.1.3 Interface

L'interface est le lien entre l'utilisateur et les objets et outils fournis dans Axel. Dans la mesure où le maintient de la représentation algébrique des objets accessibles à l'utilisateur demeure un objectif principal d'Axel, la conception d'une interface intuitive et utile, dans l'esprit des outils de modélisation et des systèmes de calcul formel est un véritable défi. Axel propose trois principaux types d'interactions : son interface graphique, un formalisme de fichier et un environnement de script interactif.

4.1.3.1 Interface graphique

Le modeleur dispose d'un *viewer* principal qui fournit une grande quantité d'interactions avec le clavier et la souris. Il permet une navigation efficace dans un espace à trois dimensions sans encombrer l'écran avec des vues inutiles. Par exemple, il est possible d'aligner la vue avec les axes de coordonnées, de faire un zoom sur une région donnée, d'établir des positions pour la caméra, et d'effectuer des animations.

Dans le *viewer*, chaque objet est associé à une *frame* qui est l'équivalent tri-dimensionnel des *layers* (calques) que l'on rencontre dans les logiciels de manipulation d'images. Un objet peut être manipulé au moyen des translations, rotations et zooms habituels. Cela peut se faire soit relativement par rapport à sa propre *frame* qui définit un système de coordonnées locales, soit

par rapport à la scène qui correspond à un système de coordonnées globales. Ces *frames* peuvent également être organisées au sein d'une hiérarchie, en fournissant les relations parents-enfants.

Le reste de l'interface graphique peut être agencé dynamiquement comme un ensemble de modules qui interagissent et qu'on appelle des *docks*. Par exemple, quand un objet ou un outil est invoqué dans le *dock* principal qui s'appelle *Objects and Tools*, les *docks* correspondants *Object Properties* et *Tool Properties* sont mis à jour dynamiquement pour que l'utilisateur puisse disposer de nouveaux paramètres.

C'est que par le biais du *dock Object Properties* que l'utilisateur peut accéder à la représentation algébrique de l'objet et ainsi le modifier directement dans le sous module correspondant appelé *widget*. Ce *dock* est également muni de nombreux *widgets* intuitifs tels qu'une vue de l'espace des paramètres associé à une courbe ou à une surface paramétrée. On peut alors afficher dynamiquement l'image de l'ensemble des paramètres choisis par l'utilisateur dans le *widget*.

Un autre module intitulé *Scene Inspector Dock* permet de garder une trace des objets présents dans la scène courante et même de les cacher afin de ne se concentrer que sur une partie. Enfin, l'*History Dock* permet de garder la trace de la construction incrémentale d'un modèle géométrique.

4.1.3.2 Interface de fichier

Axel est fourni avec un formalisme de données qui permet à l'utilisateur de spécifier un ensemble d'objets ou de sauvegarder et partager un modèle. Le formalisme est un langage de type XML ce qui permet de rendre simple sa lecture et son écriture. L'exemple suivant illustre comment spécifier la courbe B-spline illustrée sur la figure 4.2.

```
<axl>
   <curve type="bspline" name="bsc1">
      <dimension>3</dimension>
      <number>10</number>
      <order>4</order>
      <knots>0 0 0 0 1 2 3 4 5 6 7 7 7 7</knots>
      <points>
```

```
            0 0 0 1 0 0.5 1 1 1 0 1 1.5 0 0 2
            1 0 2.5 1 1 3 0 1 3.5 0 0 4 1 0 4.5
        </points>
    </curve>
</axl>
```

Le formalisme permet également d'intégrer des scripts qui peuvent servir à être exécutés lorsque le fichier a été analysé par l'application ou à enregistrer des informations liées à l'état du *viewer* comme la position de la caméra, le champ de vision...

4.1.3.3 Interface de script

Axel est muni d'un interpréteur de script. Cela permet d'interagir dynamiquement avec l'application en utilisant son API (*Application Programming Interface*) et donc d'élargir considérablement ses fonctionnalités.

Le langage de script est ECMAScript et ce dernier a été enrichi avec des objets qui permettent une interaction avec l'application principale, le *viewer* ou d'autres modules d'Axel comme l'*Object Manager* ou le *Tool Manager*.

A titre d'exemple, on peut utiliser le script suivant pour produire la même courbe B-spline décrite précédemment mais en modifiant certaines options du *viewer*.

```
var white = new Color(255, 255, 255) ;
Viewer.setBackgroundColor(white) ;
Camera.setPosition(0.0, 1.0, 0.0) ;

var p1  = new Point(0.0 0.0 0.0) ;
var p2  = new Point(1.0 0.0 0.5) ;
var p3  = new Point(1.0 1.0 1.0) ;
var p4  = new Point(0.0 1.0 1.5) ;
var p5  = new Point(0.0 0.0 2.0) ;
var p6  = new Point(1.0 0.0 2.5) ;
var p7  = new Point(1.0 1.0 3.0) ;
var p8  = new Point(0.0 1.0 3.5) ;
var p9  = new Point(0.0 0.0 4.0) ;
var p10 = new Point(1.0 0.0 4.5) ;

var curve = ToolManager.interpolate() ;
```

```
ObjectManager.addObject(curve) ;
```

Cet exemple de script utilise des objets, qui sont construits à l'intérieur du `Viewer`, et aussi la `Camera` pour contrôler le *widget* de visualisation principal. Il crée ensuite un ensemble de points en utilisant le type `Point`, qui n'est rien d'autre qu'un alias sur un constructeur de classe interne à Axel. Ces points sont interpolés comme une courbe B-spline grâce au `ToolManager` et cette courbe est ajoutée au *viewer* en utilisant l'`ObjectManager`. Ce sont là encore des alias sur des classes internes à Axel.

4.2 Point de vue développeur

Cette section donne un aperçu de la philosophie de développement d'Axel, de ses principales entités et de la facilité avec laquelle on peut l'étendre.

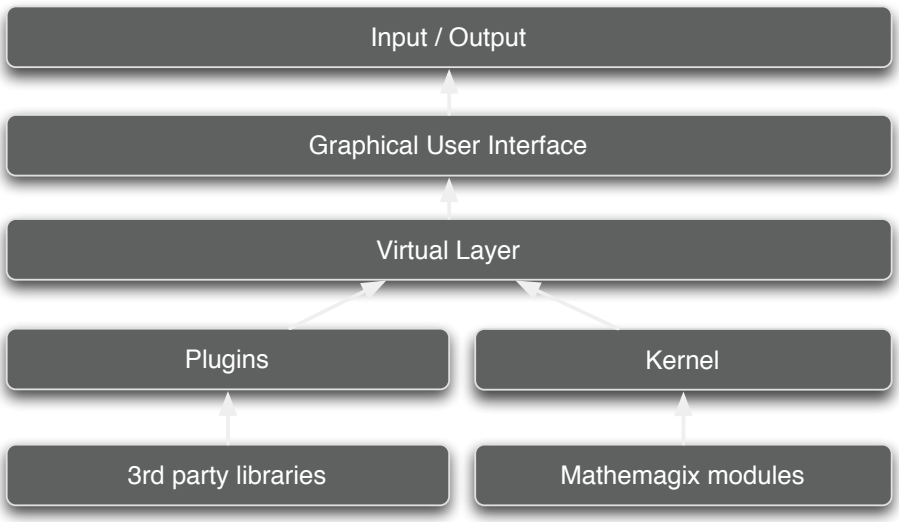


FIG. 4.7: Architecture interne d'Axel.

Comme on peut le voir sur la figure 4.7, Axel est organisée sous forme de hiérarchie. Au niveau le plus haut dans cette hiérarchie, l'utilisateur interagit avec l'application à l'aide de fichiers, de scripts ou de l'interface graphique. Les objets et les outils sont définis au sein d'une couche virtuelle qui est en

fait une hiérarchie virtuelle de classes disposant d'un mécanisme de sélection et qui permet donc d'interfacer des fonctionnalités qui se trouvent dans les couches les plus basses, à savoir les *plugins*. Certains d'entre eux sont primordiaux car ils fournissent les principales fonctionnalités de l'application. Il s'agit du noyau géométrique (*geometric kernel*) et du noyau algébrique (*algebraic kernel*). Le premier fournit une interface entre les objets géométriques manipulés dans Axel et de leur définition dans la librairie correspondante. Le second est utilisé pour apporter un certain nombre d'outils algébriques sur les structures de données utilisées pour la représentation des objets.

4.2.1 *Framework*

Comme on l'a vu dans les sections précédentes, un certains nombre d'objets mathématiques (*e.g.* les polynômes dans la base monomiale ou de Bernstein) sont nécessaires pour représenter les courbes et les surfaces et des algorithmes spécifiques sont associés à ces différentes représentations.

Une attention particulière est portée à la généricité dans la conception des structures pour lesquelles on veut garder une certaine efficacité. Grâce à la paramétrisation du code en utilisant des *templates* et au contrôle de leur instanciation en utilisant des *traits* et des *template expressions* [33], on peut faire de la programmation générique sans pour autant perdre en efficacité. Les implémentations génériques, qui permettent la réutilisation du code sur les différents types de représentation de données, sont combinées avec des implémentations spécialisées qui sont adaptées pour des opérations critiques sur certaines de ces structures.

Le fait qu'une structure mathématique représente un objet géométrique particulier est une information importante pour programmer de manière générique (cela permet de spécifier les algorithmes adéquats). En considérant une hiérarchie de classes et en implémentant des spécialisations à chacun de ses niveaux, l'implémentation la plus spécifique d'un algorithme est toujours choisie à l'exécution ce qui permet d'être efficace et de simplifier le développement.

Dans cette section, on décrit la conception de la couche interne d'Axel permettant une telle combinaison.

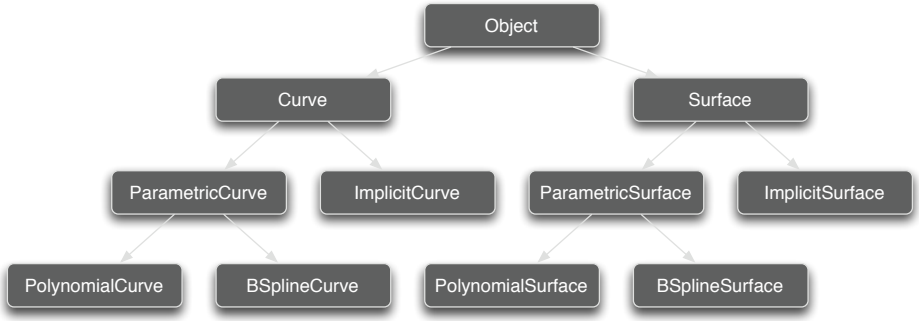


FIG. 4.8: Hiérarchie virtuelle d'objets.

4.2.1.1 Hiérarchie virtuelle d'objets

Les objets géométriques et algébriques peuvent être organisés au sein de différentes catégories. De manière plus pragmatique, cette classification d'objets se retrouve dans une structure d'arbre dans laquel les noeuds correspondent aux classes abstraites. Une relation parent enfant dans cet arbre correspond à une relation d'héritage. La structure dans son ensemble est appelé une hiérarchie virtuelle.

Une feuille correspond à la manière la plus spécifique de « voir » un objet et chaque chemin qui part de la racine pour arriver à cette feuille dans cette hiérarchie virtuelle est une manière de « voir » cet objet de la plus générique à la plus spécifique. Dans un tel parcours, à chaque niveau, l'objet acquiert de nouveaux attributs qui peuvent donner lieu à de nouvelles fonctions ou des spécialisations (surcharge en la programmation objet) de fonctions existantes.

Outre le fait que la classification est une notion naturelle qui correspond à la réalité, il y a deux avantages principaux à une telle organisation. D'une part, cette hiérarchie permet de lancer les algorithmes adéquats en choisissant l'implémentation la plus adaptée au problème. D'autre part, elle donne la possibilité au développeur d'intégrer ses propres types de données à n'importe quel niveau de la hiérarchie et donc, par héritage, de bénéficier de tous les algorithmes et fonctionnalités définis à un niveau plus haut.

Le mécanisme de sélection peut être fait de plusieurs façons. En effet, de nombreux langages tels que le C++ ont leur propre mécanisme de sélection

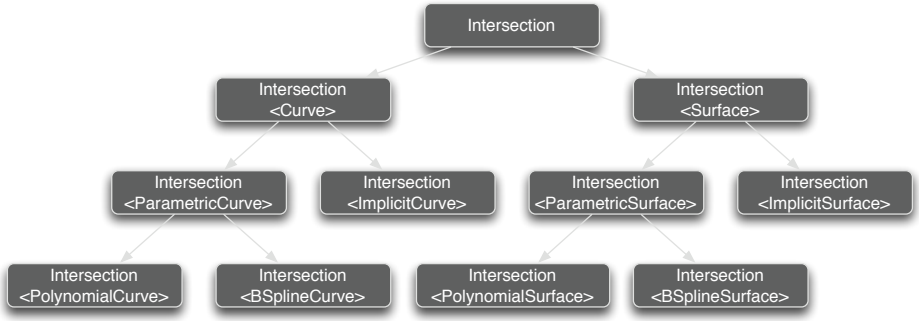


FIG. 4.9: Hiérarchie virtuelle d'outils.

(en utilisant le mot clé `virtual` dans ce cas). Sinon, il est possible d'utiliser des *design patterns* dédiés tels que les *Concept-Vue-Interface pattern*. Dans ce cas, la hiérarchie virtuelle est construite en utilisant un *wrapper* « templaté » pour lier une interface à son implémentation. Le mécanisme de sélection est réalisé en combinant la surcharge avec l'utilisation des espaces de nommage dans lesquels les fonctions globales sont définies de manière à pouvoir utiliser la représentation de l'objet. Par exemple, la figure 4.8 montre une hiérarchie simple d'objets géométriques qui est assez proche de celle que l'on trouve dans Axel.

4.2.1.2 Hiérarchie virtuelle d'outils

Les outils géométriques et algébriques (ou les algorithmes) peuvent également être classifiés. Il s'agit, là aussi, d'une hiérarchie virtuelle. Cependant, étant donné que les algorithmes dépendent des types de leurs arguments, cette dernière est étroitement liée à la hiérarchie virtuelle d'objets.

Le mécanisme de sélection, généralement utilisé avec les fonctionnalités d'un objet (une méthode d'une classe en programmation objet), fonctionne de la même manière sur la hiérarchie d'outils ou d'algorithmes. Ces derniers peuvent éventuellement être définis en utilisant le *pattern template method*.

Si on considère par exemple un algorithme d'intersection, il est possible de définir un schéma générique de subdivision qui subdivise un domaine donné et qui se base sur un prédicat pour s'arrêter ou continuer. Un autre

prédicat est alors utilisé pour traiter une cellule qui ne sera plus divisée. On peut utiliser ce schéma pour définir une intersection de surfaces comme le montre la figure 4.9. A ce niveau, aucun prédicat ne peut être spécialisé. Cependant en considérant que les surfaces à intersecter sont paramétrées, cela permet de disposer de fonctions d'évaluation. Ainsi, une approximation du lieu d'intersection peut-être implémentée en échantillonnant des points sur les grilles de chacune des surfaces par exemple. Maintenant, si on sait de plus que les surfaces sont polynomiales, alors on peut accéder à leurs représentations internes (en l'occurrence aux équations) et ainsi les utiliser pour mettre en place des critères d'injectivité (comme ceux décris dans les sections 3.5.4 et 3.6 du chapitre 3) permettant de rendre les calculs plus efficaces et de certifier le résultat.

Le choix d'un algorithme peut être réalisé au moment de l'exécution de différentes manières. Axel estime que l'implémentation la plus spécifique est également la plus efficace. Mais l'utilisateur peut choisir n'importe quel algorithme dans la hiérarchie virtuelle d'outils.

Dans la mesure où ces hiérarchies peuvent être utilisées pour intégrer des *plugins* dédiés, on doit également pouvoir lier ces hiérarchies avec des fonctionnalités d'une application externe. Dans la section qui suit, on décrit la méthode adoptée pour construire des modules dynamiques en combinant du code générique et spécifique à travers un mécanisme de *plugin* transparent. La connexion avec la librairie de calcul symbolique Mathemagix illustre bien cette approche. Elle sert en fait de base aux calculs avec des polynômes représentant les courbes et surfaces algébriques.

4.2.2 Système de *plugin*

Axel peut être étendu grâce à des *plugins i.e.* grâce à des extensions qui interagissent avec le logiciel et qui fournissent leurs propres fonctionnalités. Plusieurs raisons justifient l'utilisation des *plugins*. Cela permet, d'une part, à des développeurs tiers d'enrichir l'application, et d'autre part de séparer des lignes de code source qui auraient des dépendances ou des licences incompatibles avec celles d'Axel. Rendre l'application extensible grâce à des *plugins* impose de fournir une interface abstraite dont le *plugin* doit hériter. Cela se résume à un ensemble de méthodes qui doivent être surchargées. Le

code ci-dessous est un exemple très simple d'interface de *plugin*.

```
class PluginInterface
{
public :
    virtual ~PluginInterface() {}

    virtual void        init(CoreInterface *) const = 0 ;
    virtual String      name(void)            const = 0 ;
    virtual StringList  features(void)        const = 0 ;
} ;
```

Etant donné que le C++ ne fournit pas d'interfaces comme le font Java ou Objective-C avec *interfaces* ou *protocols*, une interface de *plugin* est une classe abstraite qui ne peu pas être instanciée. Cette interface est le lien entre l'application et le *plugin*.

Dans le code présenté précédemment, l'interface se compose d'une fonction d'initialisation qui donne au *plugin* un accès à une structure contenant des pointeurs vers les principales entités de l'application. Cela permet, par exemple, de laisser le *plugin* ajouter des entrées dans l'interface de l'application et de leur associer des appels à ses propre fonctions internes. En outre, cette interface simplifiée permet à l'application d'obtenir des informations sur le *plugin*, telles que son nom ou une description de l'ensemble des fonctionnalités qu'il fournit.

Axel offre deux types de connexions différentes : les noyaux et les *plugins*. Dans ce deuxième cas, le *plugin* peut être utilisé comme une *glue* entre le modeleur et n'importe quelle librairie de calcul. Cela permet d'utiliser Axel comme une interface graphique de la librairie *gluée*. Ces *plugins* peuvent être développés par n'importe qui du moment qu'il dispose de la documentation adéquate. A contrario, les noyaux sont fournis avec Axel et, même s'ils ne sont pas obligatoires pour utiliser le logiciel, ils fournissent les fonctionnalités les plus intéressantes. Le noyau algébrique est le niveau d'abstraction le plus bas pour gérer la représentation des objets géométriques alors que le noyau géométrique rassemble des algorithmes de plus haut niveau qui maintiennent les structures de données associées aux représentations lors de leurs exécutions. Ces modules ont été construits principalement sur la librairie Synaps qui aujourd'hui a fusionné avec le projet Mathemagix et la

librairie Sisl [18] dédiée aux représentations de types B-spline et NURBS.

4.2.3 Système de noyaux

4.2.3.1 Noyau géométrique

Le noyau géométrique contient toutes les fonctionnalités avancées liées aux opérations géométriques. A chaque fois qu'un objet est créé dans Axel, ce noyau maintient un dictionnaire entre son abstraction géométrique dans l'application et une interface virtuelle de cet objet dans le noyau. Pour illustrer cela, considérons le contexte suivant : on désire afficher une courbe implicite avec de « gros » coefficients et de degré « très élevé ». Au moment où la courbe est instanciée dans Axel, une interface avec un équivalent virtuel est créé dans le noyau géométrique et est associé à l'objet initial. Pour tracer la courbe, le noyau fournit sa structure topologique. Ce calcul est effectué par le module *shape* de la librairie Mathemagix qui est *glué* avec le noyau géométrique. Afin de faire des économies en temps et en espace, la structure produite par ce calcul est entièrement manipulée par le noyau. Par la suite, toute manipulation de l'objet se terminera par des appels de fonctions du noyau et la récupération de la structure de données du noyau correspondra à son abstraction géométrique dans le modeleur. De cette manière, un « gros » polynôme représentant la courbe ne sera instancié qu'une fois. Il sera associé, via la dictionnaire, à l'objet et manipulé à travers des algorithmes du noyau et ce, sans perte d'efficacité.

Le module *shape* de Mathemagix possède son propre mécanisme de sélection ainsi qu'une hiérarchie virtuelle d'objets et d'outils. Cela permet de simplifier la spécialisation d'un algorithme ou l'ajout d'un nouveau type de données.

Pour résumer, le noyau géométrique a trois principaux objectifs. Le premier est de *gluer* les interfaces d'objets géométriques de Mathemagix avec Axel afin de fournir des algorithmes efficaces tout en maintenant des structures algébriques non triviales. Le deuxième est de faire des économies à la fois en temps et en espace en évitant la duplication des calculs et des données. Et enfin, le troisième est de fournir un second niveau de sélection sur les interfaces qui sont associées (grâce au dictionnaire) aux objets virtuels construits dans Axel.

4.2.3.2 Noyau algébrique

Le noyau algébrique contient toutes les fonctionnalités avancées liées aux représentations algébriques d'objets géométriques dans Axel. A chaque fois qu'un objet est créé, ce noyau maintient un dictionnaire entre l'entité géométrique qui est dans Axel et l'entité algébrique correspondante (dans le noyau algébrique). Pour illustrer cela, considérons le contexte suivant : on désire calculer les points singuliers d'une courbe implicite sur un domaine donné. Au moment où la courbe implicite est instanciée dans Axel, le noyau algébrique créé un polynôme bivarié qui représente cette courbe. Pour obtenir les points singuliers (*i.e.* les solutions de $f(x, y) = \partial_x f(x, y) = \partial_y f(x, y) = 0$), le noyau doit faire appel à un solveur polynomial ainsi qu'à des calculs de dérivées partielles. Ces dernières fonctionnalités sont *gluées* à partir des modules *algebrix* et *subdivix* de Mathemagix. Ainsi, le noyau ne sera qu'un *wrapper* sur ces fonctions.

Pour avoir la liste des points singuliers, le noyau récupère le polynôme représentant la courbe en utilisant le dictionnaire, calcule ses dérivées partielles en utilisant ses fonctions internes, et instancie un solveur. Pour finir, les solutions du solveur sont converties en points Axel et sont retournés afin d'être ajoutés à l'*object manager*.

Pour résumer, le noyau algébrique a deux principaux objectifs. Le premier est de *gluer* les calculs algébriques de Mathemagix avec Axel. Le second est de bénéficier des objets algébriques pour représenter les objets géométriques d'Axel et d'effectuer diverses opérations sur leurs représentations.

Conclusion et perspectives

L'outil CAO est aujourd'hui largement employé et ce, dans des domaines très diversifiés. La démocratisation des logiciels existants rend son utilisation de plus en plus accessible. Cependant, la jeunesse de cette discipline suscite des recherches actives dans le milieu scientifique. Notamment en ce qui concerne les méthodes d'approximation des surfaces utilisées ainsi que les procédés d'intersection correspondants qui ne sont pas encore très bien maîtrisés. La demande étant importante, il semble pertinent de proposer de nouvelles alternatives aux approches déjà existantes.

Dans cette thèse, nous avons proposé une méthode d'approximation par des surfaces polynomiales qui, comme on l'a vu, permet d'avoir des représentations relativement compactes et une plus grande flexibilité grâce à leur géométrie. L'intersection des surfaces a été traité à partir de ce type de modèle car il s'agit d'un problème important issu du domaine. Toutefois, si on espère voir un jour des éditeurs de logiciels de CAO adopter un tel type d'approximation, cela suppose indubitablement une élaboration de tout un panel d'outils qui permettent de manipuler ce « nouveau » modèle. Ce travail reste à faire et si, une fois tout cela réalisé, ce point de vue se révèle être un compromis significatif sur le rapport qualité/efficacité, alors il pourra être largement adopté par les industriels sur un court terme et être perçu comme une alternative intéressante à un modèle qui n'a quasiment pas évolué depuis les débuts de la CAO. Bien entendu, les consommateurs de ces logiciels de modélisation seront également amenés à manipuler les outils qui auront été développés. Même si pour ces utilisateurs, ce maniement sera invisible, la qualité sera, quant à elle, largement appréciable et les domaines concernés se-

ront très diversifiés : design automobile, conception de circuits électroniques, industrie du textile et, de manière générale, tous les métiers faisant appel à la CAO.

L'existence de la librairie de calculs numériques et symboliques Synaps ([50]), qui a aujourd'hui fusionnée avec le projet Mathemagix ([63]), permet de fournir une base *open source* solide en ce qui concerne les algorithmes et les structures de données mettant en jeux des polynômes comme par exemple les solveurs polynomiaux. Par ailleurs, cette librairie constitue le noyau algébrique du modeleur géométrique Axel ([12]) qui est également *open source*. Ce dernier contient plus de 300 000 lignes de code C++ utilisant la librairie Qt et a été téléchargé plus de 10 000 fois. Il est constitué d'une architecture modulaire fournissant une interface sur les outils algébriques et les algorithmes géomètriques fondamentaux permettant la conception facilitée de nouvelles fonctionnalités sous la forme de *plugins*. En particulier, l'implémentation des routines d'approximation peut servir de base à l'élaboration de nouveaux *plugins* correspondants aux outils précédemment cités. Cela permettra d'enrichir rapidement le travail qui a été effectué dans cette thèse dans le but de le voir éventuellement adopté par des logiciels commerciaux et largement utilisés dans le monde de l'industrie.

Bibliographie

[1] J. Abdeljaoued, G. M. Diaz-Toca, and L. González-Vega. Minors of bezout matrices, subresultants and the parameterization of the degree of the polynomial greatest common divisor. *Int. J. Comput. Math.*, 81(10) :1223–1238, 2004.

[2] L. Alberti and B. Mourrain. Regularity criteria for the topology of algebraic curves and surfaces. In Ralph Martin, Malcom Sabin, and Joab Winkler, editors, *Mathematics of Surfaces XII LNCS*, volume 4647 of *LNCS*, pages 1–28. Springer, 2007.

[3] L. Alberti and B. Mourrain. Visualisation of implicit algebraic curves. In *Pacific Conference on Computer Graphics and Applications 2007*, volume 15 of *Pacific Graphics*, pages 449–452, USA, 2007. IEEE Computer Society.

[4] C. L. Bajaj, C. M. Hoffman, R. E. Lynch, and J. E. H. Hopcroft. Tracing surface intersections. *Comput. Aided Geom. Des.*, 5(4) :285–307, 1988.

[5] R. E. Barnhill and S. N. Kersey. A marching method for parametric surface/surface intersection. *Comput. Aided Geom. Des.*, 7(1-4) :257–280, 1990.

[6] M. Barton and B. Jüttler. Computing roots of polynomials by quadratic clipping. *Comput. Aided Geom. Des.*, 24(3) :125–141, 2007.

[7] L. Busé. *Étude du résultant sur une variété algébrique*. PhD thesis, Université de Nice Sophia-Antipolis, 12 2001.

[8] L. Busé and C. D'Andrea. Inversion of parameterized hypersurfaces by means of subresultants. In *ISSAC '04 : Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, pages 65–71, New York, NY, USA, 2004. ACM.

[9] L. Busé, M. Elkadi, and A. Galligo. Intersection and self-intersection of surfaces by means of bezoutian matrices. *Comput. Aided Geom. Des.*, 25(2) :53–68, 2008.

[10] L. Busé and A. Galligo. Using semi-implicit representation of algebraic surfaces. In *Proceedings of the Shape Modeling and Applications Conference*, pages 342–345. IEEE Computer Society, 2004.

[11] L. Busé and B. Mourrain. MULTIRES, 2000. A maple package, for the manipulation of multivariate polynomials.

[12] S. Chau, B. Mourrain, and J. Wintz. Axel algebraic geometric modeler. `http ://axel.inria.fr`.

[13] S. Chau, M. Oberneder, A. Galligo, and B. Jüttler. Intersecting biquadratic Bézier surface patches. In Bert Jüttler and Ragni Piene, editors, *Computational Methods for Algebraic Spline Surfaces*, pages 61–79, Oslo, 2007. Springer.

[14] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms : An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer Verlag, New York, 1992.

[15] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer-Verlag, New York, 1997.

[16] G. M. Diaz-Toca and L. González-Vega. Various new expressions for subresultants and their applications. *Appl. Algebra Eng. Commun. Comput.*, 15(3-4) :233–266, 2004.

[17] A. Dickenstein, M.J. Rojas, K. Rusekz, and J. Shih. Extremal real algebraic geometry and a-discriminants. Preprint, 2007.

[18] T. Dokken. The sisl library. `http ://www.sintef.no/sisl`.

[19] T. Dokken. Finding intersections of B-spline represented geometric

entities using recursive subdivsion techniques. *Comput. Aided Geom. Des.*, 2 :189–195, 1985.

[20] T. Dokken. Approximate implicitization. *Mathematical Methods for Curves and Surfaces : Oslo 2000*, pages 81–102, 2001.

[21] T. Dokken. Approximate implicitization for surface intersection and self-intersection. In *ECITTT Euroconference on CAE Integration - tools, Trends and Technologies*, 2001.

[22] T. Dokken and J.S. Seland. Real-time algebraic surface visualization. In *Geometric Modelling, Numerical Simulation, and Optimization*, pages 163–183. Springer Berlin Heidelberg, 2007.

[23] A. Eigenwillig, M. Kerber, and N. Wolpert. Fast and exact geometric analysis of real algebraic plane curves. In *ISSAC '07 : Proceedings of the 2007 international symposium on Symbolic and algebraic computation*, pages 151–158, New York, NY, USA, 2007. ACM.

[24] G. Elber and M.S. Kim. Geometric constraint solver using multivariate rational spline functions. In *Proceedings of the sixth ACM Symposium on Solid Modelling and Applications*, pages 1–10. ACM Press, 2001.

[25] M. Elkadi and B. Mourrain. Some applications of bezoutians in effective algebraic geometry. Rapport de Recherche 3572, INRIA, Sophia Antipolis, 1998.

[26] M. Elkadi and B. Mourrain. *Introduction à la résolution des systèmes polynomiaux*, volume 59 of *Mathématiques et Applications*. Springer, 2007.

[27] R. T. Farouki and T. N. T. Goodman. On the optimal stability of the bernstein basis. *Math. Comput.*, 65(216) :1553–1566, 1996.

[28] M. Fioravanti, L. González-Vega, and I. Necula. Computing the intersection of two ruled surfaces by using a new algebraic approach. *J. Symb. Comput.*, 41(11) :1187–1205, 2006.

[29] M. Fioravanti, L. Gonzalez-Vega, and A. Seidl. Real implicitization of curves and geometric extraneous components. In *SNC '07 : Proceedings of the 2007 international workshop on Symbolic-numeric computation*, pages 87–96, New York, NY, USA, 2007. ACM.

[30] R. Fournier, N. Kajler, and B. Mourrain. Visualization of mathematical surfaces : the izic server approach. *J. of Symbolic Computations*, 19 :159–173, 1994.

[31] J. Hoschek G. Farin and M.S. Kim. *Handbook of Computer Aided Geometric Design*. Elsevier, 2002.

[32] A. Galligo and J-P. Pavone. A sampling algorithm computing self-intersections of parametric surfaces. In Bernard Mourrain, Mohamed Elkadi, and Ragni Piene, editors, *Algebraic Geometry and Geometric Modeling*, pages 185 – 204, Nice, France, 2006. Springer.

[33] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns : elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[34] L. González-Vega and I. Necula. Efficient topology determination of implicitly defined algebraic plane curves. *Comput. Aided Geom. Design*, 19(9) :719–743, 2002.

[35] L. González-Vega and G. Trujillo. Implicitization of parametric curves and surfaces by using symmetric functions. In *ISSAC '95 : Proceedings of the 1995 international symposium on Symbolic and algebraic computation*, pages 180–186, New York, NY, USA, 1995. ACM.

[36] T. A. Grandine and F. W. Klein. A new approach to the surface intersection problem. *Computer Aided Geometric Design*, 14 :111–134, 1997.

[37] G.-M. Greuel and G. Pfister. Singular and applications. Jahresbericht der DMV 108, 2006.

[38] T. Gunji, S. Kim, M. Kojima, A. Takeda, K. Fujisawa, and T. Mizutani. Phom, a polyhedral homotopy continuation method for polynomial systems. *Computing*, 73(1) :57–77, 2004.

[39] I. Hanniel and G. Elber. Subdivision termination criteria in subdivision multivariate solvers. In M.-S Kim and K. Shimada, editors, *Proceedings of Geometric Modeling and Processing 2006 (GMP2006)*, pages 115–128, 2006.

[40] J. Hass, R. T. Farouki, C. Yong Han, X. Song, and T. W. Sederberg.

Guaranteed consistency of surface intersections and trimmed surfaces using a coupled topology resolution and domain decomposition scheme. *Adv. Comput. Math.*, 27(1) :1–26, 2007.

[41] C.M. Hoffman. Implicit curves and surfaces in cagd. Computer Graphics and Applications, 1993.

[42] M. E. Hohmeyer. A surface intersection algorithm based on loop detection. In *ACM Symposium on Solid Modeling Foundations and CAD/-CAM Applications*, pages 197–207, 91.

[43] J. Hoschek and D. Lasser. *Fundamentals of computer aided geometric design.* A. K. Peters, Ltd., Natick, MA, USA, 1993. Translator-Larry L. Schumaker.

[44] B. Jüttler, P. Chalmovianský, M. Shalaby, and E. Wurm. Approximate algebraic methods for curves and surfaces and their applications. In *SCCG '05 : Proceedings of the 21st spring conference on Computer graphics*, pages 13–18, New York, NY, USA, 2005. ACM.

[45] A. Khetan. The resultant of an unmixed bivariate system. *J. Symb. Comput.*, 36(3-4) :425–442, 2003.

[46] A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen. Fast and robust ray tracing of general implicits on the gpu. Technical report, University of Utah Technical, 2007.

[47] C. Liang, B. Mourrain, and J-P. Pavone. Subdivision methods for the topology of 2d and 3d implicit curves. In Bert Jüttler & Ragni Piene, editor, *Computational Methods for Algebraic Spline Surfaces, 2005*, pages 171–186, Oslo, Norway, 2007. Springer.

[48] R. Morris. A new method for drawing algebraic surfaces. In *Proceedings of the 5th IMA Conference on the Mathematics of Surfaces*, pages 31–47, New York, NY, USA, 1994. Clarendon Press.

[49] B. Mourrain and J-P. Pavone. Subdivision methods for solving polynomial equations. Technical Report 5658, INRIA Sophia-Antipolis, 2005.

[50] B. Mourrain, J-P. Pavone, P. Trebuchet, E.P. Tsigaridas, and J. Wintz. SYNAPS : A library for dedicated applications in symbolic numeric

computing. In M.E. Stillman, N. Takayama, and J. Verschelde, editors, *Software for algebraic geometry IMA Volumes in Mathematics and its Applications.*, volume 148, pages 81–110. Springer, 2007.

[51] B. Mourrain and Ph. Trébuchet. Generalized normal forms and polynomial system solving. In M. Krauers, editor, *Proc. Intern. Symp. on Symbolic and Algebraic Computation*. New-York, ACM Press., 2000.

[52] B. Mourrain, M. N. Vrahatis, and J-C. Yakoubsohn. On the complexity of isolating real roots and computing with certainty the topological degree. *J. Complexity*, 18(2) :612–640, 2002.

[53] T. Nishita, T. W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface patches. *SIGGRAPH Comput. Graph.*, 24(4) :337–345, 1990.

[54] N.M. Patrikalakis. Surface-to-surface intersections. *IEEE Comput. Graph. Appl.*, 13(1) :89–95, 1993.

[55] J.P. Pavone. *Auto-intersection de surfaces pamatrées réelles.* PhD thesis, Université de Nice Sophia-Antipolis, 2004.

[56] S. Plantinga and G. Vegter. Isotopic approximation of implicit curves and surfaces. In *SGP '04 : Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 245–254, New York, NY, USA, 2004. ACM.

[57] M. Shalaby, B. Jüttler, and J. Schicho. Approximate implicitization of planar curves by piecewise rational approximation of the distance function. *Appl. Algebra Eng. Commun. Comput.*, 18(1-2) :71–89, 2007.

[58] E.C. Sherbrooke and N.M. Patrikalakis. Computation of the solutions of nonlinear polynomials systems. *Computer Aided Geometric Design*, 10(5) :379–405, October 1993.

[59] Z. Sír, R. Feichtinger, and B. Jüttler. Approximating curves and their offsets using biarcs and pythagorean hodograph quintics. *Computer-Aided Design*, 38(6) :608–618, 2006.

[60] Z. Sír, J. Gravesen, and B. Jüttler. Curves and surfaces represented by polynomial support functions. *Theor. Comput. Sci.*, 392(1-3) :141–157, 2008.

[61] J.P. Técourt. *Sur le calcul effectif de la topologie de courbes et surfaces implicites*. PhD thesis, Université de Nice Sophia-Antipolis, 2005.

[62] J.B. Thomassen. *Self-Intersection Problems and Approximate Implicitization*, pages 155–170. Springer, 2005.

[63] J. van der Hoeven, G. Lecerf, B. Mourrain, and O. Ruatta. Mathemagix computer algebra system. `http ://mathemagix.org`.

[64] J. Verschelde. Algorithm 795 : Phcpack : a general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. Math. Softw.*, 25(2) :251–276, 1999.

[65] J. Wintz. *Algebraic methods for geometric modeling*. PhD thesis, Université de Nice Sophia-Antipolis, 2008.

[66] J. Wintz and B. Mourrain. A subdivision arrangement algorithm for semi-algebraic curves : an overview. In *Pacific Conference on Computer Graphics and Applications 2007*, volume 15 of *Pacific Graphics*, pages 449–452, Maui, USA, 2007. IEEE Computer Society.

[67] E. Wurm and B. Jüttler. Approximate implicitization via curve fitting. In *Symposium on Geometry Processing*, pages 240–247, 2003.

[68] J-C. Yakoubsohn. Finding a cluster of zeros of univariate polynomials. *J. Complexity*, 16(3) :603–638, 2000.

Résumé

Cette thèse porte sur un des problèmes majeurs issus du domaine de la Conception Assistée par Ordinateur (CAO) à savoir celui de l'intersection. On aborde cette problèmatique avec une approche novatrice passant par une nouvelle forme de représentation des surfaces dites « procédurales ». Cette dernière se base sur des approximants plus fins que les triangles habituellement utilisés, il s'agit de carreaux de surfaces paramétrées polynomiales de bas degré. L'approximation ainsi obtenue possède des caractères intéressants en termes de qualité et de représentation. Cependant, la mise en oeuvre d'une telle stratégie nécessite l'élaboration d'outils adaptés. En particulier, pour le problème d'intersection, il faut savoir intersecter efficacement les approximants. Pour cela, une méthode algorithmique permet de se focaliser uniquement sur des configurations d'intersection « pertinentes ». Plusieurs méthodes sur l'intersection des surfaces paramétrées polynomiales sont ensuite exposées de manière effective. Enfin, les aspects d'implémentation sont également abordés à travers l'intégration des algorithmes développés dans un modeleur algébrique géométrique.

Mots clés : CAO, modélisation géométrique, courbes et surfaces, approximation, interpolation, algorithmes et programmation efficaces.

Abstract

The intersection problem is one of the major task in Computer Aided Geometric Design (CAGD). In order to deal with this topic, we choose a model of approximation for the parameterized surfaces that are given by evaluations (the so called procedural surfaces). Many authors use triangular meshes but in this thesis, we use more complex shape primitives. More precisely it is the polynomial parameterized surfaces of small degree. The quality of the resulted approximation is better than the usual mesh method especially for the computed intersection locus. But if we want to use this kind of representation, we have to intersect efficiently such two polynomial parameterized surfaces. So, as a preprocessing, a detection of suitable intersection configurations between patches is given. Then, several intersection methods for the polynomial parameterized surfaces are exposed in details. Finally, the elaborated algorithms are implemented in an algebraic geometric modeleur.

Keywords : CAGD, geometric modeling, curves and surfaces, approximation, interpolation, algorithms and efficient programming.