Algebraic Methods for Geometric Modeling

Julien Wintz

Université de Nice Sophia-Antipolis

École Doctorale STIC

THÈSE

Présentée pour obtenir le titre de :

Docteur en Sciences de l'Université de Nice Sophia-Antipolis

Spécialité : Informatique

par

Julien Wintz

Algebraic Methods for Geometric Modeling

Soutenue publiquement à l'INRIA le 5 Mai 2008
devant le jury composé de :

Président :	André	Galligo	Université de Nice, France
Rapporteurs :	Gershon	Elber	Technion, Israel
	Tor	Dokken	Sintef, Norway
Examinateurs :	Pascal	Schreck	Université Louis Pasteur, France
	Christian	Arber	Missler, France
Directeur :	Bernard	Mourrain	Inria Sophia-Antipolis, France

Algebraic methods for geometric modeling

Julien Wintz

Abstract

The two fields of algebraic geometry and algorithmic geometry, though closely related, are traditionally represented by almost disjoint communities. Both fields deal with curves and surfaces but objects are represented in different ways. While algebraic geometry defines objects by the mean of equations, algorithmic geometry use to work with linear models. The current trend is to apply algorithmic geometry algorithms to non linear models such as those found in algebraic geometry. Such algorithms play an important role in many practical fields such as Computer Aided Geometric Design. Their use raises important questions when it comes to developing software featuring such models. First, the manipulation of their representation implies the use of symbolic numeric computations which still represent one major research interest. Second, their visualization and manipulation is not straightforward because of their abstract nature.

The first part of this thesis covers the use of algebraic methods in geometric modeling, with an emphasis on topology, intersection and self-intersection for arrangement computation of semi-algebraic sets with either implicit or parametric representation. Special care is given to the genericity of the algorithms which can be specified whatever the context, and then specialized to meet specific representation requirements.

The second part of this thesis presents a prototype of an algebraic geometric modeling environment which aim is to provide a generic yet efficient way to model with algebraic geometric objects such as implicit or parametric curves or surfaces, both from a user and developer point of view, by using symbolic numeric computational libraries as a backend for the manipulation of the polynomials defining the geometric objects.

Résumé

Les domaines de géométrie algébrique et de géométrie algorithmique, bien qu'étroitement liés, sont traditionnellement représentés par des communautés de recherche disjointes. Chacune d'entre elles utilisent des courbes et surfaces, mais représentent les objets de différentes manières. Alors que la géométrie algébrique définit les objets par le biais d'équations polynomiales, la géométrie algorithmique a pour habitude de manipuler des modèles linéaires. La tendance actuelle est d'appliquer les algorithmes traditionnels de géométrie algorithmique sur des modèles non linéaires tels que ceux trouvés en géométrie algébrique. De tels algorithmes jouent un rôle important dans de nombreux champs d'application tels que la Conception Assistée par Ordinateur. Leur utilisation soulève d'importantes questions en matière de développement logiciel. Tout d'abord, la manipulation de leur représentation implique l'utilisation de calculs symboliques numériques qui représentent toujours un domaine de recherche majeur. Deuxièmement, leur visualisation et leur manipulation n'est pas évidente, en raison de leur caractère abstrait.

La première partie de cette thèse porte sur l'utilisation de méthodes algébriques en modélisation géométrique, l'accent étant mis sur la topologie, l'intersection et l'auto-intersection dans le cadre du calcul d'arrangement d'ensembles semi-algébriques comme les courbes et surfaces à représentation implicite ou paramétrique. Une attention particulière est portée à la généricité des algorithmes qui peuvent être spécifiés quel que soit le contexte, puis spécialisés pour répondre aux exigences d'une certaine représentation.

La seconde partie de cette thèse présente le prototypage d'un environnement de modélisation géométrique dont le but est de fournir un moyen générique et

efficace pour modéliser des solides à partir d'objets géométriques à représentation algébrique tels que les courbes et surfaces implicites ou paramétriques, à la fois d'un point de vue utilisateur et d'un point de vue de développeur, par l'utilisation de librairies de calcul symbolique numérique pour la manipulation des polynômes définissant les objets géométriques.

Contents

Preface

La thèse se décompose en deux parties intrinsèquement liées. La première, plus théorique, correspond à la proposition d'un algorithme générique de calcul d'arrangement de courbes ou surfaces par subdivision. Cet algorithme de haut niveau mêle des méthodes algébriques telles que des calculs de topologie, d'intersection et auto-intersection, avec des méthodes algorithmiques de segmentation et de parcours de graphes.

La seconde, plus pratique, correspond au prototypage d'un modeleur qui se veut algébrique géométrique, en ce sens qu'il permet la manipulation de la représentation algébrique des objets géométriques telles que les courbes et surfaces implicites ou paramétriques au moyen d'algorithmes à la frontière entre la géométrie algébrique et la géométrique algorithmique tels que ceux évoqués dans la première partie de cette thèse.

Préliminaires algébriques. Les courbes et surfaces en géométrie algébrique sont généralement représentées par morceaux au moyen de diverses équations polynomiales. Les méthodes telles que l'intersection ou l'auto-intersection de ces objets peuvent se réduire à la résolution de systèmes non linéaires d'équations polynomiales exprimées dans la base monomiale ou la base de Bernstein. D'autres opérations géométriques utilisent des techniques de projection pour lesquelles le calcul de résultant est un outil algébrique très pratique. Les techniques de subdivision, peuvent avoir recours aux nombres algébriques pour exprimer les racines d'un polynôme de manière exacte. Le chapitre 1 introduit les notions mathématiques nécessaires à la lecture de ce

document.

Préliminaires géométriques. Traditionnellement, la modélisation géométrique identifie un ensemble de techniques servant à modéliser certaines classes de formes. Elle a d'importantes applications dans plusieurs domaines industriels dont l'automobile, l'aérospatial ou l'architecture.

Les courbes et surfaces en sont les principales entités et sont utilisées pour décrire des formes réalistes. Il est donc nécessaire pour la communauté scientifique de s'intéresser à ce type d'entités géométriques et aux méthodes associées. Les sections 2.1 et 2.2 définissent de telles courbes et surfaces.

De manière combinée, ces courbes et surfaces définissent des solides représentant des objets physiques en s'appuyant principalement sur deux grandes classes de représentation : la réprésentation par les bords et la représentation constructive.

Un choix de représentation est primordial pour un logiciel dans la mesure où les techniques de modélisation sont directement liées à la nature des objets géométriques. Dans la section 2.4, nous donnons un aperçu rapide de certains logiciels proposant des courbes et des surfaces pour la modélisation géométrique.

Préliminaires algorithmiques. La géométrie algorithmique a permis de grandes avancées, principalement sur le traitement des objets discrets. Cependant, les objets géométriques continus présentent de nombreux avantages par rapports aux objets géométriques discrets. D'abord, leur représentation est exacte. Il est par exemple impossible de définir une sphère avec un maillage alors qu'une surface Nurbs le permet. Ensuite, leur représentation est infiniment plus compacte. Par exemple, approcher une sphère de manière précise demande un nombre de sommets et de faces directement proportionnel au niveau de détail requis alors qu'une équation polynomiale décrit le même objet de manière implicite. Les préliminaires algorithmiques de cette thèse commencent par définir une terminologie non ambiguë pour l'expression des problèmes géométriques qui composent ce manuscrit, ainsi que les structures de données utilisées par les algorithmes. Ensuite un état de l'art résume les dernières avancées dans les domaines du calcul de topologie,

d'intersection et d'auto-intersection de courbes et surfaces à représentation implicite ou paramétrique en opposant les techniques à base de balayage et celles à base de subdivision.

Algorithme générique d'arrangement. La première contribution de cette thèse est une méthode générique de calcul d'arrangement par subdivision. Un arrangement d'une collection d'objets est la décomposition de l'espace en sommets, arrêtes, faces *etc*, induites par ces objets.

La méthode proposée est générique à plusieurs titres. D'abord, elle est indépendante de la dimension, quand bien même elle sera spécialisée en dimension 2 pour le calcul d'arrangement de courbes dans le plan et en dimension 3 pour le calcul d'arrangement de surfaces dans l'espace. Les mêmes méthodes ont été appliquées avec succès en dimension 4 dans l'espace des paramètres des deux surfaces paramétriques par exemple. Ensuite, elle est hétérogène. Elle sera spécialisée pour des représentations implicites comme paramétriques ou linéaires par morceaux. Enfin, elle peut être appliquée soit de manière dynamique pour maintenir une structure d'arrangement supportant l'insertion de nouveaux objets ou la suppression d'objets existants, soit de manière statique en considérant une collection fixe d'objets et en calculant le résultat en une seule passe. Cette généricité est rendue possible par la flexibilité de l'approche par subdivision.

Quelle que soit la dimension, le type d'objet ou la méthode de calcul, l'algorithme a un très bon comportement numérique. D'abord les calculs pouvant nécessiter des outils approchés sont effectués une seule fois afin de certifier le résultat et d'assurer la cohérence de la méthode. Ensuite, l'utilisation de structures de segmentation permet d'exhiber des conditions nécessaires à l'intersection d'objets qui permettent de filtrer les appels aux outils de résolution et de limiter leur nombre au strict minimum.

Toute méthode de partitionnement adaptative est guidée par un critère de subdivision, aussi la méthode procède de la façon suivante : tant que l'on ne peut déduire la topologie d'un objet (resp. d'un ensemble d'objets) dans le domaine courant, ce dernier est subdivisé et les cellules de subdivision ainsi obtenues sont testées à leur tour. Les régions définies par la topologie de l'objet (resp. des objets) sont finalement déduites *localement* dans les cellules

qui ont passé le test avec succès puis assemblées de manière ascendante à travers la structure hiérarchique de subdivision pour obtenir l'ensemble des régions définies *globalement* par l'objet (resp. les objets) dans le domaine de départ.

Spécialisation aux courbes. Le chapitre 5 est la spécialisation de l'algorithme générique de calcul d'arrangement par subdivision au cas des courbes dans le plan. Cette spécialisation revient à fournir un test de régularité (utilisé comme critère de subdivision dans l'algorithme générique) pour chaque type de courbe parmi implicite, paramétrique et linéaire par morceaux. Ce test de régularité permet de vérifier que la topologie d'un objet (resp. d'un ensemble d'objets) peut être déterminé à partir d'informations sur le bord de la cellule de subdivision. Le calcul de topologie étant directement lié au test de régularité, ce dernier est également fourni par la spécialisation. La gestion globale étant fournie par l'algorithme générique, ce chapitre ne s'intéresse qu'à traiter le problème localement.

Le calcul d'arrangement de courbes implicites est celui qui pose le plus de problèmes. La section 5.1 distingue le cas de cellules non singulières de celui de cellules singulières. Dans le premier cas les dérivées partielles des polynômes qui représentent la courbe fournissent les informations nécessaires à l'analyse du comportement des segments d'une même courbe dans une cellule. En utilisant des indices pour une direction donnée, un algorithme de connexion de branches permet d'obtenir localement la topologie de la courbe, puis d'en déduire les régions. Dans le second cas, celui d'une auto-intersection transverse ou tangente, ou d'un point isolé, un calcul de degré topologique permet de déduire le nombre de branches émanant du lieu singulier et ainsi de fournir un algorithme trivial de connexion. Dans tous les cas, les points d'intersection de la courbe avec le bord de la cellule, les points critiques, extremums et singuliers sont obtenus au moyen des solveurs par subdivision présentés dans le chapitre 1 en utilisant la représentation de Bernstein des polynômes qui représentent les courbes. En considérant des bornes inférieurs et supérieurs pour les coefficients des polynômes dans la base de Bernstein, nous obtenons une technique d'enveloppement qui permet la gestion optimisée de courbes représentées par des polynômes de haut degré à grand coefficients.

Le cas paramétrique ne pose pas de problème majeur. En effet les points d'intersection de la courbe avec le bord de la cellule de subdivision et les points critiques sont aisément calculés. L'obtention des points singuliers tels que les points de rebroussement est rendue possible par l'utilisation de solveurs non linéaires à condition que la représentation des courbes soit polynomiale. Des méthodes de segmentation permettent également d'approcher les points critiques et singuliers à une précision donnée par un pas d'échantillonnage de l'espace des paramètres. Les algorithmes de connexion sont sensiblement les mêmes que dans le cas implicite, seules les méthodes utilisées pour calculer les entités géométriques intermédiaires varient.

La gestion de courbes linéaires par morceaux (séquence de segments de droite) peut conceptuellement se déduire du cas parametré de par le fait qu'une séquence de segments s'apparente à l'échantillonnage d'une courbe paramétrique. Pour obtenir les points critiques et les points d'auto-intersection, on utilise une structure dite de segmentation monotone qui permet l'isolation efficace des changements de direction des segments de droites.

Spécialisation aux surfaces. Si le cas des courbes commence à être bien documenté pour ce qui est des calculs de topologie et d'intersection, le cas des surfaces pose encore bien des défis et de nombreuses propositions sont régulièrement publiées principalement pour le calcul d'intersection de surfaces paramétriques (polynomiales rationnelles, Bézier, B-spline ou Nurbs). L'intersection de surfaces implicites mène au calcul topologique d'un courbe implicite spatiale par définition même de leur lieu d'intersection.

La topologie d'une surface implicite peut se caractériser localement de la manière suivante. À proximité d'une strate de dimension 2, la topologie de la surface est similaire à celle d'un hyperplan. À proximité d'une strate de dimension 1, la topologie de la surface est similaire à celle d'un cylindre. À proximité d'une strate de dimension 0, la topologie de la surface est similaire à celle d'un cône. L'intersection d'une surface avec les bords de la cellule de subdivision (un cube dans l'espace) peut être de deux natures : soit une courbe implicite quand il s'agit d'une face, soit un point quand il s'agit d'une arrête ou d'un coin. Le lieu singulier d'une telle surface est contenu dans sa silhouette que l'on définit formellement par la variété polaire de la surface. Aussi dans le cas d'une strate de dimension 2, le critère

de régularité opère uniquement sur les faces en utilisant l'outillage que nous fournit la spécialisation aux courbes implicites. Dans le cas d'une strate de dimension 1, le critère de régularité s'enrichit par le test de régularité de la variété polaire de la surface. Ce dernier nécessite le calcul de la topologie d'une courbe implicite spatiale définie comme l'intersection de deux surfaces implicites, fournit en suivant une approche par subdivision exploitant le champ de vecteurs tangents à la courbe. Finalement, nous traitons le cas d'une strate de dimension 0 par le test de régularité de la projection de la variété polaire sur la plan défini par l'unique point singulier de la strate dans la cellule.

Contrairement au cas des courbes, les problèmes d'intersection et d'auto-intersection de surfaces à représentation paramétrées ne sont pas triviaux et mènent naturellement à la manipulation de courbes implicites dans l'espace des paramètres. Aussi, les critères exhibés dans le chapitre 5 sont directement applicables aussi bien pour traiter la topologie singulière d'un objet que pour traiter les conflits qui peuvent survenir entre deux objets. La subdivision étant conduite dans l'espace euclidien, il est cependant nécessaire de garder un certain contrôle sur le résultat de ces tests dans l'espace des paramètres. En vérifiant que les images des espaces respectifs ne s'intersectent pas dans l'espace euclidien, on garantit la nature injective de l'approche.

Applications. Le chapitre 7 introduit deux applications immédiates de l'algorithme générique d'arrangement. La première illustre son utilisation statique dans un contexte de CAO où des surfaces rognées sont définies soit au moyen d'une procédure d'intersection soit par nature dans le cas d'une surface qui s'auto-intersecte comme c'est souvent le cas des surfaces construites par extrusion. L'algorithme d'arrangement, appliqué dans l'espace des paramètres de la surface permet l'identification des régions à supprimer.

La seconde application proposée constitue une avancée majeure en géométrie algorithmique, elle permet le calcul du diagramme de Voronoi d'un ensemble de courbes définies de manière paramétrique par des polynômes (courbes appelées rationnelles). Cette application utilise de manière dynamique le calcul localisé de régions dans l'espace des paramètres formé par les courbes en conflit. Une succession d'opérations booléennes sur les régions obtenues

permet de définir incrémentalement le diagramme de Voronoi avec une complexité optimale en termes de calculs d'intersection.

Un modeleur algébrique géométrique. La seconde contribution de cette thèse est un prototype de modeleur algébrique géométrique nommé Axel. Le logiciel tient une place importante dans chacune des communautés scientifiques de géométrie algébrique et géométrie algorithmique. Alors que le premier produit des systèmes de calcul algébrique (CAS) et des librairies de calcul symbolique et numérique comme Maple, Magma, Singular, Mathemagix, Synaps *etc*, le second fournit des librairies algorithmiques telles que Cgal. D'autres solutions, plus dédiées à la CAO sont aussi disponibles comme l'environnement Irit ou les librairies Sisl et GoTools.

D'un point de vue utilisateur, Axel permet la visualisation et la manipulation interactive des objets géométriques tels que les courbes et surfaces paramétriques, implicites ou discrètes. La visualisation dynamique d'objets géométriques à représentation implicite est en soit une avancée majeure rendue possible par des calculs topologiques dans un domaine où les solutions proposées jusqu'à lors fournissent un résultat statique au moyen de techniques basées sur le lancer de rayon. La manipulation est non seulement rendue possible par un système avancé de calques tri-dimensionnels et par les points de contrôle de certains modèles paramétrés, mais aussi par le biais d'algorithmes qui manipulent directement la représentation algébrique des objets géométriques.

D'un point de vue développeur, Axel est une application qui fournit une interface virtuelle sur les objets élémentaires en modélisation géométrique dans une architecture modulaire basée sur un système de noyaux et de plugins. Cette architecture a pour vocation de simplifier la connexion avec d'autres outils et leur interopérabilité.

Le chapitre 8 présente le logiciel à la fois d'un point de vue utilisateur et d'un point de vue développeur. Les objets et les outils sont d'abord présentés en illustrant les calculs de topologie, d'intersection, d'auto-intersection et d'arrangement, ainsi que les diverses interfaces de flux de fichier, de script et l'interface graphique qui fournit des moyens interactifs et intuitifs d'interagir avec la représentation algébrique des objets géométriques. Ensuite,

l'architecture interne du logiciel est passée en revue, en particulier avec ses hiérarchies de classes virtuelles pour les objets et les outils qui permettent un double niveau de sélection et des facilités de codage. Finalement, les concepts de noyau et de plugins sont brièvement introduits.

Annexes. Le premier annexe correspond au manuel de référence du format de fichier défini par Axel, basé sur XML. Il rappelle les concepts de base du métalangage de description puis définit la grammaire du langage et illustre chaque spécification d'objet par des exemples.

Le second annexe illustre le mécanisme de noyaux et ses capacités de connexion avec pour exemple la librairie Sisl. Il illustre le mécanisme de bijection entre les objets virtuels d'Axel et les instances des objets fournies par des librairies connectées au noyau géométrique et son utilisation dans, entre autres, des calculs d'offset.

Le troisième annexe illustre le mécanisme de plugins et ses capacités de connexion avec pour exemple le système Irit. Il illustre en particulier comment définir un nouveau type objet à l'intérieur du plugin et comment le raccrocher à la hiérarchie virtuelle d'objets d'Axel. Dans cet exemple, le plugin modifie l'interface graphique de l'application en ajoutant des entrées de menu pour invoquer ses parseurs. Les objets chargés sont ensuite graphiquement rendus par Irit dans Axel à travers le système de plugins.

Le dernier annexe rend compte du portage des modules internes d'Axel à un environnement de réalité virtuelle. Ce travail est basé sur l'enrichissement par l'équipe d'ingénieurs de recherche de l'Inria Sophia Antipolis du moteur 3D Ogre pour la gestion des concepts et du matériel utilisés par un tel environnement.

Introduction

Domain. Solid modeling has emerged as a central area of research in such diverse applications as CAD (Computer-Aided Design) and CAM (Computer-Aided Manufacturing) in automobile, aeronautic, architecture or movie industries.

To specify elaborated shapes, solid modeling mainly has recourse to two families of representations. The first one is a constructive representation called *CSG* (Constructive Solid Geometry) which consists in assembling elements of simpler geometry such as cubes or spheres by the mean of boolean operations like union, intersection or difference. With this approach, a solid is represented by a tree which leaves are primitive solids and internal nodes are either rigid motions (translation, rotation, scaling) or boolean operations.

The second one, called *B-Rep* (Boundary Representation), describes objects by their boundaries in terms of n-dimensional entities such as vertices (0-dimensional entity), edges (1-dimensional entity), faces (2-dimensional entity), volumes (3-dimensional entity) and so on. The topological model is then a structure gathering these n-dimensional entities together with incidency and adjacency relationships.

CSG and B-Rep representations have inherent strength and weaknesses. CSG models are intuitive and offer an easy workflow for design. B-Rep models are more flexible for many operations. As a consequence, there is a strong tendency to combine these two representations to benefit from both their advantages.

As a field, solid modeling spans several disciplines from computer science to mathematics. It is therefore a broad subject that benefits a diversity of viewpoints. In particular it finds its main entities in geometric modeling with curves and surfaces which have brought powerful design possibilities *e.g.* with freeform surface modeling.

Curves and surfaces with algebraic representation feature many advantages which have made of them the representation of choice in CAD. First they provide better accuracy by their exact nature. Second, they yield compact models. Such representations include implicit and parametric ones.

Problem. To use such curves and surfaces in solid modeling it is necessary to be able to perform boolean operations on them and to describe the resulting shapes by their boundary. Arrangements are high-level algorithms that allow to solve such problems.

Arrangements of geometric objects have been intensively studied in combinatorial and computational geometry for several decades. Given a collection of objects their *arrangement* is the decomposition of the space into regions induced by these objects. They allow to perform any boolean operation on the input objects and represent regions by a set of vertices, edges, faces and so on. The problem is then to be able to compute an arrangement of implicit or parametric curves or surfaces.

Besides the fact that implicit or parametric curves or surfaces have numerous advantages, their representation is however difficult to manipulate. Algebraic methods therefore play then an important role in geometric modeling and so, in solid modeling as well.

Computing an arrangement of objects with such representations implies one to carry out topology computation, intersection computation and possibly, self-intersection computation.

All these operations may necessitate algebraic computations such as polynomial system solving, resultant computation or algebraic numbers manipulation, which enlarges even more the scope of methods needed by an arrangement computation.

Approach. We propose a generic approach to the arrangement computation using a subdivision scheme.

The method is generic in several respects. First, it does not dependent on the dimension, even though it will be specialized in dimension 2 for the computation of an arrangement of curves in the plane and in dimension 3 for the computation of an arrangement of surfaces in the space.

Secondly, it is heterogeneous. It can be specialized for either implicit, parametric or piecewise-linear representations.

Finally, it can be applied either dynamically in order to maintain an arrangement structure while new objects are inserted or existing objects are removed, or statically considering a collection of objects and computing the result in a single pass. This genericity is made possible by the flexibility of the subdivision scheme.

Any method of adaptive partitioning is guided by a subdivision criterion. The method proceeds the following way: while we can not infer the topology of an object (or a set of objects) in the current subdivision cell, the latter is subdivided and resulting cells are tested in turn. This test, called *regularity criterion* has a local nature and is provided by a specialization of the generic algorithm.

Regions are then locally defined by the topology of the object (or objects) in so called regular cells. Their definition is directly related to the regularity test and therefore also provided in a specialization of the algorithm.

The generic part then merges all these *local* regions across the hierarchical subdivision structure to obtain the set of regions *globally* defined by the object (or the set of objects) in the input domain.

The use of a subdivision scheme, as opposed to a sweep scheme, allows to avoid costly projections at critical points as well as subsequent numerical errors, by enclosing these critical parts in a region in which the configuration can be deduced from information on its boundary.

Whatever the dimension, the type of the objects or the computation method, the algorithm has a very good numerical behavior. First, some computations that may require approximate tools are performed only once to ensure the consistency of the method. Second, the use of segmentation structures brings

necessary conditions for the conflict of regions that allow to filter the use of algebraic solvers and reduce the algorithm complexity.

Validation. We specialize the generic algorithm to the case of curves and surfaces. These specializations first consist in providing a regularity test that will serve as a subdivision criterion for the generic algorithm for each type of object among implicit, parametric and piecewise linear. It assures that objects in a cell of subdivision are in a configuration that allows to deduce regions from information on the boundary of the cell. The determination of regions from the topology of objects within the cell is directly linked to the regularity test, it is therefore also provided by the specialization together with the regularity test. The overall management being provided by the generic algorithm, a specialization only deals with the problem locally.

For the case of curves, we will distinguish two major families of configurations: non-singular configurations and singular configurations. In the first case, algebraic tools permit us to ask for a certain level of monotony of the objects within the cell of subdivision so that we are able to provide an algorithm connecting intersection points of the curves with the border of the cell to obtain branches locally isotopic to the input objects. In the second case, using degree theory to compute the topological degree of a unique singular point within the cell, we want to ensure a star shaped configuration by counting the number of branches stemming out from a singular point. The corresponding connection algorithm is then trivial.

If the case of curves begins to be well documented as far as topology and intersection computation are concerned, surfaces still pose many challenges and many proposals are regularly published mainly for the computation of the intersection of parametric surfaces (rational polynomial, Bézier, B-spline or Nurbs). The intersection of implicit surfaces leads to a topology computation of an implicit space curve by definition of the intersection locus.

The topology of a surface can be locally characterized as follows. Nearby a 2-dimensional stratum, the topology of a surface is similar to the one of a hyperplane. Nearby a 1-dimensional stratum, the topology of a surface is similar to the one of a cylinder. Nearby a stratum of dimension 0, the topology of a surface is similar to the one of a cone. The singular locus of

such a surface is contained in its silhouette that is formally defined by the polar variety of the surface. The general idea to deal with implicit surfaces is to compose regularity criteria using the planar ones on the facets of the subdivision cell together with the analysis of the polar variety in the case of 1-dimensional strata or of its projection in the case of 0-dimensional strata. Since parametric surfaces naturally lead to the definition of implicit curves in some parameter space we will see how the local treatment of a cell of subdivision in an arrangement of parametric surfaces can benefit from the local treatment of implicit curves.

We validate this study into an algebraic geometric modeling environment called Axel. It allows the visualization and manipulation of geometric objects with algebraic representation such as implicit or parametric curves or surfaces.

Its main features are topology, interpolation, approximation, intersection, self-intersection and arrangement computation of implicit and parametric curves and surfaces.

The arrangement implementation follows its design. Using the template method design pattern together with virtual methods it is generic and runs inside the modeler as a daemon which looks for insertions or deletions of objects to maintain the arrangement structure.

Outline. The first three chapters set the framework with an emphasis on objects, algorithms and software. These preliminaries aim at setting the terminology used throughout this thesis and remind some state of the art investigations namely in the fields of geometric modeling, algorithmic geometry and algebraic geometry.

Second, the main theoretical achievement of the thesis, arrangement computation, is presented. Chapter 4 introduces the generic subdivision approach and the next two chapters specialize it to the cases of curves and surfaces. The latter are followed by the evocation of immediate yet very useful applications of the algorithm. Most of the necessary mathematical background is presented in chapter 1, however, some concepts that are used only once are discussed right where they are introduced.

Chapter 8 illustrates the main technical achievement of the thesis, the pro-

totyping of a new kind of software to let the disjoint fields of algebraic and algorithmic geometry meet in a geometric modeling context. The appendices give more details about its data file formalism, its highly modular architecture using dynamic kernels and plugins built on top of external computational libraries as a support and finally a first attempt at using its internals in a virtual reality environment.

Chapter 1

Algebraic preliminaries

Curves and surfaces in algebraic geometry are usually represented by piecewise polynomial equations of various types. Many interrogations on such objects such as intersection or self-intersection reduce to solving systems of non linear polynomial equations in either monomial or Bernstein basis. Many geometric operations use projection techniques for which a resultant computation is a very convenient algebraic tool. Subdivision solving methods provide approximate solutions but algebraic numbers allow to exactly represent roots of polynomials.

This chapter presents some algebraic ingredients that will be extensively used in the different computations proposed in this thesis.

1.1 Bernstein basis

The *Bernstein basis* is often used in a subdivision process involving algebraic geometric objects since they have a number of useful properties [48] which make them a convenient tool to encode their representation within a given domain. The representation of a polynomial in the Bernstein basis is known to be numerically more stable than the monomial basis representation [49].

Moreover, it has a direct geometric meaning, in terms of control points and useful properties such as the convex hull and the variation diminishing properties.

1.1.1 Univariate Bernstein basis

Given an arbitrary univariate polynomial function $f(x) \in \mathbb{K}$, we can convert it to a representation of degree d in Bernstein basis, which is defined by:

$$f(x) = \sum_i b_i B_i^d(x) \tag{1.1}$$

$$B_i^d(x) = \begin{pmatrix} d \\ i \end{pmatrix} x^i (1-x)^{d-i} \tag{1.2}$$

where b_i is usually referred to as control coefficients. The above formula can be generalized to an arbitrary interval $[a, b]$ by a variable substitution $x' = (b-a)x + a$. We denote by $B_d^i(x; a, b) = \binom{d}{i}(x-a)^i(b-x)^{d-i}(b-a)^{-d}$ the corresponding Bernstein basis on $[a, b]$.

There are several useful properties regarding Bernstein basis given as follows.

Property 1.1 (Convex-Hull): Since $\sum_i B_d^i(x) \equiv 1$ and $B_d^i(x; a, b) \geq 0$ for all $x \in [a, b]$, where $i = 0, ..., d$, the graph of $f(x) = 0$, which is given by $(x, f(x))$, should always lie within the convex-hull defined by the control coefficients $(\frac{i}{d}, b_i)$ [48].

Property 1.2 (De Casteljau Subdivision): Given $x_0 \in [0, 1]$, $f(x)$ can be represented piece-wisely by:

$$f^{(0)}(x) = \sum_{i=0}^{d} b_0^{(i)} B_i^d(x), x \in [0, x_0] \tag{1.3}$$

$$f^{(1)}(x) = \sum_{i=0}^{d} b_i^{(d-i)} B_i^d(x), x \in [x_0, 1] \tag{1.4}$$

$$b_i^{(k)} = (1 - x_0) b_i^{(k-1)} + x_0 b_{i+1}^{(k-1)} \tag{1.5}$$

Definition 1.1: The number of sign changes $V(\mathbf{b})$ is defined recursively for a sequence of coefficients $\mathbf{b}_k = b_1 \dots b_k$:

$$V(\mathbf{b}_{k+1}) = V(\mathbf{b}_k) + \begin{cases} 1, \text{ if } b_i b_{i+1} < 0 \\ 0, \text{ otherwise} \end{cases} \tag{1.6}$$

1.1.2 Multivariate Bernstein basis

The univariate Bernstein basis representation can be generalized to multivariate ones. Briefly speaking, we can rewrite the definition (see equation 1.1) in the form of tensor products. Suppose for $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^n$, $f(\mathbf{x}) \in \mathbb{K}[\mathbf{x}]$ having the maximum degree $\mathbf{d} = (d_1, ..., d_n)$ has the form:

$$f(\mathbf{x}) = \sum_{k_1=0}^{d_1} ... \sum_{k_n=0}^{d_n} b_{k_1...k_n} B_{k_1}^{d_1}(x_1)...B_{k_n}^{d_n}(x_n) \tag{1.7}$$

The De Casteljau subdivision for the multivariate case proceeds similarly to the univariate one, since the subdivision can be done independently with regards to a particular variable x_i. The Descartes' law also applies for the multivariate case. For a polynomial of n variables, the coefficients can be viewed as a tensor or dimension n.

1.2 Bernstein solvers

A critical operation, which we will have to perform in geometric computations on curves and surfaces, is to solve polynomial equations. In such computations, we start with input polynomial equations (possibly with some incertitude on the coefficient) and we want to compute an approximation of the real roots of these equations or boxes containing these roots. Such operation should be performed very efficiently and with guarantee, since they will be used intensively in geometric computations.

This section describes subdivision solvers which are based on *certified* exclusion criteria. In other words, starting from an initial bounded domain, sub-domains which are guaranteed not to contain a real solution of the polynomial equations are removed. A parameter $\epsilon > 0$ is controlling the size of the boxes that are kept while existence and uniqueness criteria are applied to produce certified isolation intervals which contain a single root. The interest of these subdivision methods (*e.g.* [63]), compared to homotopy solvers [103], [60] or algebraic solvers [85] is that only local information related to the initial domain are used, avoiding an approximation of all the complex roots of the system. The methods are particularly efficient for systems where

the number of real roots is much smaller that the number of complex roots or where the complex roots are far from the domain of interest. Multiple roots however usually reduce their performance if their isolation is required, in addition to their approximation.

1.2.1 Univariate Bernstein solver

Let us consider first an exact polynomial $f = \sum_{i=0}^{d} a_i x^i \in \mathbb{Q}[x]$. Our objective is to isolate the real roots of f, *i.e.* to compute intervals with rational endpoints that contain one and only one root of f, as well as the multiplicity of every real root. Here is the general scheme of the subdivision solver that we consider, augmented appropriately so that it also outputs the multiplicities. It uses an external function $V(f, I)$, which bounds the number of roots of f in the interval I (see property 1.1).

Algorithm 1.1: Real root isolation

Input: A polynomial $f \in \mathbb{Z}[x]$, such that $\deg(f) = d$.

Output: A list of intervals with rational endpoints, which contain one and only one real root of f and the multiplicity of every real root.

Compute the square-free part of f, *i.e.* f_{red} ;
Compute an interval $I_0 = (-B, B)$;
Initialize a queue Q with I_0 ;
while $Q \neq \emptyset$ **do**
    Pop an interval I from Q and compute $v := V(f, I)$;
    if $v = 0$ **then** discard I ;
    if $v = 1$ **then** output I ;
    if $v \geq 2$ **then** split I into I_L and I_R and push them to Q ;
end
Determine the multiplicities of the real roots, using the square-free factorization of f ;

Another interesting property of the univariate Bernstein representation related to the Descartes' law of signs (see proposition 1.3) is that there is a simple and yet efficient test for the existence of real roots in a given interval.

Property 1.3 (Descartes' Law of signs): Given a polynomial $f(x) = \sum_i^n b_i B_i^d(x; a, b)$, the number N of real roots of f on $]a, b[$ is less than or

equal to $V(\mathbf{b})$, where $\mathbf{b} = (b_i)_{i=1\ldots n}$ and $N \equiv V(\mathbf{b}) \mod 2$.

With this property,

- if $V(\mathbf{b}) = 0$, the number of real roots of f in $[a, b]$ is 0.
- if $V(\mathbf{b}) = 1$, the number of real roots of f in $[a, b]$ is 1.

The approach can also be extended to polynomials with interval coefficients, by counting 1 sign variation for a sign sub-sequence $(+, ?, -)$ or $(-, ?, +)$, 2 sign variations for a sign sub-sequence $(+, ?, +)$ or $(-, ?, -)$, 1 sign variation for a sign sub-sequence $(?, ?, ?)$, where ? is the sign of an interval containing 0. Again in this case, if a family $\overline{f}$ of polynomials is represented by the sequence of intervals $\bar{\mathbf{b}} = [\bar{b}_0, \ldots, \bar{b}_d]$ in the Bernstein basis of the interval $[a, b]$:

- if $V(\bar{\mathbf{b}}) = 1$, all the polynomials of the family $\overline{f}$ have one root in $[a, b]$,
- if $V(\bar{\mathbf{b}}) = 0$, all the polynomials of the family $\overline{f}$ have no roots in $[a, b]$.

This subdivision algorithm, using interval arithmetic, yields either intervals of size smaller than ϵ, which might contain the roots of $f = 0$ in $[a, b]$ or isolating intervals for all the polynomials of the family defined by the interval coefficients.

1.2.2 Multivariate Bernstein solver

We consider here the problem of computing the solutions of a polynomial system:

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ \vdots \\ f_s(x_1, \ldots, x_n) = 0 \end{cases} \tag{1.8}$$

in an interval $I := [a_1, b_1] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$. The method for approximating the real roots of this system, that we describe now, uses the representation of multivariate polynomials in Bernstein basis, analysis of sign variations and univariate solvers (section 1.2.1). The output is a set of small-enough boxes, which contain these roots. This subdivision solver which can be seen as an improvement of the *Interval Projected Polyhedron* algorithm in [97], is described in more details in [80].

In the following, we use the Bernstein basis representation of a multivariate polynomial f in the domain $I := [a_1, b_1] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$:

$$f(x_1, \ldots, x_n) = \sum_{i_1=0}^{d_1} \cdots \sum_{i_n=0}^{d_n} b_{i_1,\ldots,i_n} \, B_{i_1}^{d_1}(x_1; a_1, b_1) \cdots B_{i_n}^{d_n}(x_n; a_n, b_n).$$

Definition 1.2: For any $f \in \mathbb{R}[\mathbf{x}]$ and $j = 1, \ldots, n$, let:

$$m_j(f; x_j) = \sum_{i_j=0}^{d_j} \min_{\{0 \le i_k \le d_k, k \ne j\}} b_{i_1,\ldots,i_n} \, B_{i_j}^{d_j}(x_j; a_j, b_j) \tag{1.9}$$

$$M_j(f; x_j) = \sum_{i_j=0}^{d_j} \max_{\{0 \le i_k \le d_k, k \ne j\}} b_{i_1,\ldots,i_n} \, B_{i_j}^{d_j}(x_j; a_j, b_j) \tag{1.10}$$

Theorem 1.1 (Projection Lemma): For any $\mathbf{u} = (u_1, \ldots, u_n) \in I$, and any $j = 1, \ldots, n$, we have

$$m(f; u_j) \le f(\mathbf{u}) \le M(f; u_j) \tag{1.11}$$

As a direct consequence, we obtain the following corollary:

Corollary 1.1: For any root $\mathbf{u} = (u_1, \ldots, u_n)$ of the equation $f(\mathbf{x}) = 0$ in the domain I, we have $\underline{\mu}_j \le u_j \le \overline{\mu}_j$ where:

1. $\underline{\mu}_j$ (resp. $\overline{\mu}_j$) is either a root of $m_j(f; x_j)$ (resp. of $M_j(f; x_j)$) in $[a_j, b_j]$ or equal to a_j (resp. b_j) in the case where $m_j(f; x_j)$ (resp. $M_j(f; x_j)$) has no root on $[a_j, b_j]$
2. $m_j(f; u) \le 0 \le M_j(f; u)$ on $[\underline{\mu}_j, \overline{\mu}_j]$.

The solver proceeds in the following main steps: 1. apply a preconditioning step to the equations, 2. reduce the domain, 3. if the reduction ratio is too small, split the domain, until the size of the domain is smaller than a given epsilon.

The following important ingredients parameterize the algorithm.

Preconditioning strategy. That is, a transformation of the initial system into a system, which has a better numerical behavior. Solving the

system $\mathbf{f} = 0$ is equivalent to solving the system $M\,\mathbf{f} = 0$, where M is an $s \times s$ invertible matrix. As such a transformation may increase the degree of some equations, with respect to some variables, it has a cost, which might not be negligible in some cases. Moreover, if for each polynomial of the system not all the variables are involved, that is if the system is sparse with respect to the variables, such a preconditioner may transform it into a system which is not sparse anymore. In this case, we would prefer a partial preconditioner on a subset of the equations sharing a subset of variables. We consider global transformations, which minimize the distance between the equations, considered as vectors in an affine space of polynomials of a given degree and local straightening (for $s = n$), which locally transform the system $\mathbf{f}$ into a system $J^{-1}\mathbf{f}$, where $J = (\partial_{x_i} f_j(\mathbf{u}))_{1 \le i,j \le s}$ is the Jacobian matrix of $\mathbf{f}$ at a point $\mathbf{u}$ of the domain I, where it is invertible.

It can be proved that the reduction based on the polynomial bounds m and M behaves like Newton iteration near a simple root, that is, we have a quadratic convergence with this transformation.

Reduction strategy. That is, the technique used to reduce the initial domain for searching the roots of the system. It can be based on convex hull properties as in [97] or on root localization, which is a direct improvement of the convex hull reduction and consists in computing the first (resp. last) root of the polynomial $m_j(f_k; u_j)$, (resp. $M_j(f_k; u_j)$), in the interval $[a_j, b_j]$. The current implementation of this step allows us to consider the convex hull reduction, as one iteration step of this reduction process.

The guarantee that computed intervals contain the roots of f, is obtained by controlling the rounding mode of the operations during the De Casteljau computation.

Subdivision strategy. That is, the technique used to subdivide the domain, in order to simplify forthcoming steps for searching the roots of the system. Some simple rules can be used to subdivide a domain and reduce its size. The approach, that we are using in our implementation is the parameter domain bisection: a domain b is then split in half in a direction j for which $|b_j - a_j|$ is maximal. But instead of choosing the size of the interval as a criterion for the direction in which we split, we may choose

another criterion depending also on the values of the functions m_i, M_j or f_j (for instance where $M_j - m_j$ is maximal).

A bound for the complexity of this method is detailed in [80]. It involves metric quantities related to the system $\mathbf{f} = 0$, such as the Lipschitz constant of $\mathbf{f}$ in B, the entropy of its near-zero level sets, a bound d on the degree of the equations in each variable and the dimension n.

1.3 Algebraic numbers

Algebraic numbers are of particular importance in geometric problems such as arrangement or topology computation. In geometric modeling the treatment of algebraic curves or surfaces implicitly or explicitly leads to the manipulation of algebraic numbers. They provide an exact representation of the roots of the polynomials defining theses curves or surfaces.

An *algebraic number* is a root of a polynomial $p(x)$ with coefficients in $\mathbb{K}$ ($p(x) \in \mathbb{K}[x]$). An *algebraic integer* is a root of a polynomial where the leading coefficient is 1.

Let α be an algebraic number and $p(x)$ be a polynomial of degree d with $p(\alpha) = 0$. If $p(x)$ is irreducible over $\mathbb{K}$ (cannot be written in $\mathbb{K}[x]$ as the product of two polynomials different from 1), it is called the *minimal polynomial* of α. The other roots $\alpha_2, \ldots, \alpha_d$ of the minimal polynomial in $\overline{\mathbb{K}}$ are the *conjugates* of α. The *degree* of the algebraic number α is the degree of the minimal polynomial defining α. If α, β are algebraic numbers, then $\alpha \pm \beta, \alpha \cdot \beta, \alpha/\beta$ (if $\beta \neq 0$) and $\sqrt[k]{\alpha}$ are algebraic numbers. If α, β are algebraic integers, then $\alpha \pm \beta, \alpha \cdot \beta$ and $\sqrt[k]{\alpha}$ are algebraic integers.

A natural way to encode a real algebraic number α over $\mathbb{Q}$ is by using a polynomial $p(x)$ of $\mathbb{Q}[x]$, which vanishes at α, and an *isolating interval* $[a, b]$ containing α such that $a, b \in \mathbb{Q}$ and $p(x)$ has exactly one real root in $[a, b]$.

This representation is not unique, since the size of the interval $[a, b]$ can reduce to any $\epsilon > 0$ close to 0. If we assume moreover that P is a square-free polynomial ($\gcd(p, p') = 1$), then α is a simple root of p and the sign of p changes at α.

Remark 1.1: An alternative representation is *Thom encoding* [12]. The basic idea behind this representation is that the signs of all derivatives of p obtained by evaluation over the real roots of p uniquely characterize and order these real roots. This representation beside the uniqueness property is also more general than the isolating interval representation. •

To compare algebraic numbers we have recourse to another algebraic tool, namely, Sturm sequences.

Definition 1.3 (Sturm sequence): Let p, q be two univariate polynomials. A polynomial sequence $f_0 = p, f_1 = q, \ldots, f_s$ is a Sturm sequence if:

1. f_s divides all the f_i, $i = 1 \ldots s$. Let $\delta_i = f_i/f_s$, $i = 1 \ldots s$.

2. If c is a real number such that $\delta_j(c) = 0$ with $0 < j < s$ then

$$\delta_{j-1}(c)\delta_{j+1}(c) < 0 \tag{1.12}$$

3. If c is a real number such that $\delta_0(c) = 0$ then $\delta_0(x)\delta_1(x)$ has the sign of $x - c$ in a neighborhood of c.

For any sequence S of real polynomials and $a \in \mathbb{R}$, we denote by $V(S, a)$ the number of variations of signs of the values of the polynomials in S at a. Then we have the well-known theorem of Sturm (see for instance [12]).

Proposition 1.1 (Sturm theorem): Assume $S = \text{Sturm}(p, p'q)$ and $]a, b[$ is an interval such that $p(a)p(b) \neq 0$. The difference $V(S, a) - V(S, b)$ is equal to the difference between the number of roots α of p in $]a, b[$ (without multiplicity) such that $q(\alpha) > 0$ and the number of roots α of p in $]a, b[$ such that $q(\alpha) < 0$:

$$V(S, a) - V(S, b) = Z_{q>0}(p) - Z_{q<0}(p) \tag{1.13}$$

Remark 1.2: If in proposition 1.1, p or q is square-free, the computation of $\text{Sturm}(p, q)$ is sufficient. •

We recall here the definition of pseudo-remainder as defined in the book of Basu-Pollack and Roy [12].

Definition 1.4 (Pseudo-remainder): Let

$$P = a_p x^p + \cdots + a_0 \tag{1.14}$$

$$Q = b_q x^q + \cdots + b_0 \tag{1.15}$$

be two polynomials in $D[x]$ where D is a subring of $\mathbb{R}$. Note that the only denominators occurring in the euclidean division of P by Q are $b_q^i, i \leq p + q - 1$.

The signed pseudo-remainder denoted $\mathrm{Prem}(P, Q)$, is the remainder in the euclidean division of $b_q^d P$ by Q, where d is the smallest even integer greater than or equal $p - q + 1$. Note that the euclidean division of $b_q^d P$ by Q can be performed in D and that $\mathrm{Prem}(P, Q) \in D[x]$.

An efficient way to compute a Sturm sequence is to compute a Sturm-Habicht sequence.

Definition 1.5 (Sturm-Habicht sequence): Let p and q be univariate polynomials, $d = \max(\deg(p), \deg(q) + 1)$, $\mathrm{coef}_k(p)$ the coefficient of x^k in p, and $\delta_k = (-1)^{k(k-1)/2}$.

The Sturm-Habicht sequence of p and q is defined inductively as follows:

1. $H_d = p$, $h_d = 1$
2. $H_{d-1} = q$

Assume that we have computed $H_d, \ldots, H_{j-1}$, $h_d, \ldots, h_j$ with $h_j \neq 0$ and $H_{j-1} \neq 0$. Let $k = deg(H_{j-1})$. Then if $k < j-1$, let $H_k = \delta_{j-k} \frac{\mathrm{coef}_k(H_{j-1})^{j-1-k}}{h_j^{j-1-k}} H_{j-1}$, $h_{j-1} = 1$ for $l \in \mathbb{N}$ with $k < l < j - 1$, let $H_l = 0$, $h_l = 0$, $h_k = \mathrm{coef}_k(H_k)$ $H_{k-1} = \delta_{j-k+2} \frac{\mathrm{Prem}(H_j, H_{j-1})}{h_j^{j-k+1}}$.

This Sturm-(Habicht) sequence can also be useful for gcd computations, since the gcd corresponds to the last non-zero term of the sequence. In particular, it yields a way to compute the square-free part $p/(\gcd(p, p')$ of a polynomial $p \in \mathbb{Q}[x]$.

Comparison. Let us describe briefly how we use Sturm's theorem to compare two algebraic numbers $\alpha = (p,]a, b[)$ and $\beta = (q,]c, d[)$, assuming for

simplicity that α and β are *simple roots* of p and q. If $b < c$ (resp. $d < a$) we have $\alpha < \beta$ (resp. $\beta < \alpha$). Let us assume now that $a < c < b < d$ (the other cases being treated similarly). First we compute the sign s of $p(a)p(c)$. If $s < 0$, then we have $\alpha \in]a, c[$ and $\alpha < \beta$. If $s = 0$, we have $\alpha = c$ (since $\alpha \neq a$), which implies that $\alpha < \beta$. Otherwise $s > 0$, p has no root in the interval $[a, c]$. We compute $S = \text{Sturm}(p, p'q)$ and $v := V(S, c) - V(S, b)$. Let us assume first that $q(c) > 0$, $q(b) < 0$. Then if $v = 1$, by Sturm's theorem $q(\alpha) > 0$ and $\alpha < \beta$. If $v = -1$, $q(\alpha) < 0$ and $\alpha > \beta$. If $v = 0$, then $q(\alpha) = 0$ and $\alpha = \beta$. If now $q(c) < 0$, $q(b) > 0$, we negate the previous output. Finally, if $q(c)$ and $q(b)$ are of the same sign, then $\alpha < \beta$.

1.4 Resultants

A projection operator is an operator which associates to an overdetermined polynomial system in several variables, a polynomial depending only on the coefficients of this system, which vanishes when the system has a solution.

Let us first overview the case of two univariate polynomials. Given two polynomials f and $g \in \mathbb{K}[x]$ of positive degree, say

$$f = a_0 x^l + \cdots + a_l, \qquad a_0 \neq 0,\ l > 0 \tag{1.16}$$

$$g = b_0 x^m + \cdots + b_m, \qquad b_0 \neq 0,\ m > 0 \tag{1.17}$$

the resultant of f and g, denoted $\text{Res}(f, g)$, is the determinant of the $(l + m) \times (l + m)$ matrix

$$\text{Res}(f,g) = \det \begin{pmatrix} a_0 & & & & b_0 & & & \\ a_1 & a_0 & & & b_1 & b_0 & & \\ a_2 & a_1 & \ddots & & b_2 & b_1 & \ddots & \\ \vdots & a_2 & \ddots & a_0 & \vdots & b_2 & \ddots & b_0 \\ a_l & \vdots & \ddots & a_1 & b_m & \vdots & \ddots & b_1 \\ & a_l & & a_2 & & b_m & & b_2 \\ & & \ddots & \vdots & & & \ddots & \vdots \\ & & & a_l & & & & b_m \end{pmatrix} \tag{1.18}$$

where empty positions are filled with zeroes is called the *Sylvester matrix* of

f and g.

Remark 1.3: The two most common representations for the resultant of two polynomials in one variable are the Sylvester matrix and the Bezout matrix [106]. Each entry of the Sylvester matrix is either zero or a coefficient of one of the original polynomial equations. The entries of the Bezout matrix are more complicated, but the Bezout resultant employs a much smaller matrix. Resultants of two polynomials in one variable can also be represented by hybrids of the Sylvester and Bezout matrices as well as by companion matrices (see example 1.1). •

Property 1.4 (Integer polynomial): Res(f, g) is an integer polynomial in the coefficients of f and g.

Property 1.5 (Common factor): $\text{Res}(f, g) = 0$ if and only if f and g have a common factor in $\mathbb{K}[x]$.

Property 1.6 (Elimination): There are polynomials A and $B \in \mathbb{K}[x]$ such that $Af + Bg = \text{Res}(f, g)$. The coefficients of A and B are integer polynomials in the coefficients of f and g.

Resultants are also useful to compute the squarefree part of a polynomial.

Theorem 1.2: Let $f, g \in \mathbb{K}[x]$ be two polynomials of degrees $\deg(f) = n > 0$ and $\deg(g) = m > 0$. Then f and g have a common factor of degree greater than $l \geq 0$ if and only if there are polynomials A and B in $\mathbb{K}[x]$, with $\deg(A) < m - l$ and $\deg(B) < n - l$ which are not both zero, and such that $Af + Bg = 0$.

As an immediate consequence we obtain a statement about the degree of the greatest common divisor of f and g.

Corollary 1.2: The degree of the gcd of two polynomials $f, g \in \mathbb{K}[x]$ is equal to the smallest index h such that for all polynomials A and $B \in \mathbb{K}[x]$, with $\deg(A) < m - h$ and $\deg(B) < n - h$: $Af + Bg \neq 0$.

Corollary 1.3: The degree of the gcd of two polynomials $f, g \in \mathbb{K}[x]$ is equal to the smallest index h such that for all rational polynomials A and

B with $\deg(A) < m - h$ and $\deg(B) < n - h$: $\deg(Af + Bg) \geq h$.

We are interested in determining the degree of the greatest common divisor of two polynomials f and g. According to corollary 1.3 we have to test in succession whether for $l = 1, 2, 3, \ldots$ there exist polynomials A and B, with the claimed restriction of the degrees such that the degree of $Af + Bg$ is strictly smaller than l. The first index h, for which this test gives a negative answer, is equal to the degree of the gcd. The test for $l = 0$ can be made by testing whether the resultant of f and g is equal to zero. For $l = 1, 2, 3, \ldots$, we proceed in a similar way. Let l be a fixed index and let

$$f(x) = f_n x^n + f_{n-1} x^{n-1} + \cdots + f_0, \tag{1.19}$$

$$g(x) = g_m x^m + g_{m-1} x^{m-1} + \cdots + g_0. \tag{1.20}$$

We are looking for two polynomials

$$A(x) = a_{m-l-1} x^{m-l-1} + \cdots + a_1 x + a_0 \tag{1.21}$$

$$B(x) = b_{n-l-1} x^{n-l-1} + \cdots + b_1 x + b_0, \tag{1.22}$$

such that $\deg(Af + Bg) < l$. There are $m + n - 2l$ unknown coefficients $a_{m-l-1}, \ldots, a_0, b_{n-l-1}, \ldots, b_0$. The polynomial $A(x)f(x) + B(x)g(x)$ has degree at most $n+m-l-1$. The $m+n-2l$ coefficients of $x^l, x^{l+1}, \ldots, x^{m+n-l-1}$ have to be zero in order to achieve $\deg(Af + Bg) < l$. This leads to a linear system

$$(a_{m-l-1}, \ldots, a_0, b_{n-l-1}, \ldots, b_0) \cdot S_l = (0, \ldots, 0) \tag{1.23}$$

where S_l is the submatrix of the Sylvester matrix of f and g obtained by deleting the last $2l$ columns, the last l rows of f-entries, and the last l rows of g-entries. We call $\mathrm{Sr}_l(f, g) = \det S_l$ the l^{th} *subresultant* of f and g. For $l = 0$, the equality $\mathrm{Res}(f, g) = \mathrm{Sr}_0(f, g)$ holds. In fact, S_l is a submatrix of S_i for $l > i \geq 0$. The $2\,l \times 2\,l$ minors of the submatrix of the Sylvester matrix of f and g obtained by deleting the last l rows of f-entries, can be collected in order to construct a polynomial, which has interesting properties. To be more specific, we need the following definition.

Definition 1.6 (Determinant polynomial): Let $\mathcal{M}$ be a $s \times t$ matrix, $s \leq t$, over an integral domain D. The determinant polynomial of $\mathcal{M}$ is:

$$\text{detpol}(\mathcal{M}) = |\mathcal{M}_s|\, x^{t-s} + \cdots + |\mathcal{M}_t| \tag{1.24}$$

where $\mathcal{M}_j$ denotes the submatrix of $\mathcal{M}$ consisting of the first $s-1$ columns followed by the j^{th} column, for $s \leq j \leq t$.

Definition 1.7 (Subresultant): Let $f, g \in \mathbb{K}[x]$, two polynomials with $deg(f) = n > 0$, $deg(g) = m > 0$. For $0 \leq l \leq \min(f, g)$, we define:

$$M_l = \text{mat}(x^{n-l-1}f(x), x^{n-l-2}f(x), \ldots, f(x), x^{m-l-1}g(x), \ldots, g(x)) \tag{1.25}$$

Then the l^{th} subresultant polynomial of f and g is $S_l = \text{Sr}_l(f, g) = \text{detpol}(M_l)$.

Notice that the coefficient of x^l in $\text{Sres}_l(P, g)$ is the l^{th} subresultant coefficient, denoted $\text{Sr}_j(f, g)$.

Proposition 1.2: [12, 104] Two polynomials f and g of positive degree have a gcd of degree h if and only if h is the least index l for which $\text{Sr}_l(f, g) \neq 0$. In this case, their gcd is $\text{Sr}_l(f, g)(x)$.

Using subresultants, one can compute the squarefree part of a univariate polynomial, as shown in algorithm 1.2.

Algorithm 1.2: Square free part of a univariate polynomial

Input: A polynomial $f \in \mathbb{K}Z[x]$.

Output: The squarefree part f^r of f.

Compute the last non-zero subresultant $\text{Sr}(x)$ of $f(x)$ and $f'(x)$;

Compute $f^r = f/\text{Sr}(x)$;

Example 1.1 (Companion matrices): Using these resultant matrix formulations, solving a polynomial problem can be reduced to solving the generalized eigenvector problem $T^t(x)\mathbf{v} = 0$, where $T(x)$ is a matrix of size $N \times N$ with polynomial coefficients, or equivalently a polynomial with $N \times N$ matrix coefficients. If $d = \max_{i,j}\{\deg(T_{ij}(x))\}$, we obtain $T(x) = T_d x^d + T_{d-1}x^{d-1} + \cdots + T_0$, where T_i are $n \times n$ matrices. The problem is

transformed into a generalized eigenvalue problem $(A - \lambda B)\,\mathbf{v} = 0$:

$$A = \begin{pmatrix} 0 & I & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I \\ T_0^t & T_1^t & \cdots & T_{d-1}^t \end{pmatrix}, \; B = \begin{pmatrix} I & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & I & 0 \\ 0 & \cdots & 0 & -T_d^t \end{pmatrix} \tag{1.26}$$

where A, B are $N \times d$ constant matrices. We have the following property:

$$T^t(x)\,\mathbf{v} = 0 \Leftrightarrow (A - xB) \begin{pmatrix} \mathbf{v} \\ x\,\mathbf{v} \\ \vdots \\ x^{d-1}\,\mathbf{v} \end{pmatrix} = 0. \tag{1.27}$$

This applies for implicit curve intersection problems, like in [21]. Given two polynomials $p, q \in \mathbb{Q}[x, y]$, we compute their resultant matrix, with respect to y. This yields a matrix $T(x)$, from which we deduce the coordinates of the intersection points by solving the generalized eigenvector problem $T(x)^t\,\mathbf{v} = 0$. ∎

Generalizing this, suppose we are given $n + 1$ homogeneous polynomials $f_0, \ldots, f_n$ in variables $x_0, \ldots, x_n$ and assume each f_i has positive total degree. Then we get $n + 1$ equations in $n + 1$ unknowns:

$$\begin{cases} f_0(x_0, \ldots, x_n) & = 0 \\ \vdots & \\ f_n(x_0, \ldots, x_n) & = 0 \end{cases} \tag{1.28}$$

Since f_i are homogeneous of positive total degree, these equations always have the solution $x_0 = \cdots = x_n = 0$, that we call the *trivial* solution. Hence, the crucial question is whether there is a *non trivial* solution. In general, the existence of a non trivial solution depends on the coefficients of the polynomials $f_0, \ldots, f_n$: for most values of the coefficients, there are no non trivial solution, while for certain values, some exist.

Theorem 1.3: If we fix positive degrees $d_0, \ldots, d_n$, then there is a unique

integer polynomial Res in the coefficients of $f_0, \ldots, f_n$ which has the following properties:

1. $\text{Res}(f_0, \ldots, f_n) = 0$ if $(f_0, \ldots, f_n)$ have a common factor in $\mathbb{K}[x]$
2. Res is irreducible.

Remark 1.4: Resultants for multivariate polynomials have at least dozen different representations many of which are valid in only special cases. The Sylvester formulation represents the resultant by a single determinant each of whose entries is either zero or one of the coefficients of the original polynomial equations. The Dixon formulation uses smaller matrices than the Sylvester one, but employs more complicated Bezoutian entries [29]. Macaulay represents the resultant as the ratio of two determinants whose entries are similar to the Sylvester resultant [77]. There are also hybrid representations and for low degree polynomial systems there are special formulations such as the Jacobian representation [107]. •

Remark 1.5: There are two other techniques known in the literature for eliminating a set of variables. They are Grobner bases and Ritt-Wu's algorithm. The algorithm for Grobner bases generates special bases for polynomial ideals and was originally formulated by Buchberger in [18]. Eliminating a set of variables is a special application of Grobner bases. Ritt-Wu's algorithm for variable elimination has been developed in [105] using an idea proposed by Ritt [93]. •

Chapter 2

Geometric preliminaries

Geometric modeling traditionally identifies a body of techniques that can model certain classes of curves or surfaces subject to particular conditions of shapes and smoothness. It has important applications in several industries including automobile, aerospace, architecture *etc.*

Curves and surfaces are main entities that can be used to describe realistic shapes. There is therefore an emerging need to research the use of such geometric entities and techniques to interrogate them. Sections 2.1 and 2.2 define such curves and surfaces.

Combined together, they describe solids representing shapes of physical objects mainly relying on two representations: boundary representation and constructive representation, briefly presented in section 2.3.

The representation of a shape is then a paramount choice for a geometric or solid modeling software since modeling techniques are intrinsically linked to the nature of the geometric objects. In section 2.4 we briefly overview some software that propose curves and surfaces for geometric modeling.

2.1 Curves

Some important graphic problems are two dimensional and can be solved with curves. Examples are technical drawing or Computer Aided Machining

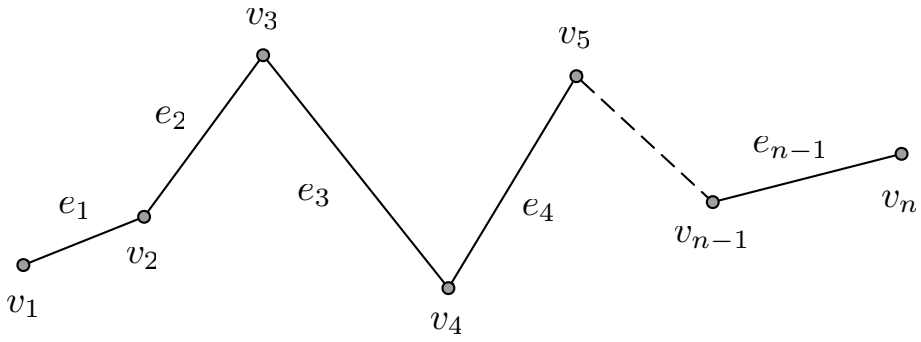


Figure 2.1: A piecewise linear curve.

(CAM) in which material is cut along a given curve. A lot of research effort has gone into curves and many methods are known, dealing with rather general curves that can be classified into three main families: piecewise linear curves, parametric curves or implicit curves.

2.1.1 Piecewise linear curves

A piecewise linear curve is a sequence of connected line segments. The term is adequate in an algebraic geometry context to mark the difference between linear and non linear representations of curves but some readers may recognize this object as a planar mesh constituted of a set of vertices and a set of edges together with adjacency relationships.

Figure 2.1 illustrates a piecewise linear curve composed of n vertices $v_1 \dots v_n$ and $n-1$ edges $e_1 \dots e_{n-1}$.

2.1.2 Parametric curves

Parametric representations are the most common in computer graphics since they are, by nature, both easy to visualize and, in the case of splines, easy to manipulate.

The representations of curves that we mention in what follows are based on polynomials since they are simple functions that are easy to calculate and flexible enough to create many different shapes. Any function can however

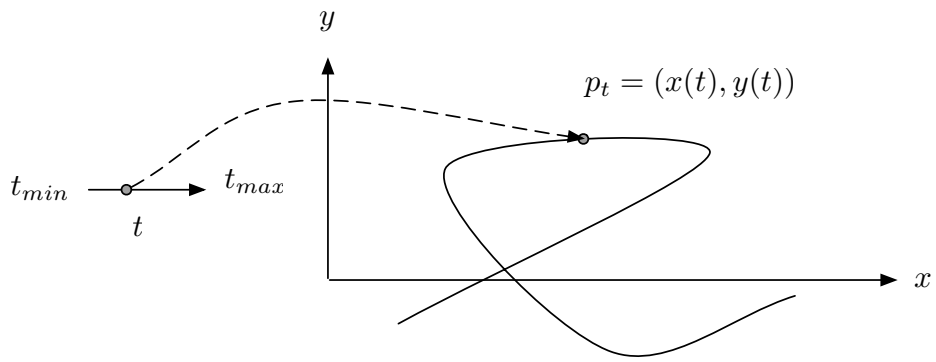


Figure 2.2: A rational curve.

be used to create a parametric curves *e.g.* trigonometric expressions.

Polynomial rational curves. A uniform rational polynomial curve is defined by the formula

$$c(t) = \begin{cases} x(t)/w(t) \\ y(t)/w(t) \end{cases} \tag{2.1}$$

where $x : \mathbb{R}[t] \mapsto \mathbb{R}$, $y : \mathbb{R}[t] \mapsto \mathbb{R}$ and $w : \mathbb{R}[t] \mapsto \mathbb{R}$ are polynomial functions evaluated to obtain the image of $t \in \mathbb{R}[t]$ by c in $\mathbb{R}^2$ as the point $p_t = (x(t)/w(t), y(t)/w(t))$.

A uniform non-rational polynomial curve is derived by the previous formula when $w(t) = 1$ or defined by the formula

$$c(t) = \begin{cases} x(t) \\ y(t) \end{cases} \tag{2.2}$$

B-spline curves. A B-spline curve is defined by the formula

$$c(t) = \sum_{i=1}^{n} \mathbf{p_i} B_{i,k,t}(t) \tag{2.3}$$

as a linear combination of a sequence of control points and B-spline basis functions $B_{i,k,t}$ uniquely determined by a knot vector $\mathbf{t}$ and the order k.

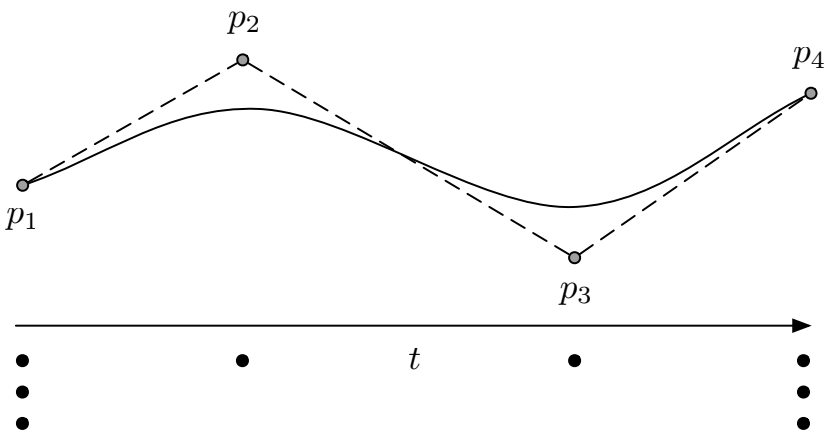


Figure 2.3: A B-spline curve.

The complete representation of a B-spline curve consists of:

d The dimension of the underlying Euclidean space

n The number of vertices

k The order of the B-spline

$\mathbf{t}$ The knot vector of the B-spline: $\mathbf{t} = (t_1, t_2, \ldots, t_{n+k})$

$\mathbf{p}$ The control points of the B-spline curve

The parameter range of a B-spline curve c is the interval $[t_k, t_{n+1}]$ and so mathematically, the curve is a mapping $c : [t_k, t_{n+1}] \mapsto \mathbb{R}^d$, where d is the Euclidean space dimension of its control points.

The data in the representation must satisfy certain conditions: 1. The knot vector must be non-decreasing: $t_i \leq t_{i+1}$. Moreover, two knots t_i and t_{i+k} must be distinct: $t_i < t_{i+k}$. 2. The number of vertices should be greater than or equal to the order of the curve: $n \geq k$.

A B-spline of order k is the sum of two B-splines of order $k-1$, each weighted with weights in the interval $[0, 1]$. In fact we define B-splines of order 1 explicitly as box functions,

$$B_{i,1}(t) = \begin{cases} 1 \text{ if } t_i \leq t \leq t_{i+1} \\ 0 \text{ otherwise} \end{cases} \tag{2.4}$$

and then the complete definition of a k-th order B-spline is

$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} B_{i-1,k-1}(t) \tag{2.5}$$

B-splines satisfy some important properties for curve and surface design. Each B-spline is non-negative and it can be shown that they sum to one,

$$\sum_{i=1}^{n} B_{i,k,t}(t) = 1 \tag{2.6}$$

B-spline curves satisfy the *convex hull property*: the curve lies in the convex hull of its control points. Furthermore, the support of the B-spline $B_{i,k,t}$ is the interval $[t_i, t_{i+k}]$ which means that B-spline curves have local control: moving one control point only alters the curve locally.

The *control polygon* of a B-spline curve is the polygonal arc formed by its control points, $\mathbf{p_1}, \ldots, \mathbf{p_n}$. This means that the control polygon, regarded as a parametric curve, is itself a piecewise linear B-spline curve of order two. If we increase the order, the distance between the control polygon and the curve increases. A higher order B-spline curve tends to smooth the control polygon and at the same time mimic its shape. For example, if the control polygon is convex, so is the B-spline curve. Another property of the control polygon is that it will get closer to the curve if it is redefined by inserting knots into the curve and thereby increasing the number of vertices. If the refinement is infinite then the control polygon converges to the curve.

The *knots* of a B-spline curve describe the following properties of the curve: 1. The parameterization of the B-spline curve 2. The continuity at the joins between the adjacent polynomial segments of the B-spline curve.

The number of equal knots determines the degree of continuity. If k consecutive internal knots are equal, the curve is discontinuous. Similarly if $k - 1$ consecutive internal knots are equal, the curve is continuous but not in general differentiable. A continuously differentiable curve with a discontinuity in the second derivative can be modeled using $k - 2$ equal knots *etc.*

B-spline curves do not in general pass through the two end control points. Increasing the multiplicity of a knot reduces the continuity of the curve at

that knot. The control polygon will coincide with the curve at a knot of multiplicity $k-1$, and a knot with multiplicity k indicates C^{-1} continuity (a discontinuous curve). Repeating the knots at the end k times will force the endpoints to coincide with the control polygon. Such knot vectors and curves are known as *clamped*.

Figure 2.3 shows a clamped B-spline curve of order $k = 4$ defined by $n = 4$ control points, together with its control polygon and its knot vector made up of $n + k$ knots, distributed along the parameter range $[t_k, t_{n+1}]$.

Nurbs curves. A Nurbs (Non-Uniform Rational B-Spline) curve is a generalization of a B-spline curve,

$$c(t) = \frac{\sum_{i=1}^{n} w_i \mathbf{p}_i B_{i,k,t}(t)}{\sum_{i=1}^{n} w_i B_{i,k,t}(t)} \tag{2.7}$$

In addition to the definition of a B-spline curve, the Nurbs curve c has a sequence of weights $w_1, \ldots, w_n$. One of the advantages of Nurbs curves over B-spline curves is that they can be used to represent conic sections exactly (taking the order k to be three). A disadvantage is that Nurbs curves depend non linearly on their weights, making some calculations, like the evaluation of derivatives, more complicated and less efficient than with B-spline curves. The representation of a Nurbs curve is the same as for a B-spline except that it also includes:

w A sequence of weights $\mathbf{w} = (w_1, w_2, \ldots, w_n)$

Under the condition that weights are (strictly) positive, a Nurbs curve, like its B-spline cousin, enjoys the convex hull property.

2.1.3 Implicit curves

Implicit curves can also by denominated level set or isocontour since they correspond to the section of an implicit surface with a plane in the case of planar curves or the intersection of two or more implicit surfaces in the case of spatial curves. An implicit curve is said to be of degree $n = max(i + j)$, where n is the maximum sum of powers of all terms $a_m x^{i_m} y^{j_m}$.

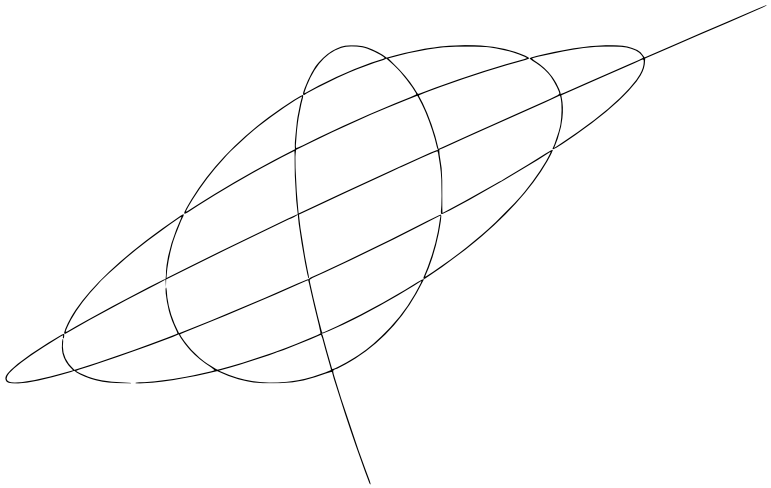


Figure 2.4: An implicit planar curve.

Planar implicit curves. A planar curve can be expressed in the implicit form as

$$f(x, y) = 0 \tag{2.8}$$

When f is linear in variables x and y, equation 2.8 represents a straight line. If $f(x, y)$ is of the second degree in x and y, 2.8 represents a class of plane curves called conic sections.

A singular point of an algebraic curve is a point where the curve has "nasty" behavior such as a cusp or a point of self-intersection. More formally, a point (x, y) on a curve $f(x, y) = 0$ is singular if the x and y partial derivatives of f are both zero at the point (x, y): $f(x, y) = \partial_x f(x, y) = \partial_y f(x, y) = 0$.

Figure 2.4 shows the curve represented by the a bi-variate polynomial of degree 8. Its visualization is carried out through a topology computation in the bounding box $[-5, 5] \times [-3, 3]$.

Spatial implicit curves. The implicit representation of a spatial curve can be expressed as an intersection curve of two or more implicit surfaces:

$$f(x, y, z) = 0 \cap g(x, y, z) = 0 \tag{2.9}$$

Figure 2.5 shows for instance a spatial curve represented by two trivariate

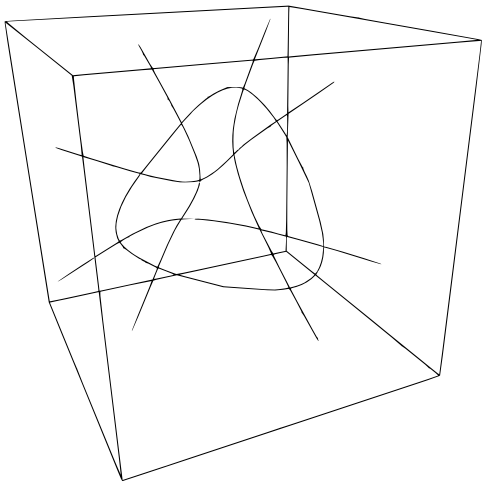


Figure 2.5: An implicit spatial curve.

polynomial equations of degree 4. Its visualization is performed through a topology computation in the bounding box $[-3,3] \times [-3,3] \times [-3,3]$.

2.2 Surfaces

Similarly to the curve case, there are many ways to represent surfaces. This section presents linear representation such as regular meshes or subdivision surfaces as well as non linear representations such as implicit or parametric representations.

2.2.1 Piecewise linear surfaces

A piecewise linear surface is a collection of vertices and polygons that defines the shape of a linear geometric object.

Regular meshes. Also referred to as surface meshes or polygonal surfaces, they usually consist of triangles, quadrilaterals or other simple convex polygons, since this simplifies rendering, but they can also contain objects made of general polygons with optional holes.

An internal representation of a piecewise linear surface contains a list of vertices, optionally a list of indexes describing which vertices are linked to

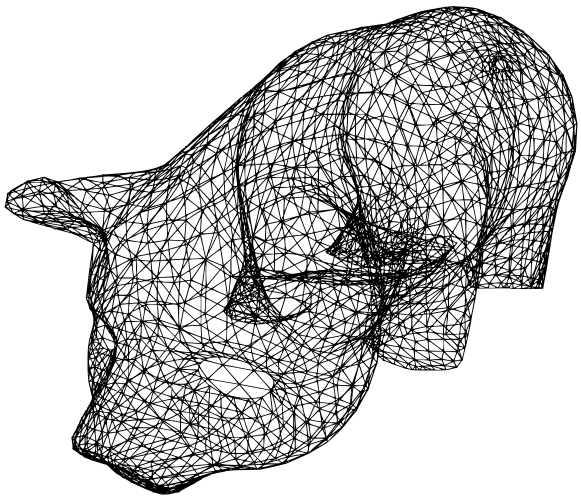


Figure 2.6: A piecewise linear surface.

form polygonal faces, a list of edges (pairs of indexes) and a list of polygons that link edges.

The choice of a data structure for faces is governed by the application: it's easier to deal with triangles than general polygons, especially in computational geometry. For optimized algorithms it is necessary to have a fast access to topological information such as edges or neighboring faces, this requires more complex structures such as the winged-edge representation, a doubly connected edge list (DCEL) or combinatorial maps.

Figure 2.6 shows a triangular piecewise linear surface made up of 1843 vertices and 3560 faces.

Subdivision surfaces. A smooth surface can be calculated from a coarse piecewise linear model as the limit of an iterative process of subdividing each polygonal face into smaller faces that better approximate the smooth surface. The resulting model is then called a subdivision surface.

The process starts with a given polygonal mesh. A refinement scheme is then applied to this mesh in order to subdivide it, creating new vertices and new faces. The positions of the new vertices in the mesh are computed based on the positions of nearby old vertices. In some refinement schemes, the positions of old vertices might also be altered (possibly based on the

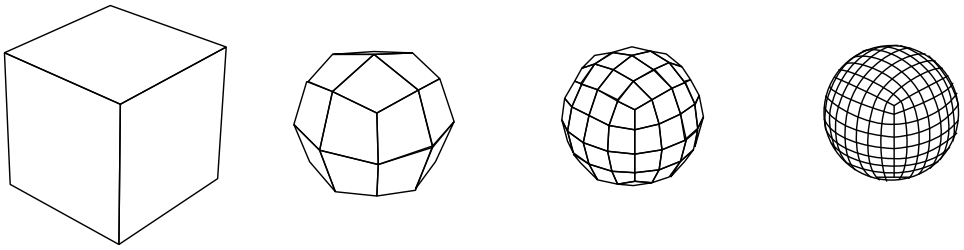


Figure 2.7: Subdivision surface using Catmull-Clark scheme.

positions of new vertices). Note that the resulting mesh can be passed through the same refinement scheme again and so on.

Subdivision surface refinement schemes can be broadly classified into two categories [23]: interpolating ones and approximating ones. Interpolating schemes are required to match the original position of vertices in the original mesh. Approximating schemes are not, they adjust these positions as needed. In general, approximating schemes have greater smoothness, but the user has less overall control of the outcome. This is analogous to spline surfaces and curves, where Bézier splines are required to interpolate certain control points, while B-Splines are not. Another division in subdivision surface is the type of polygon that they operate on, some use quadrilaterals while others operate on triangles.

In approximation schemes, the limit surfaces approximate the initial meshes and, after subdivision, newly generated control points are not on the limit surfaces. The following schemes are the most widespread:

Catmull-Clark use a generalization of bi-cubic uniform B-splines to produce their subdivision scheme. For arbitrary initial meshes, this scheme generates limit surfaces that are C2 continuous everywhere except at extraordinary vertices where they are C1 continuous (see figure 2.7).

Doo-Sabin extend Chaikin's corner-cutting method for curves to surfaces. They use the analytical expression of bi-quadratic uniform B-spline surface to generate their subdivision procedure to produce C1 limit surfaces with arbitrary topology for arbitrary initial meshes.

Remark 2.1: Subdivision surfaces were discovered simultaneously by Edwin Catmull and Jim Clark [24] and Daniel Doo and Malcom Sabin [33]. •

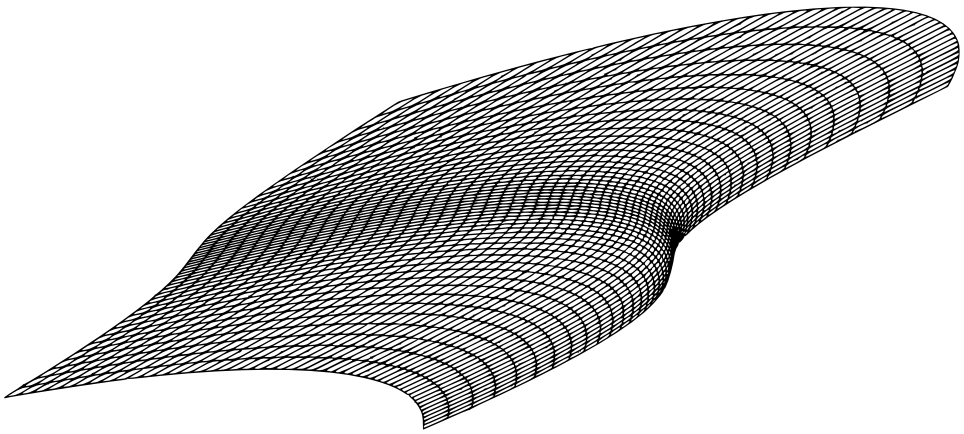


Figure 2.8: A rational surface.

Remark 2.2: Subdivision surfaces share many properties with Nurbs surfaces such as the convex-hull property. •

2.2.2 Parametric surfaces

There are many types of parametric representations for surfaces. A relevant point for organizing such geometric objects within a taxonomy is that even though the encoding of the geometry may vary from a parametric representation to another, all parametric surfaces share common properties: they are given in evaluation, *e.g.* defined by map $s : \mathbb{R}^2 \mapsto \mathbb{R}^3$.

Polynomial rational surfaces. In parametric representation, the coordinates (x, y, z) of a point of a surface patch are expressed as functions of the parameters, *e.g.* u and v, in the range $[u_{min}, u_{max}] \times [v_{min}, v_{max}]$:

$$s(u,v) = \begin{cases} x(u,v)/w(u,v) \\ y(u,v)/w(u,v) \\ z(u,v)/w(u,v) \end{cases} \tag{2.10}$$

The functions $x(u, v)$, $y(u, v)$, $z(u, v)$ and $w(u, v)$ are continuous and possess a sufficient number of partial derivatives. A parametric surface is said to be of class r, if the functions have continuous partial derivatives up to order r, inclusively.

Figure 2.8 illustrates a polynomial rational surface defined by the polynomials $x(u,v) = u$, $y(u,v) = -v$ and $z(u,v) = -u^3 - v^2 + 2v - 1$ ($w(u,v) = 1$).

B-spline surfaces. A tensor product B-spline surface is defined as

$$s(u,v) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbf{p}_{i,j} B_{i,k_1,u}(u) B_{j,k_2,v}(v) \tag{2.11}$$

with control points $\mathbf{p}_{i,j}$ and two variables (or parameters) u and v. The formula shows that a basis function of a B-spline surface is a product of two basis functions of B-spline curves. This is why a B-spline surface is called a tensor-product surface. The following is a list of the components of the representation:

d The dimension of the underlying Euclidean space

$n1$ The number of vertices with respect to the first parameter

$n2$ The number of vertices with respect to the second parameter

$k1$ The order of the B-spline surface in the first parameter

$k2$ The order of the B-spline surface in the second parameter

u The knot vector of the B-spline surface with respect to the first parameter, $\mathbf{u} = (u_1, u_2, \ldots, u_{n_1+k_1})$

v The knot vector of the B-spline surface with respect to the second parameter, $\mathbf{v} = (v_1, v_2, \ldots, v_{n_2+k_2})$

p The control points of the B-spline surface

The representation of the B-spline surface must fulfill the following requirements: 1. Both knot vectors must be non-decreasing. 2. The number of vertices must be greater than or equal to the order with respect to both parameters: $n_1 \geq k_1$ and $n_2 \geq k_2$.

The properties of the representation of a B-spline surface are similar to the properties of the representation of a B-spline curve. The control points $\mathbf{p}_{i,j}$ form a control net as shown in figure 2.9. The control net has similar properties to the control polygon of a B-spline curve, described in section 2.1.2. A B-spline surface has two knot vectors, one for each parameter. In

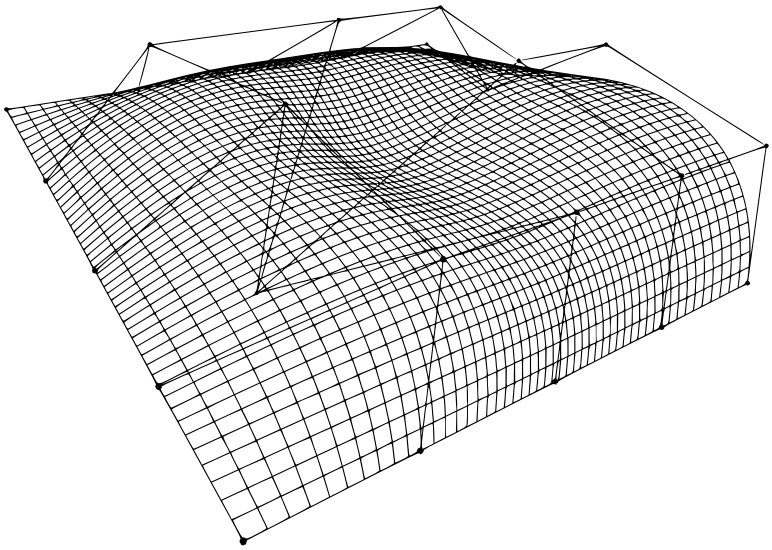


Figure 2.9: A B-spline surface.

figure 2.9 we can see isocurves, surface curves defined by fixing the value of one of the parameters.

Nurbs surfaces. A Nurbs (Non-Uniform Rational B-Spline) surface is a generalization of a B-spline surface,

$$s(u,v) = \frac{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} w_{i,j}\mathbf{p}_{i,j}B_{i,k_1,\mathbf{u}}(u)B_{j,k_2,\mathbf{v}}(v)}{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} w_{i,j}B_{i,k_1,\mathbf{u}}(u)B_{j,k_2,\mathbf{v}}(v)} \tag{2.12}$$

In addition to the data of a B-spline surface, the Nurbs surface has weights $w_{i,j}$. Nurbs surfaces can be used to exactly represent several common 'analytic' surfaces such as spheres, cylinders, tori, and cones. A disadvantage is that Nurbs surfaces depend non linearly on their weights, making some calculations, like with Nurbs curves, less efficient. The representation of a Nurbs surface is the same as for a B-spline except that it also includes

w The weights of the Nurbs surface, $w_{i,j}$, $i = 1,\ldots,n_1$, $j = 1,\ldots,n_2$, so $w = (w_{1,1}, w_{2,1},\ldots, w_{n_1,1},\ldots, w_{1,2},\ldots, w_{n_1,n_2})$.

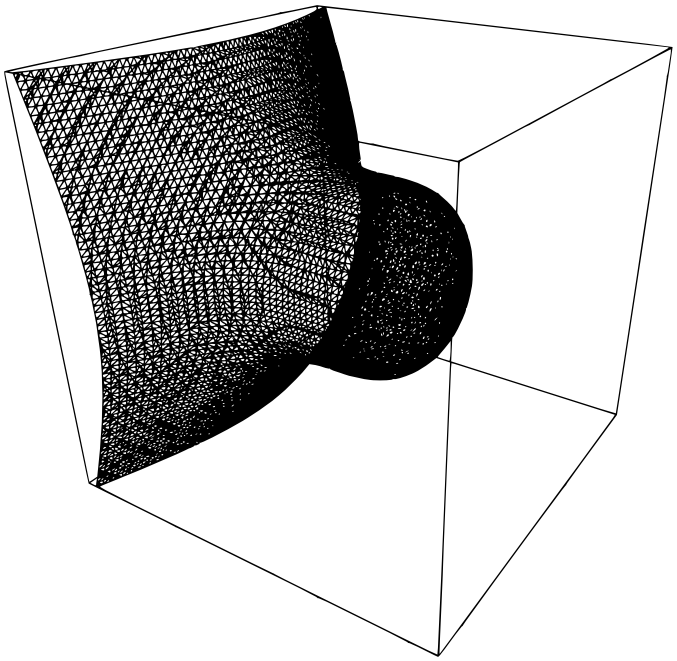


Figure 2.10: An implicit surface.

2.2.3 Implicit surfaces

An implicit surface is defined as the set of roots of a polynomial in the form:

$$f(x, y, z) = 0 \tag{2.13}$$

An implicit surface is said to be of degree $n = max(i + j + k)$, where n is the maximum sum of powers of all terms $a_m x^{i_m} y^{j_m} z^{k_m}$.

When f is linear in variables x, y and z, it represents a plane. If f is of second degree in variables x, y and z, it represents a quadric. Higher degree surfaces are respectively called cubic surfaces, quartic surfaces, quintic surfaces, sextic surfaces, septic surfaces, octic surfaces, nonic surfaces, decic surfaces, dodecic surfaces *etc.*

Figure 2.10 shows the implicit surface defined by the polynomial $x^3 + y^2 + z^2 - 1$.

2.3 Solids

The representations discussed in section 2.1 and 2.2 allow us to describe curves and surfaces in 2 or 3 dimensions. Solids are higher level geometric objects composed of curves and surfaces and present many properties, *e.g.* , they are manifold, they define an interior and an exterior.

Solid modeling addresses the representation of an object by two major distinct ways: boundary representation and constructive representation.

In constructive solid geometry (CSG), a solid is represented as a set of theoretic boolean expressions of primitive solid objects of simpler structure. Both the surface and the interior of an object are defined.

A boundary representation (B-Rep) on the other hand describes only the oriented surface of a solid as a data structure composed of vertices, edges and faces. The orientation convention permits us to decide on which side of the surface the solid's interior lies.

CSG and B-Rep representations have inherent strength and weaknesses. For instance, a CSG object is always valid in the sense that its surface is closed, orientable and encloses a volume, provided the starting primitives fulfill these properties. A B-Rep object on the other hand is easily rendered on a graphic system. As a consequence, there is a strong tendency to combine both representations to benefit from both their advantages.

2.3.1 Constructive representation

A CSG object is constructed with a set of standard primitives using regular boolean operations. These standard primitives usually are cubes, spheres, cones, cylinders and torus. Each one of these primitives may be associated to a local frame coordinate that must be related one to another with respect to a common world frame coordinate.

The main boolean operations are *union*, *intersection* and *difference*. They differ from the corresponding mathematical set theoretic operations in the sense that they are used in a way which allows to eliminate unwanted lower dimensional structures, their result is the closure of the operation on the

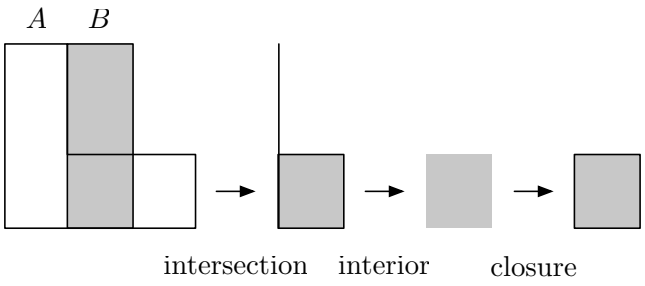


Figure 2.11: Regularized boolean intersection.

interior of the solid (we refer to [68] for more information). This results in *regularized* boolean operators.

Figure 2.11 illustrates a regularized intersection. Two parallelepipeds are first united to obtain an "L-shaped" solid, namely A, which is intersected in turn with another gray filled parallelepiped B to obtain a cube. The usual mathematical operation created a dangling edge which is removed by taking the interior. The closure finally gives the result.

The CSG representation of a solid is conveniently represented by a tree, called a CSG-tree. Leaves of this tree are primitive solids and interior nodes are either rigid motions (translations, rotations, scaling) or boolean expressions.

Figure 2.12 is the CSG-tree representing the construction of the solid operated in figure 2.11.

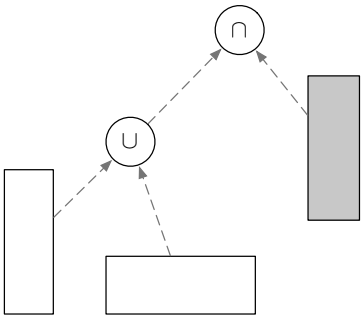


Figure 2.12: CSG-tree corresponding to the previous operation. Boolean operations are regularized ones.

2.3.2 Boundary representation

Boundary schemes are the most widely used representations for solids. In boundary representation models are composed of two parts: topology and geometry.

Briefly, the topological description specifies vertices, edges and faces *abstractly* (see remark 2.3), and indicates their incidences and adjacencies.

The geometric description specifies, for examples the equations of the surfaces of which the faces are a subset, or the euclidean coordinates of a vertex.

Remark 2.3: Any of the curves described in section 2.1 can be used in B-Rep to describe bounded edges and any of the surfaces described in section 2.2 can be used to describe bounded faces. •

The boundary representation implies to pay strict attention to preserve several properties: manifoldness and orientability.

A *manifold* surface has the property that around every one of its points, there exist a neighborhood that is homeomorphic to the plane.

Remark 2.4: The manifoldness property excludes to describe faces by self-intersecting or touching surfaces. •

A manifold surface is *orientable* if one can distinguish its interior from its exterior.

Remark 2.5: There exist a well known trick to decide whether a manifold surface is orientable or not. It consists in defining an arbitrary orientation at one point p and any closed path on the surface. If it is possible to return to p while moving along that path with an opposite orientation then the surface is not orientable, otherwise it is orientable. The Klein bottle is a famous example of non orientable manifold surface. •

These properties are crucial requirements to certify further operations on objects. For example, a boolean operation on non orientable surfaces may provide a wrong result since the operation may use the orientation property.

2.3.3 Semi-algebraic sets

Both implicit and parametric curves and surfaces actually are semi-algebraic sets that we define now.

An algebraic set is the locus of zeros of a collection of polynomials. For example, the circle is the set of zeros of $x^2 + y^2 - 1$ and the point at $(0, 0)$ is the set of zeros of x and y. The algebraic set $\{(x, 0)\} \cup \{(0, y)\}$ is the set of solutions to $xy = 0$.

A semi-algebraic set is a subset of $\mathbb{R}^n$ which is a finite boolean combination of sets of the form

$$\{f(x_1, \ldots, x_n) \geq 0 \cup g(x_1, \ldots, x_n) = 0\} \tag{2.14}$$

where f and g are polynomials in $x_1, \ldots, x_n$ over the reals.

After Requicha [92], it can be shown that sets that are bounded, regular and semi-algebraic possess all the desired properties, and therefore provide appropriate models for solids. These sets are usually called simply r-sets. Intuitively, r-sets are curved polyhedra with faces lying on algebraic surfaces. A more precise characterization follows.

A semi-algebraic half space is a set of points that satisfy an algebraic inequality

$$\{p : f(p) \leq 0\} \tag{2.15}$$

where f is a polynomial. For example, the inequality

$$ax + by + cz + d \leq 0 \tag{2.16}$$

defines a planar half space, *i.e.* the portion of 3-space which lies to one side of the plane defined by the equation

$$ax + by + cz + d = 0 \tag{2.17}$$

A semi-algebraic set is the result of a finite number of (standard, unregularized) set-theoretic operations on semi-algebraic half spaces. For example, a finite solid cylinder is the intersection of three semi-algebraic half spaces. One of these is cylindrical and the other two are planar, as shown schemat-

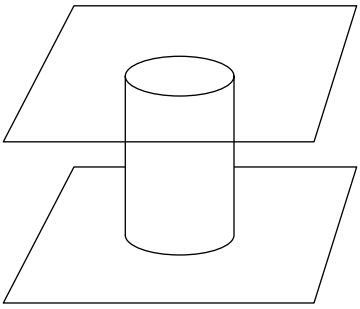


Figure 2.13: A finite cylinder is the intersection of a cylindrical set and two planar half spaces.

ically in figure 2.13 (each half space itself is unbounded, only bounded portions of the half space boundaries are shown in the figure).

Because $-f$ is also a polynomial, and $-f \leq 0$ is equivalent to $f \geq 0$, we could have defined semi-algebraic half spaces with inequalities of the form $f \geq 0$. Furthermore, the intersection of the two half spaces $f \geq 0$ and $f \leq 0$ is the set defined by the equation $f = 0$, and this is an algebraic set. Therefore algebraic sets are special cases of semi-algebraic sets.

It can also be shown that the interior, boundary and closure of a semi-algebraic set are also semi-algebraic. Therefore, a finite number of regularized boolean operations on semi-algebraic sets produces another semi-algebraic set. This implies that a set defined by boolean operations (as found in CSG) on semi-algebraic half space primitives also is semi-algebraic. Furthermore, if the primitives are r-sets, the result also is an r-set, because regularized booleans preserve boundedness, regularity and semi-algebraicity. This implies that CSG representations in the domain of r-sets are always valid.

A polynomial has a finite number of coefficients, and a semi-algebraic set is the result of a finite number of (non-regularized) boolean operations on a finite number of half spaces defined by polynomial inequalities. Therefore, a semi-algebraic set is always finitely describable.

It is also true that a bounded semi-algebraic set in 3-space is uniquely determined by its boundary, which is semi-algebraic as well.

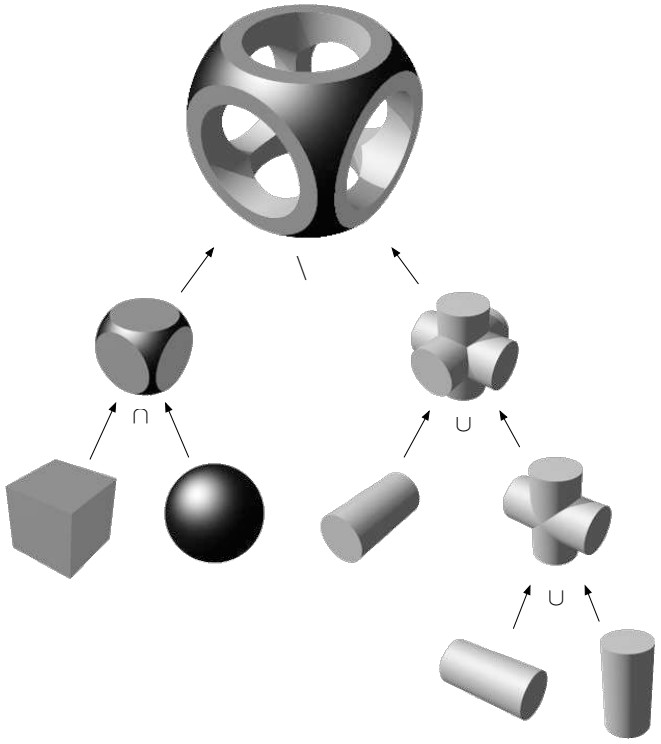


Figure 2.14: A CSG tree on semi algebraic sets.

2.4 Software

Many geometric modeling software or toolkits are available either commercially or open source. Applications usually feature a model representation of choice depending on a targeted use. We briefly overview the characteristics of some of them.

2.4.1 Applications

CAD suites. Autodesk is the leading CAD solution provider. It is divided into four industry-specific business divisions: Manufacturing Solutions (MSD), Architecture, Engineering & Construction (AEC), the Media and Entertainment Division (M&E), and Platform Solutions & Emerging Business (PSEB). Platform Solutions and Emerging Business division develops and manages Autodesk's flagship product, AutoCAD. The Manufacturing Solutions Division develops and manages Autodesk Inventor Series and Au-

toCAD Mechanical. The Architecture Engineering and Construction division develops and manages AutoCAD Architecture. The principal product offerings from the Media and Entertainment Division are Maya and 3DS Max that will be discussed later. Lets us focus on AutoCAD and Inventor.

AutoCAD is a CAD software application for 2D and 3D design and drafting. In earlier releases, AutoCAD used primitive entities such as lines, polylines, circles, arcs, and text as the foundation for more complex objects. Since the mid 90s, AutoCAD has supported custom objects through its C++ API and now includes a full set of basic solid modeling and 3D tools, but lacks some of the more advanced capabilities of solid modeling applications.

AutoCAD's native file format, DWG, and to a lesser extent, its interchange file format, DXF, have become *de facto* standards for CAD data interoperability. AutoCAD in recent years has included support for DWF, a format developed and promoted by Autodesk for publishing CAD data. In 2006, Autodesk estimated the number of active DWG files to be in excess of one billion. In the past, Autodesk has estimated the total number of DWG files in existence to be more than three billion.

Unlike AutoCAD, **Inventor** is based on the most advanced parametric modeling techniques used by products like SolidWorks and Pro/ENGINEER. Inventor accomplishes this using an approach that Autodesk calls "Functional Design".

Inventor users begin by designing parts. These parts can then be combined into assemblies or design within the context of an assembly. As a parametric modeler, it should not be confused with traditional CAD programs. It is used in design and engineering to produce and perfect new products. Whereas in non-parametric CAD programs the dimensions are geometry-driven, a parametric modeler allows the geometry to be dimension-driven. If the dimensions are altered, the geometry automatically updates based on the new dimension. This allows the designer to store their design intent within the model, whereas non-parametric modeling is more akin to a "digital drafting board". Inventor also has tools for sheetmetal part creation, welded part creation, and, starting with version 10, a rendering and animation environment called Inventor Studio based on the mental ray rendering

engine.

As an example of workflow, in order to make a simple cube, a user would first make a square sketch, then use the extrude tool to make a cube feature out of it. If a user then wanted to add a shaft coming out of the cube, he could add a sketch on the desired face, draw a circle, and then extrude that circle to create a shaft. The best aspect of this design is that all of the sketches and features can be edited later, without having to redo the entire part. This system of modeling is much more intuitive than in older modeling environments, where if you wanted to change basic dimensions, you would usually have to delete the entire file and start over. As the final part of the process, parts are then connected to make assemblies.

This method of modeling allows the creation of very large, complicated assemblies, especially since sets of parts can be put together before they are joined to the main assembly, and some projects may have many sub assemblies.

Inventor uses specific file formats for parts (.IPT), assemblies (.IAM) and drawing views (.IDW or .DWG) but the DWG file format can be also imported/exported. Autodesk has been pushing Design Web Format (.DWF) as the preferred 2D/3D data interchange and review format within the Autodesk family of products.

In the last several years Inventor has grown to include functionality contained in many of the mid-level to high level 3D modelers. Inventor uses Shape Manager as its geometric modeling kernel, which is proprietary to Autodesk and was derived from the ACIS modeling kernel.

SolidWorks is a 3D mechanical CAD program that was developed by SolidWorks Corporation - now a subsidiary of Dassault Systèmes. It uses the Parasolid geometric modeling kernel. SolidWorks was introduced in 1995 as a low-cost competitor to CAD programs such as Pro/ENGINEER and CATIA, and is currently one of the most popular products in the midrange mechanical CAD market.

SolidWorks employs a parametric, feature-based approach for creating models and assemblies. Parameters refer to constraints or conditions whose values determine the size, shape, characteristics, and behavior of the model or

assembly. Parameters can be either numeric, for example dimension values such as the diameter of a circle or the length of a line; or geometric, such as conditions like tangent, concentric, coincident, parallel, horizontal, and the like. Numeric parameters such as dimensions can easily be related to each other through equations to capture even the most complicated design intent. This approach brings advantages comparable to Autodesk Inventor.

CATIA (Computer Aided Three dimensional Interactive Application) is a multi-platform CAD/CAM/CAE commercial software suite developed by French company Dassault Systèmes and marketed world-wide by IBM. The software was originally intended for the development of Dassault's Mirage fighter jet, but became a runaway success and was subsequently adopted by numerous well known companies world-wide, such as Boeing and IBM. The software was also famously used by architect Frank Gehry in his building of the Guggenheim Museum Bilbao. CATIA is written in the C++ programming language.

Commonly referred to as a 3D Product Lifecycle Management software suite, CATIA supports multiple stages of product development. The stages range from conceptualization, through design (CAD) and manufacturing (CAM), until analysis (CAE).

CATIA provides an open development architecture through the use of interfaces, which can be used to customize or develop applications. The supporting application programming interfaces are as follows: Fortran and C programming languages for version 4 (V4), Visual Basic and C++ programming languages for version 5 (V5).

TopSolid is a 3D CAD software edited and developed by the company Missler Software. Its range of software includes a whole family: from the more general, mechanical oriented TopSolid'Design to job specific solutions: sheet metal TopSolid'Fold, wood TopSolid'Wood, toolmaking: TopSolid'Mold for mold makers and TopSolid'Progress for press tool designers. TopSolid also incorporates an integrated Computer Aided Manufacturing (CAM) line of products: Mechanical machining TopSolid'Cam, sheet metal TopSolid'PunchCut, wood TopSolid'WoodCam, wire electroerosion TopSolid'Wire. TopSolid also incorporates a 2D draft module TopSolid'Draft and a structural calculation module TopSolid'Castor.

TopSolid is, therefore, a CAD/CAM solution based on the geometric modeler ParaSolid. It is, of course, capable of reading and creating files in all available formats as well as in such formats as Catia and ParaSolid.

TopSolid is sold worldwide and has been rated as the 2nd CAD/CAM editor in France behind Dassault Systèmes (software Catia and SolidWorks). It has been rated as the 1st French CAM editor and is ranked among the Top 10 CAD/CAM editors worldwide and is the fastest growing CAD/CAM Company in 2007.

Creation suites. **Blender** is the leading open-source creation suite, with a robust feature set similar in scope and depth to other high-end 3D software such as SoftimageXSI, Cinema 4D, 3ds Max, Lightwave and Maya. Its features include advanced simulation tools such as rigid body, fluid, and soft body dynamics, modifier based modeling tools, powerful character animation tools, a node based material and compositing system and Python for embedded scripting.

It covers the geometric workflow from modeling to rendering including texturing, rigging and animation using uv-unwrapping, shading, physics and particles, compositing and much more.

The software provides different representations for geometric objects but is particularly efficient with subdivision surfaces. Its basic primitives include plane, circle, cube, sphere, cylinder and cone. With any regular mesh as a starting point, Blender can calculate a smooth subdivision surface on the fly using Catmull-Clark algorithm, allowing high resolution mesh modeling without the need to save and maintain huge amounts of data.

Blender also implements other representations such as Bézier, B-Splines and Nurbs curves and surfaces. Models using such a representation are defined by less data, they produce nice results using less memory at modeling time. Some modeling techniques such as extruding a curve along a path or computing a lofted surface (process called skinning in Blender) are only possible when using such curves. Note however that even though so called “blobs” are proposed, there is a lack of implicit surfaces. In particular, one could expect modeling physical phenoma using low degree algebraic surfaces such as quadrics.

Finally Blender integrates a node based compositor fully integrated within the rendering pipeline and an internal file system that allows one to pack multiple scenes into a single file (called a ".blend" file) which is less a structured specification of objects and relationships but closer to a direct binary dump of the program's memory space. This makes it very hard to convert a ".blend" file to another format using external tools, although dozens of import/export scripts that run inside Blender itself make it possible to inter-operate with other 3D tools, accessing the object data via its python API.

Maya is a popular, proprietary integrated node-based 3D software suite, evolved from Wavefront Explorer and Alias PowerAnimator, now owned by Autodesk. The software is released in two versions: Maya Complete (the less powerful package) and Maya Unlimited. Maya Personal Learning Edition (PLE) is available at no cost for non-commercial use, although rendered images are watermarked.

Popular models such as Nurbs, polygons and subdivision surfaces are available, but it is with Nurbs that Maya gives the best of itself.

A feature that makes Maya even more powerful is that it can connect anything to anything, *e.g.* one can use a color intensity of a shader to control the movement of a door opening and closing. To control the node based system of Maya, fully reconfigurable user interface can be scripted with MEL script code. Maya 8.5 has introduced support for the Python scripting language.

Research tools. We finally mention a solid modeling environment emerged from research in the field of CAD, which is what is most comparable (but far more experienced) to what we have achieved in this thesis. **Irit** is a solid modeling environment that allows one to design primitive based models using boolean operations as well as freeform surface's based models. Beyond its strong support for Bézier and B-spline curves and (trimmed) surfaces, it has several unique features such as strong symbolic computation, support of trivariate spline volumes, multivariate spline functions and triangular patches, as well as numerous unique applications such as surface layout decomposition, metamorphosis of curves and surfaces, and artistic line art drawings of parametric and implicit forms. A rich set of compu-

tational geometry tools for freeform curves and surfaces is offered, such as offsets, bisectors, convex hulls, diameters, kernels, and distance measures. The solid modeler is highly portable across different hardware platforms, including a whole variety of Unix machines and Windows PC.

The system is designed for simplicity and is geared toward research. As such, no graphical user interface exists or is planned in the near future. The modeling is performed using the main module/executable of the system. A textual interface (or PUI for programmable user interface) is available which provides the interaction interface. An interpreter processes the user's command and executes them. This interpreter includes general mechanisms that are common in high level programming languages such as loops, conditional sentences, and functions. In addition, features that can be found in modern languages such as operator overloading and object oriented design are extensively used. This interpreter is best employed under the Emacs editor that forks out Irit as a sub process (available both for Unix and Windows).

2.4.2 Toolkits

Open Inventor, originally IRIS Inventor, is a C++ object oriented retained mode 3D graphics API designed by SGI to provide a higher layer of programming for OpenGL. The strategy was based on the premise that people were not developing enough 3D applications with OpenGL because it was too time-consuming to do so with the low-level interface provided by OpenGL. If 3D programming were made easier, through the use of an object oriented API, then more people would create 3D applications and SGI would benefit.

OpenGL is a low level library that takes lists of simple polygons and renders them as quickly as possible. To do something more practical like "draw a house", the programmer must break down the object into a series of simple OpenGL instructions and send them into the engine for rendering. One problem is that OpenGL performance is highly sensitive to the way these instructions are sent into the system, requiring the user to know which instructions to send and in which order, and forcing them to carefully cull the data to avoid sending in objects that aren't even visible in the resulting image. For simple programs a tremendous amount of programming has to

be done just to get started.

Open Inventor was written to address this issue, and provide a common base layer to start working with. Objects could be subclassed from a number of pre-rolled shapes like cubes and polygons, and then easily modified into new shapes. The "world" to be drawn was placed in a scene graph run by OpenInventor, with the system applying occlusion culling on objects in the graph automatically. OpenInventor also included a number of controller objects and systems for applying them to the scene, making common interaction tasks easier. Finally, OpenInventor also supplied a common file format for storing "worlds", and the code to automatically save or load a world from these files. Basic 3D applications could then be written in a few hundred lines under OpenInventor, by tying together portions of the toolkit with "glue" code.

On the downside OpenInventor tended to be slower than hand-written code, as 3D tasks are notoriously difficult to make perform well without shuffling the data in the scene graph by hand. Another practical problem was that OpenInventor could only be used with its own file format, forcing developers to write converters to and from the internal system.

After many years of Inventor being solely available under proprietary licensing from TGS, it was released under an open source license in August 2000, which is available from SGI.

At approximately the same time, an API clone library called Coin3D was released, written in a clean room fashion from scratch, sharing no code with the original SGI Inventor library, but implementing the same API for compatibility reasons.

Coin3D is built on OpenGL and uses scene graph data structures to render 3D graphics in real-time. Basic import, rendering, and interaction with a 3D object can be implemented in just a few lines of code, and programmer efficiency is greatly increased compared with programming directly with OpenGL. OpenGL code and Coin3D code can co-exist in the same application, which makes gradual migration from OpenGL to Coin3D possible.

Crystal Space is primarily a Software Development Kit, a middleware for developing 3D applications. There is a strong focus on games in particular,

but Crystal Space itself is not limited to that. Notable features include strong cross-platform support, numerous utilities, and bindings for multiple languages.

Development-relevant features include basic helper classes such as *e.g.* containers, abstraction of platform-specific details, often requiring none to very little platform-specific code in client applications, a plugin system for extensibility, customizability and versatility, and even, a custom build system, that can also be used for client applications and provides conveniences such as generation of Visual C++ projects.

While the "heart" of Crystal Space are the 'engine' and 'renderer', essentially providing management of what should be rendered, and actual rendering, there are also helper plugins providing and abstracting file input/output, audio output, physics, input from joysticks, and GUIs. However, it does not provide any game-specific logic, such as entity management.

Ogre (Object-Oriented Graphics Rendering Engine) is a scene-oriented, flexible 3D rendering engine (as opposed to a game engine) written in C++ designed to make it easier and intuitive for developers to produce applications utilizing hardware-accelerated 3D graphics. The class library abstracts the details of using the underlying system libraries like Direct3D and OpenGL and provides an interface based on world objects and other high level classes.

As its name states, OGRE is "just" a rendering engine. As such, its main purpose is to provide a general solution for graphics rendering. Though it also comes with other facilities (vector and matrix classes, memory handling), they are considered supplemental. It is not an all-in-one solution in terms of game development or simulation as it doesn't provide audio or physics support, for instance.

Generally, this is thought of as the main drawback of OGRE, but it could also be seen as a feature of the engine. The choice of OGRE as a graphics engine allows developers the freedom to use whatever physics, input, audio and other libraries they want and allows the OGRE development team to focus on graphics rather than distribute their efforts amongst several systems. OGRE explicitly supports the OIS, SDL and CEGUI libraries, and includes the Cg toolkit.

Currently OGRE is published under a dual license (one being LGPL, the other one called OGRE Unrestricted License (OUL)), to make it possible to be chosen for console development as well, because most of the publishers reject using free/open-source software in that particular market.

OpenSceneGraph is an open source high performance 3D graphics toolkit, used by application developers in fields such as visual simulation, computer games, virtual reality, scientific visualization and modeling.

The toolkit is written in standard C++ using OpenGL, and runs on a variety of operating systems including Microsoft Windows, Mac OS X, Linux, IRIX, Solaris and FreeBSD.

Cgal, The Computational Geometry Algorithms Library, offers data structures and algorithms like triangulations (2D constrained triangulations and Delaunay triangulations in 2D and 3D), Voronoi diagrams (for 2D and 3D points, 2D additively weighted Voronoi diagrams, and segment Voronoi diagrams), Boolean operations on polygons and polyhedra, arrangements of curves and their applications (2D and 3D envelopes, Minkowski sums), mesh generation (2D Delaunay mesh generation and 3D surface mesh generation, skin surfaces), geometry processing (surface mesh simplification, subdivision and parameterization, as well as estimation of local differential properties, and approximation of ridges and umbilics), alpha shapes, convex hull algorithms (in 2D, 3D and dD), operations on polygons (straight skeleton and offset polygon), search structures (kd trees for nearest neighbor search, and range and segment trees), interpolation (natural neighbor interpolation and placement of streamlines), shape analysis, fitting, and distances (smallest enclosing sphere of points or spheres, smallest enclosing ellipsoid of points, principal component analysis), and kinetic data structures.

All these data structures and algorithms operate on geometric objects like points and segments, and perform geometric tests on them. These objects and predicates are regrouped in CGAL Kernels.

Finally, the Support Library offers geometric object generators and spatial sorting functions, as well as a matrix search framework and a solver for linear and quadratic programs. It further offers interfaces to third party software such as the GUI libraries Qt, Geomview, and the Boost Graph Library.

Bibliographical notes

The first section of this chapter is dedicated to the definition of curves and surfaces that will be encountered in this thesis. Some of them are inspired by [89], which contains many additional properties as well as interrogation methods, beyond the scope of the algorithms proposed in this work. Concerning B-splines, we have preferred the definition of [30] and [48] since it corresponds to the one of objects manipulated in our implementation (cf. chapter 8). The discussion of solids and their representation follows the point of view of [92] and [68]. Finally the presentation of software and toolkits comes from our former experience and information available on the Internet.

Chapter 3

Algorithmic preliminaries

To better understand the need of algorithms and algebra in the vast field of geometric modeling and especially their demand in targeted applications such as the ones found in CAD or CAM, let us start with an introductory example.

Figure 3.1 shows the modeling of a motorcycle disc brake rotor, as those that one can find in CAD, *e.g.* produced by some technical drawing software. The left part is the final drawing whereas the right part shows some preliminary steps of its design, where we can easily see that the final result has been

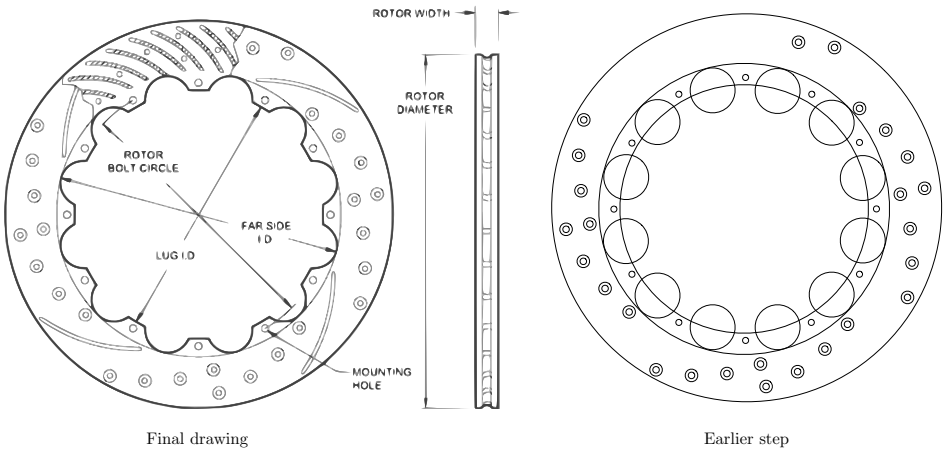


Figure 3.1: Modeling of a disc brake rotor: final drawing and earlier step.

produced by assembling pieces of simpler geometry.

The representation chosen to define these simpler geometric entities is crucial. Until recently, piecewise linear models have been used to design such mechanical pieces. Their use raises important problems. First their definition implies a huge amount of data when fine accuracy is desired. Second they represent an approximation of the shape. As shown in figure 3.2, this poses many problems when it comes to finding intersection points: cross-intersection points are approximate and tangential intersection points can even be missed. The model is therefore not appropriate to physical simulations.

Curve and surfaces with algebraic representation yield more compact yet exact models. Indeed the circles which are used to define the rotor of figure 3.1 necessitate many vertices and edges if defined by meshes, whereas parametric models such as Nurbs are more compact and implicit curves allow to define such a shape by the mean of a simple polynomial.

Since modeling such a physical entity consists in assembling several objects all together, it is not only important to be able to accurately define each one of them but one also needs interrogation tools to know what are their features, whether they intersect, whether they self-intersect and especially

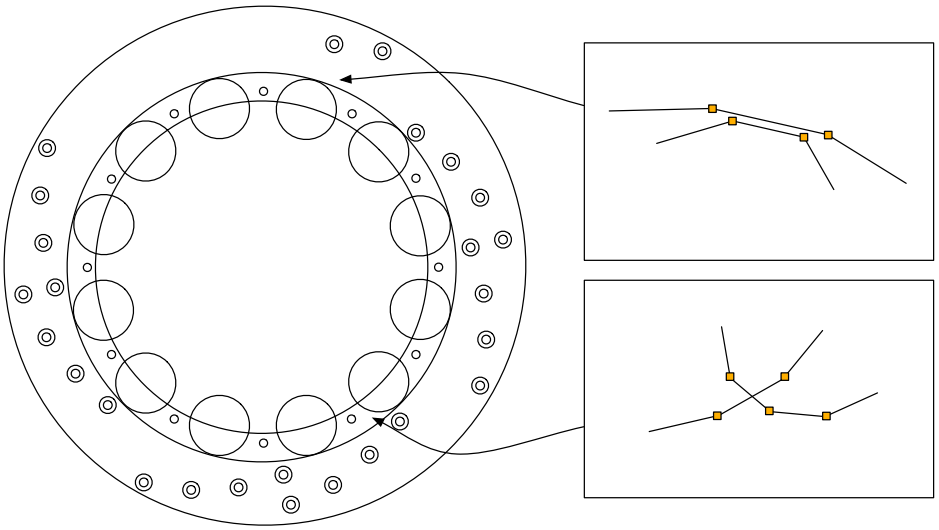


Figure 3.2: Piecewise linear models represent an approximation of the shape.

how considered all together they subdivide the ambient space.

This chapter settles a generic framework for computing with non-linear models and reminds major achievements concerning topology, intersection, self-intersection and arrangements.

3.1 Generic framework

We expose a generic framework in-between the one of computational geometry and the one of algebraic geometry. To this end, we address terminology issues to set a non ambiguous base for the following discussions. It classifies methods to compute with a collection of objects and schemes to drive the computation. Then, we introduce elementary data structures which are used to organize and manipulate the previously defined objects.

3.1.1 Terminology

As explained before, solid modeling consists in computing a combination of objects. Many methods exist in computational geometry [16] to deal with a collection of objects.

Incremental method. The most natural one is perhaps the *incremental method* which consists in processing the input to the problem, one item at a time. Algorithms implementing this method usually initiate the process by solving the problem for a small subset of the input and maintain the solution as remaining data is inserted one by one. If the order in which the data is processed is random, the incremental method is then said to be randomized. To determine the way the data is processed these incremental methods are driven by a computational scheme.

Sweep scheme. The *sweep scheme* is deeply linked with the geometric nature of the problem in hand. In 2 dimensions, numerous problems (including topology, intersection or arrangement computation) can be solved by sweeping the plane with a line. In higher dimensions, sweeping the space

with an hyperplane often reduces a d-dimensional problem into a sequence of $(d-1)$-dimensional problems much easier to deal with.

A sweep algorithm solves a two-dimensional problem by simulating a sweep of the plane with a line, agreeing on a fixed direction in the plan, say the y-axis, called the *vertical* direction. A line Δ parallel to that direction sweeps the plane when it moves continuously from left to right, from its initial position $x = -\infty$ to its final position $x = +\infty$.

Algorithms that proceed by sweeping the plane use two data structures: one structure $\mathcal{Y}$ called the state of the sweep and another $\mathcal{X}$ called the event queue. The information stored in $\mathcal{Y}$ is related to the position of the sweep line and changes when this line moves, the structure $\mathcal{Y}$ must be modified only at a finite number of discrete positions (so called events) and the maintenance of this structure yields enough information to build the solution of the original problem.

The event queue $\mathcal{X}$ stores the sequence of events yet to be processed. The sweep algorithm initializes the structure $\mathcal{Y}$ for the leftmost position $x = -\infty$ of the sweep line, and the sequence $\mathcal{X}$ with whatever events are known from the start (in increasing order of their abscissae). Each event is processed in turn, and $\mathcal{Y}$ is updated.

Example 3.1: Computing the intersection points of a set $\mathcal{S}$ of n line segments in the plane is a famous example of this kind of problem [13].

The naive solution is to test all the $n(n-1)/2$ possible pairs. The resulting algorithm would run in $O(n^2)$ time. Using the sweep scheme, one can design an algorithm with an $O((n+a)\log n)$ where a is the number of intersecting pairs.

Let's assume the line segments are in general position, that is, no pair of line segment share an endpoint, no line segment is vertical and no more than two line segments intersect at the same point.

The sweep algorithm stores in the data structure $\mathcal{Y}$ the set of segments of $\mathcal{S}$ which intersect the vertical sweep line Δ. Such segments are said to be *active* at the current position of the sweep line, ordered by the ordinates of their intersection point with Δ. This order is modified only when the line sweeps over the endpoint of a segment or over an intersection point: 1. If Δ

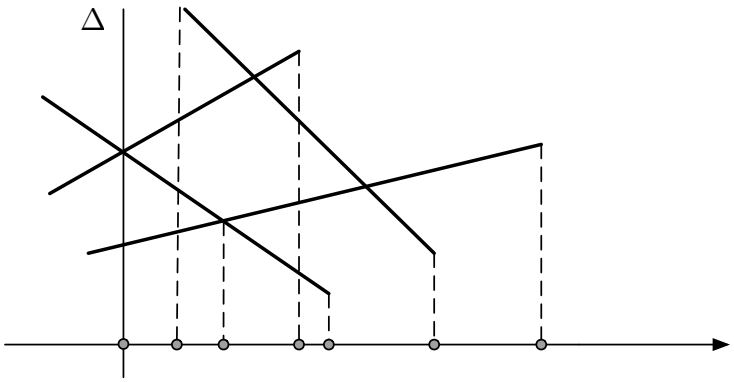


Figure 3.3: Computing the intersection of a set of line segment using a sweep scheme.

sweeps over the left endpoint of a line segment S (that is to say, the segment with the smaller abscissa), this segment is added to the structure $\mathcal{Y}$, 2. If Δ sweeps over the right endpoint of a line segment S (that is to say, the segment with the greater abscissa), this segment is removed to the structure $\mathcal{Y}$, 3. If Δ sweeps over the intersection of two line segments S and S', they switch their order in the structure $\mathcal{Y}$.

The set of events therefore includes the sweep line passing over the endpoints of the segments of S, and over the intersections. The abscissae of the endpoints are known as part of the input, and we wish to compute the abscissae of the intersection points. A prospective intersection point I is known when two active segments become consecutive in the sequence stored in $\mathcal{Y}$, the corresponding event is stored in the event queue $\mathcal{X}$. The state of the event queue is shown on figure 3.3 for a particular position of Δ.

At the beginning of the algorithm, the queue $\mathcal{X}$ stores the sequence of endpoints of the segments in S ordered by their abscissae. The data structure $\mathcal{Y}$ is empty. As long as there is an available event in the queue $\mathcal{X}$, the algorithm extracts the event with the smallest abscissa, and processes it as follows: 1. The event is associated with the left endpoint of a segment S. This segment is then inserted in $\mathcal{Y}$. If S intersects with its predecessor (resp. successor) in $\mathcal{Y}$, their intersection point is inserted into $\mathcal{X}$. 2. The event is associated with the right endpoint of a segment S. S is therefore removed from $\mathcal{Y}$. If the predecessor and successor or S in $\mathcal{Y}$ intersect in a point beyond the current position of the sweep line, this intersection point is queried

in $\mathcal{X}$ and the corresponding event is inserted if not yet present. 3. The event is associated with the intersection point of two line segments S and S'. This intersection point is computed and S and S' are exchanged in $\mathcal{Y}$. Assuming S is the predecessor of S' after the exchange, S and its predecessor are tested for intersection. If they do intersect and if the abscissa of the intersection point is greater than the current position of Δ, the corresponding event is inserted into $\mathcal{X}$. The same operation is performed for S' and its successor.■

Subdivision scheme. The *subdivision scheme* is more general and provides a hierarchical context that can be used more easily to provide levels of detail. Also, it avoids projections by enclosing the objects in boxes which are reduced by the mean of a filtering technique which ensures some configuration of the subdivision cells. It is a very convenient scheme to filter the computation using algebraic subdivision solvers.

Static or dynamic. Algorithms can also be classified depending on whether they require a preliminary knowledge of the input data or not. If an algorithm requires a global knowledge of the input, that is, it can only compute the solution of the input set in one pass, it is said *static*. On the contrary if an algorithm can maintain a solution without looking ahead at the data that remains to be inserted, it is said *semi-dynamic*. Fully *dynamic* algorithms can maintain their solution to a problem under both insertions and deletions. In fact, turning an incremental algorithm into a static or dynamic one only ends up in interacting with the adapted data structure as shown in the following subsection.

Objects. The objects that we consider in this framework are semi-algebraic sets whose representation can either be discrete, parametric or implicit. We will denote by $\mathcal{O} = \{o_1, \ldots, o_n\}$ the set of input objects of an incremental algorithm. The set of objects defined at an earlier step, *i.e.* before inserting the object o_n will be denoted $\mathcal{O}_{n-1} = \mathcal{O}_n \setminus o_n = \{o_1, \ldots, o_{n-1}\}$.

Regions. *Regions* are connected components of the input space $\mathbb{E}$, which interior does not intersect the objects of $\mathcal{O}$. They are constructed such that their boundary is a set of edges (a bounded segment of a curved object)

and possibly (depending on the dimension of the problem) a set of faces (a bounded segment of a surface) on objects of $\mathcal{O}$. In addition to to this information, a region will be associated to the set of objects involved in its representation and which determine it. Sign conditions can also be associated to a region with regard of the type of objects which determine it.

Conflict. We say that two regions *conflict* together if their interior intersects. To make the discussion easier, we may either say that an object conflicts with a region, which means that a conflict exists between regions determined by this object and another region, or that two objects conflict together, which means that a conflict exists between regions defined by these objects.

3.1.2 Data structures

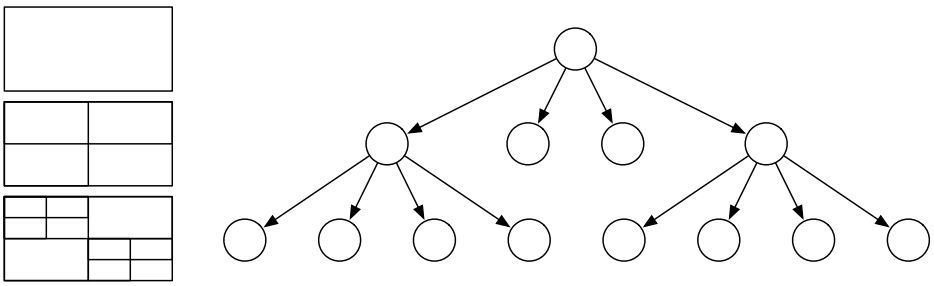
The subdivision scheme employs tree data structures to decompose the input space in smaller domains organized within a hierarchy. These structures are called space partitioning data structures.

Trees. Trees are often used in space partitioning to offer efficient structures for computing and storing a partition of the space.

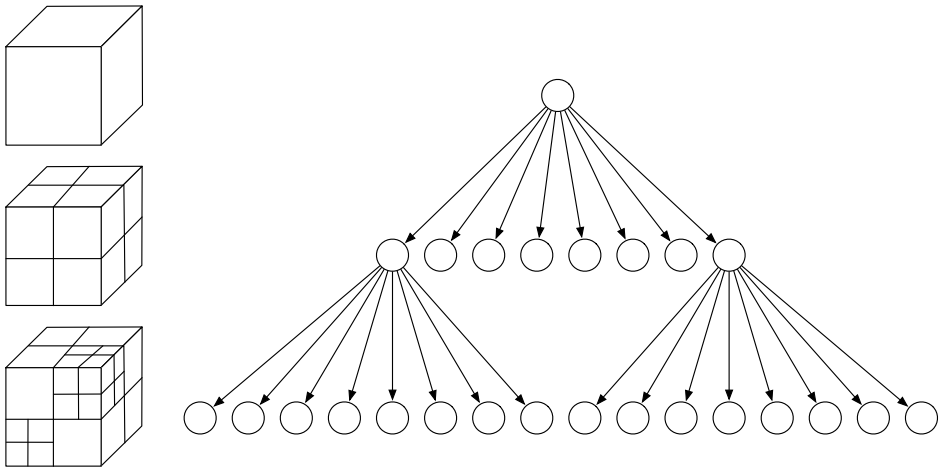
A *kd-tree* (short for k-dimensional tree) is a binary space-partitioning data structure for organizing points in a k-dimensional space. It is a useful data structure for searches involving a multidimensional search key [87].

A kd-tree uses only splitting planes that are perpendicular to one of the coordinate system axes. This differs from BSP (Binary Space Partioning) trees, in which arbitrary splitting planes can be used. In addition, in the typical definition every node of a kd-tree, from the root to the leaves, stores a point. This differs from BSP trees, in which leaves are typically the only nodes that contain points (or other geometric primitives). As a consequence, each splitting plane must go through one of the points in the kd-tree. kd-tries are a variant that store data only in leaf nodes.

A *quadtree* [95] is a hierarchical data structure based on the principle of recursive decomposition of the plane, in which each internal node has up to



(a) Recursive subdivision of a planar cell into quadrants and corresponding tree.



(b) Recursive subdivision of a spatial cell into octants and corresponding tree.

Figure 3.4: Space partitioning tree data structures.

four children. The root of the tree corresponds to the starting domain in which the subdivision has been initiated.

We do not recall the terminology of tree and refer to [2] instead, but simply remind that the depth of a tree is the size of the path (or the number of subdivisions) from the root to a leaf of the quadtree.

Figure 3.5a shows a decomposition of the input space delimited by a bounding box and the corresponding data structure.

Octrees are the three-dimensional analog of quadtrees as shown in figure 3.5b.

Graphs. Algorithms that compute a decomposition of the space organize objects and regions in graph data structures.

Sweep based algorithms often construct a *planar map* induced by a set of objects such as curves. A planar map is a bidirected graph and for every node v the cyclic ordering of the edges out of v corresponds to the ordering of the edges around v in the drawing. A crossing-free drawing of graph in the plane partitions the plane into connected regions, called faces of the drawing.

In order to be able to dynamically add or remove objects from a collection, we use an *augmented influence graph* $\mathcal{I}_a$ (see figure 3.5), which is an influence graph connected together with a conflict graph, that we describe now.

An *influence graph* is a directed, acyclic and connected graph. It possesses a single root, and its nodes correspond to the regions created by an algorithm during its execution. Therefore, a node corresponds to a region defined over the current set of objects at some point during the execution of the algorithm. The influence graph possesses two essential properties: at each step of the algorithm, a region defined over the current set of objects is associated with a leaf of the influence graph, and, the domain of influence of a region associated with a node of the influence graph is contained in the union of the domains of influence of the regions associated with the parents of that node.

In addition to the usual information stored in the influence graph, the augmented influence graph stores a conflict graph between the objects in the current set $\mathcal{O}$ and the regions stored in the nodes of the influence graph. This *conflict graph* is a system of interconnected lists: to each region stored in a node of the influence graph, corresponds a list of objects of $\mathcal{O}$ with which it conflicts, and, to each object in the current set $\mathcal{O}$ corresponds a list of regions stored in the entire influence graph that conflict with it.

As shown in figure 3.5, each node contains a region of the arrangement, connected together with a conflict graph which allows to keep track of which object is said to be the determinant or the killer of a region.

Example 3.2: An arrangement is represented by the set of leaves in the augmented influence graph, but, the latter still contains the information

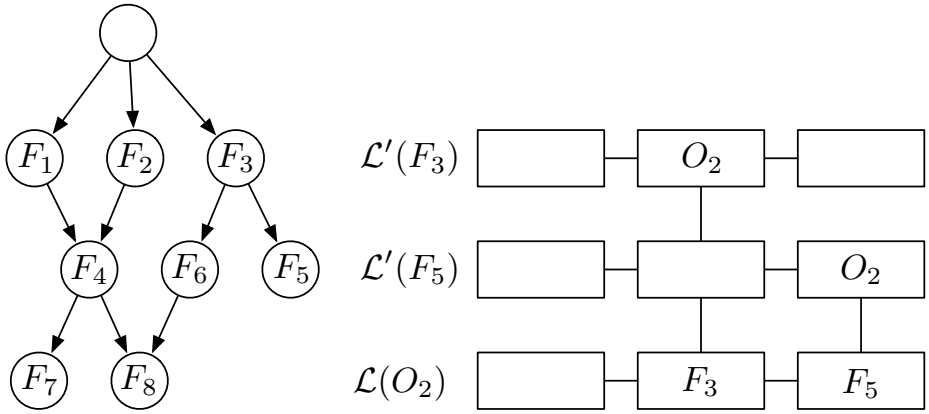


Figure 3.5: Augmented influence graph.

required to be able to remove an object from it, or to add an object to it. Indeed, leaf nodes constitute the current arrangement $\mathcal{A}_k$, where k is the number of objects in the arrangement, while other nodes (non-root and non-leaf nodes) allow to keep track of the incremental construction of the arrangement $\mathcal{A}_1 \dots \mathcal{A}_{k-1}$.

To compute an arrangement $\mathcal{A}_n$ of a collection $\mathcal{O}$ of n objects $o_1 \dots o_n$, we randomly insert the elements of the collection into the arrangement structure $\mathcal{A}$. This structure can be maintained while removing for example the object o_k, leading to the arrangement $\mathcal{A}_\Sigma$ where $\Sigma = \{o_1 \dots o_{k-1}, o_{k+1} \dots o_n\}$ is the corresponding chronological sequence.

Removing an object from an arrangement can be achieved following the general design explained in [16]: after the deletion of an object o_k, the algorithm reinserts the objects o_l of higher chronological rank $l > k$ to create new nodes and re-parent unhooked nodes in the augmented influence graph. ∎

3.2 Topology

Algebraic curves and surfaces are compact representations of shapes, which can be complex and have numerous advantages over parametric ones, such as easy determination of inside/outside of the surface. This is particularly

useful when we have to apply logical operations (union, subtraction, etc.) between two solid objects, defined implicitly. In such problems, computing the intersection of two surfaces is a critical operation, which has to be performed efficiently and accurately. Moreover, dealing with parameterized surfaces naturally leads to the computation of implicit curves. Let us mention in particular, the intersection curve of two surfaces, self-intersection curves, plane sections and ridge curves. Such problems reduce to the analysis of a curve defined by $n-1$ polynomial equations, in a space of dimension n.

One major obstacle for adopting implicit representations instead of parametric representations concerns the piecewise linear approximation of such curves or surfaces for visualization purposes. A brute force approach would be an exhaustive evaluation for approximating the zero level set, which is obviously very inefficient. A typical alternative scenario is to adopt a divide-and-conquer approach. Larger undetermined domains are broken down to smaller predictable domains in which the topological feature and eventually, the curve/surface itself can be inferred efficiently.

The problem of computing the topology of curves has been approached in different ways. A first family of methods is based on a sweeping approach. For two dimensional planar algebraic curves, such approach has been studied *e.g.* in [58] and [57]. It was later extended by Gatellier et al. in [56] to three dimensional spatial curves resulting from the intersection of two algebraic surfaces (see also [6]). These methods use a conceptual sweeping line/plane perpendicular to some projection axis, and detect the critical topological events, such as tangents to the sweeping planes and singularities. The final output of these methods is a graph of connected vertices complying to the topology of the original curve. A notable problem of aforementioned approaches is that they rely on the computation of sub-resultant sequences, which can be a bottleneck in many examples with large degree and large coefficients.

Another family of methods are the subdivision based techniques, which use a simple criterion to remove domains which do not contain the roots. A crucial problem involved here is how to efficiently and reliably deduce the root information in a given interval (or a bounding box). In these methods, instead of using monomial representation, equations are represented using

Bernstein basis [48]. Among early attempts, Sederberg [96] converted an algebraic curve into piecewise triangular Bernstein bases.

The application of subdivision methods for handling higher dimensional objects is not so well developed. In [70] a method which subdivides up to some precision level, and applies dual marching cube approach to connect points on the curve or to mesh a surface is described. The variety is covered by boxes of a given size, and the connectivity of these cells is used to deduce the piecewise linear approximation. In [3], a subdivision approach exploiting the sign variation of the coefficients in the Bernstein basis in order to certify the topology of the surface in a cell, is used for the purpose of polygonalizing an implicit algebraic surface.

Example 3.3: To illustrate the quite systematic use of the sweep scheme in topology computation, we present a method proposed by Tecourt in [56]. The topology computation of planar implicit curves is the key ingredient of many geometric problems including arrangement computation of curves and surfaces and intersection of surfaces as shown in chapters 5 and 6.

We consider a curve $\mathcal{C}$ defined as the zero locus $\mathcal{V}(f)$ of a polynomial in two variables $f(x, y) \in \mathbb{Q}[x, y]$. We can assume that f is square-free (if it is not the case, we perform a gcd-computation).

We first present from a geometric point of view the way the topology is computed. In this computation, we need to manipulate algebraic numbers.

Definition 3.1 (Critical point): A point (α, β) of $\mathcal{C} = V(f)$ is x-critical if $f(\alpha, \beta) = \partial_y f(\alpha, \beta) = 0$.

Definition 3.2 (Singular point): A point (α, β) of $\mathcal{C} = V(f)$ is singular if $f(\alpha, \beta) = \partial_y f(\alpha, \beta) = \partial_x f(\alpha, \beta) = 0$.

Definition 3.3 (Regular point): A point (α, β) of $\mathcal{C} = V(f)$ is regular if it is not critical nor singular.

Definition 3.4 (Generic position): The curve $\mathcal{C} = \mathcal{V}(f)$ is said to be in generic position if: 1. The leading coefficient of f with respect to y (polynomial in x) has no real roots. 2. For every α in $\mathbb{R}$, the number of critical points with x-coordinate α is at most 1.

Remark 3.1: In generic position, the curve has no vertical asymptote and its x-critical points have different x-coordinates. •

The geometric idea permitting to recover the topology of the curve from the computation of some particular points is as follows.

1. We compute the sequence of the subresultants of $f(x, y)$ and $\partial_y f(x, y)$ viewed as polynomials in y.

2. We compute the x-critical points $\{P_i = (\alpha_i, \beta_i)\}$.

3. We check that the curve is in generic position. If not we perform a random change of variables and restart from step 1.

4. For each critical point $P_i = (\alpha_i, \beta_i)$, we compute the number of regular points with x-coordinate α_i which are above and below P_i using Sturm sequences.

5. We compute the number of arcs above a value between two successive abscissae α_i, α_{i+1}, which is constant. It can be done for example choosing a rational x-coordinate a between α_i and α_{i+1} and computing the number of real solutions of $f(a, y) = 0$ using Sturm sequences. Then we compute numerical approximations of those different points.

6. We construct the segments connecting the points we have just computed.

The algorithm is very simple and can be observed in figure 3.6. Consider a section $x = \alpha_i$, *i.e.* all the points of the curve with abscissa α_i and the next section $x = \alpha_{i+1}$. We have chosen a rational point $a \in]\alpha_i, \alpha_{i+1}[$ and we have computed the section corresponding to $x = a$.

In the section $x = \alpha_i$, there are λ_i points above (α_i, β_i) and μ_i below.

We connect the λ_i points above (α_i, β_i) with the λ_i points of largest y-coordinate of the section $x = a$, respecting the order on the y-coordinate. We connect the μ_i points under (α_i, β_i) with the μ_i points of smaller y-coordinate of the section $x = a$, respecting the order on the y-coordinate. After that, we connect the remaining points of the section $x = a$ to the critical point (α_i, β_i). ■

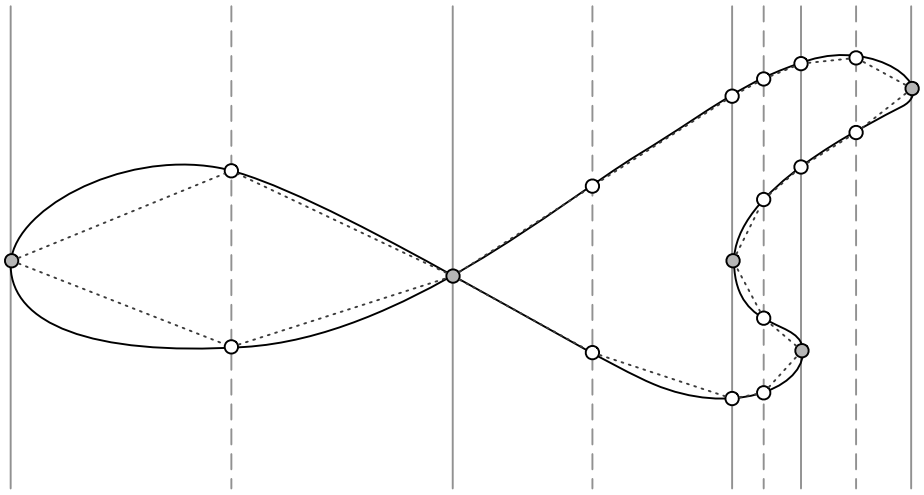


Figure 3.6: The connection algorithm provide a planar graph of point that is isotopic to the original curve.

Example 3.4: Another example is the one of using the sweep scheme in higher dimension for computing the topology of a spatial implicit curve.

For the sake of clarity, we will consider here that the curve is described as the intersection of two surfaces $P_1(x, y, z) = 0,\ P_2(x, y, z) = 0$, with $P_1, P_2 \in \mathbb{R}[x, y, z]$. We assume that the gcd of P_1 and P_2 in $\mathbb{R}[x, y, z]$ is 1, so that $\mathcal{V}(P_1, P_2) = \mathcal{C}_{\mathbb{C}}$ is of dimension 1, and all its irreducible components are of dimension 1. We are interested in describing the topology of the real part

$$\mathcal{C}_{\mathbb{R}} = \{(x, y, z) \in \mathbb{R}^3,\ P_1(x, y, z) = 0,\ P_2(x, y, z) = 0\},$$

that we will denote hereafter by $\mathcal{C}$.

Note that we do not consider examples such as surfaces intersecting tangentially along $\mathcal{C}$). Such a property can be tested by projecting into a generic direction and testing if the equation computed from the resultant of P_1, P_2 is square-free.

The general idea behind the algorithm is as follows: we use a sweeping plane in a given direction (say parallel to the (y, z) plane) to detect the critical and singular points. We also compute the positions of such points in projection on the (x, y) and (x, z) planes. Then, we connect the points of the curve of $\mathcal{C}$ on these critical planes. This yields a graph of points, connected by

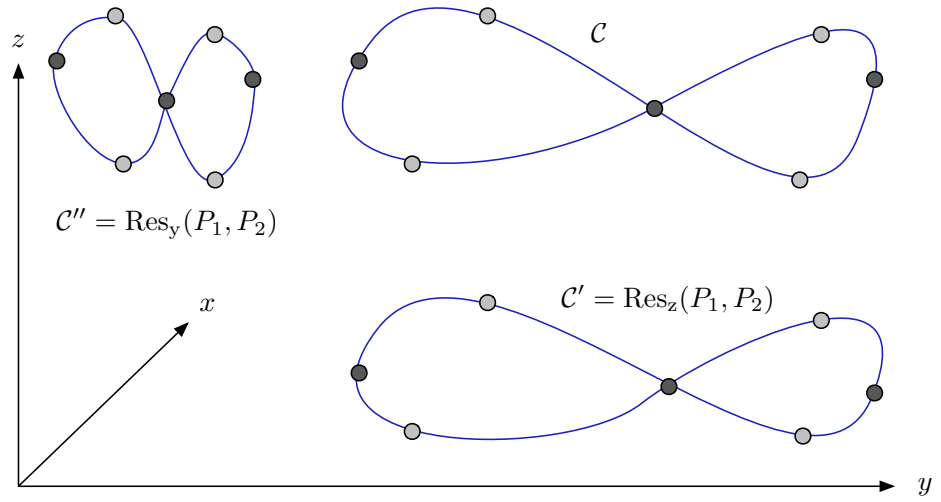


Figure 3.7: Sweeping algorithm for the topology computation of implicit spatial curve: state after step 4.

line segments, with the same topology as the curve $\mathcal{C}$. Here is the algorithm outline. We denote by $\mathcal{C}'$ (resp. $\mathcal{C}''$) the projection of the curve $\mathcal{C}$ onto the (x, y) (resp. (x, z))-plane.

1. Compute the x-critical points and their x-coordinates $\Sigma := \{\sigma_1^0, \ldots, \sigma_k^0\}$ with $\sigma_1^0 < \cdots < \sigma_k^0$.

2. Check the generic position. If the curve is not in a generic position, apply a random change of variables and restart from the first step.

3. Compute the square-free part $g(x, y)$ of $\mathcal{C}' = \mathrm{Res}_z(P_1, P_2)$.

4. Compute the square-free part $h(x, z)$ of $\mathcal{C}'' = \mathrm{Res}_y(P_1, P_2)$.

5. Compute the singular points of the curves $g(x, y) = 0$ and $h(x, z) = 0$ and insert their x-coordinate in Σ.

6. Insert new values in between the critical values of Σ: $\delta_0 < \sigma_1 < \mu_1 < \cdots < \sigma_l < \delta_1$, where $\mu_i := \frac{\sigma_i + \sigma_{i+1}}{2}$ for $i = 0, \ldots, l-1$, and δ_0, δ_1 are any value such that $]\delta_0, \delta_1[$ contains Σ. We denote by $\alpha_0 < \cdots < \alpha_m$ this new refined sequence of values.

7. Compute L_i, the set of points on $\mathcal{C}$ above α_i, for $i = 0, \ldots, m$.

8. For each $i = 0, \ldots, l-1$, connect the points L_i to those of L_{i+1}. ∎

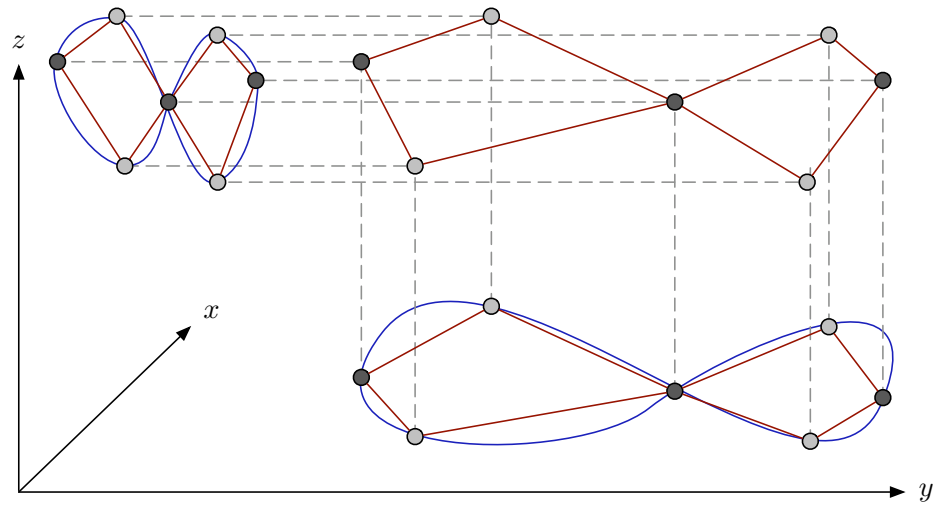


Figure 3.8: Sweeping algorithm for the topology computation of implicit spatial curve: final result.

Remark 3.2: Note that sweep algorithms are very input sensitive concerning the position of the critical points computed to get the topological graph of an object. This is caused by the nature of the sweep scheme which naturally leads to analyzing projections of the curve. Using a subdivision scheme allows to avoid the need of such strong generic position conditions since we can analyze the object in a domain of higher dimension. The subdivision scheme is the one chosen for each realization of this thesis so it will be discussed in great details in remaining chapters. •

Example 3.5: We finally mention that Mourrain and Técourt [83, 100] have proposed a meshing algorithm for algebraic surfaces that is based on sweeping a vertical plane over the surface (see figure 3.9).

We have already seen in previous examples that critical points play a crucial role in determining the topological structure of a surface. Accordingly, the algorithm uses such points to guide the sweep. It makes no smoothness or regularity assumptions about the input surface (other than those which follow from being an algebraic surface). The algorithm works for surfaces with self-intersections, fold lines, or other singularities. It however makes no guarantees about the geometric accuracy of the approximation. ■

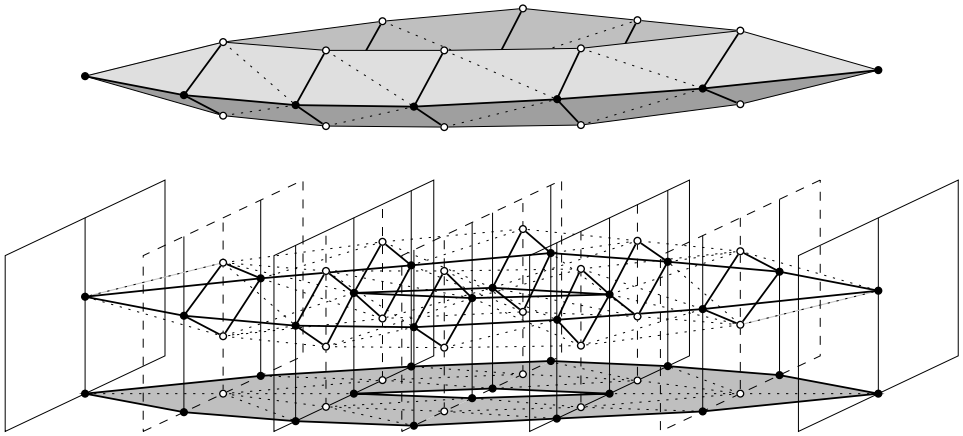


Figure 3.9: Computing the topology of an algebraic surface with a spatial sweep algorithm by connecting a sequence of vertical cuts.

3.3 Intersection

Intersection is a fundamental process in geometry, needed to build and interrogate models of complex shapes. We need intersection computation primarily to evaluate set operations on primitive volumes in creating boundary representations of complex artifacts. Such capability helps in the design representation of complex objects, in finite-element discretization, in computer animation, in feature recognition, and in simulation and control of manufacturing processes. Similarly, intersection is useful in scientific visualization to provide methods for visualizing implicitly defined objects and to contour multivariate functions representing some property of a system.

The fundamental issue in intersection problems is the efficient discovery and description of all features of the solution with high precision, *e.g.* required from the underlying geometric modeler. Reliability of intersection algorithms is a basic prerequisite for their effective use in any geometric modeling system. It is closely associated with the way the algorithm handles such features as constrictions (near singular or singular cases, for example, self-intersections), small loops, and partial surface overlap. The solutions resulting from most present techniques, implemented in practical systems, are further complicated by imprecision introduced by numerical errors in finite precision computations.

If the curve intersection problem is well handled by now (see [15] ,[89] or

[96] for a list of algorithms), the one of surface intersection is still investigated. It has strong links with curve interrogation problems, *e.g.* in [38], the ruled surface intersection is reformulated in a problem equivalent to the construction of an implicit curve in the plane.

After Patrikalakis [88], surface intersection methods can be classified in four main categories: analytic, lattice evaluation, marching, and subdivision. Most of the methods were developed in the context of polynomial surfaces.

Analytic methods rely on the derivation of a governing equation describing the intersection of two surfaces. For polynomial surfaces, the resulting equation is an algebraic curve $f(u, v) = 0$, where f is a polynomial in u, v. This equation can, for example, be obtained by substitution of the three Cartesian coordinate expressions of a rational polynomial surface $R = R(u, v)$ in the equation of an implicit algebraic surface $f(R) = 0$ (see [68]). In theory, we can handle the intersection between two rational polynomial parametric surfaces by obtaining an algebraic (implicit polynomial) representation for one of the surfaces. The relatively high degree of this algebraic representation and the subsequent substitution of the second rational polynomial surface into this high-degree equation lead to an algebraic curve of even higher degree.

Detecting the topological configuration of a high-degree algebraic curve with integer or algebraic number coefficients is a complex problem that we can approach with cylindrical algebraic decomposition. Hoffmann [68] provides an overview, and Sakkalis [94] proposed a more efficient extension. These methods, as implemented in rational arithmetic, are topologically reliable but need special attention because of large memory needs and inefficiency.

Lattice evaluation methods reduce the dimensionality of surface intersections by computing intersections of a number of isoparametric curves of one surface with the other surface. Then we connect the resulting discrete intersection points to form different solution branches. For intersections of parametric patches, the method reduces to the solution of a large number of independent systems of non-linear equations. The reduction of problem dimensionality in lattice methods involves an initial choice of grid resolution. An inappropriate choice might cause the method to miss important solution features such as small loops and isolated points that reflect near tangency

or tangency of intersecting surfaces, and thus provide incorrect connectivity.

Marching methods involve generating point sequences of an intersection curve branch by stepping from a given point on the required curve in a direction prescribed by the curve's local differential geometry [10]. However. such methods are by themselves incomplete in that they require starting points for every branch of the solution. Starting points are usually obtained using lattice and subdivision methods [11]. Marching methods also require a variable stepping size appropriate for the local length scales of the problem. Incorrect step size might lead to erroneous connectivity of solution branches or even to endless looping in the presence of closely spaced features. We can substantially improve the reliability of marching and lattice evaluation methods by determining all border, turning, and singular points of the curve.

Subdivision methods, in their most basic form, involve recursive decomposition of the problem into simpler, similar problems until we reach a level of simplicity that allows direct solution (for example, plane/plane intersection). This is followed by a phase that connects the individual solutions to form the complete solution. Initially conceived in the context of intersections of polynomial parametric surfaces, subdivision methods can be extended to the computation of algebraic/rational polynomial parametric and algebraic/algebraic surface intersections. Unlike marching methods, subdivision techniques do not require starting points (an important advantage). A disadvantage of subdivision techniques used in intersection curve evaluation is that, in actual implementations with finite subdivision steps, correct connectivity of solution branches in the vicinity of singular or near-singular points is difficult to guarantee, small loops might be missed, or extraneous loops might be present in the solution approximation. Furthermore, if we use subdivision methods for high-precision evaluation of the entire intersection set, they lead to data proliferation and are consequently slow and unattractive.

Remark 3.3: The work proposed in the following chapters addresses these quite only disadvantages of subdivision methods. First using topological degree, one can guarantee the local topology in singular configurations, also, adaptive subdivision methods coupled with efficient local techniques to get high accuracy offer the best known practical approach for the computation

of significant points. •

In Computer Aided Geometric Design (CAGD), the parameterized surfaces are used for delimiting volumes. The computation of the intersection curve between such two surfaces is thus crucial for the description of the CAGD objects. An often used method to address this problem consists in using a mesh for each surface, and then proceed to their intersection via intersection of triangles. Other methods for the intersection problem deal with global representations of the surfaces such as B-splines, however the usual CAGD procedures (offsetting, drafting, ...) do not conserve this model. In practice, so-called procedural surfaces (*i.e.* given by evaluation) are used, in CAGD systems, for representing sequences of constructions indicated by the user. Then a B-spline approximation is computed for further processing.

So, even if the intersection methods are exact, they only provide an approximation of the "real" intersection curve. Idealistically, approximations of the surfaces should not be separated from the intersection process. Let us remark that an intermediate strategy is to approximate the given surfaces by meshes of algebraic shapes more complex than the triangles. Hence the intersection locus will be more precise.

A good choice is to approximate by Bézier surface patches of small degree. Then, it is crucial to be able to efficiently intersect such two polynomial parameterized surfaces.

Example 3.6: In [25], the authors contribute to a robust solution of this problem which avoid some drawbacks as large intermediate algebraic expressions that appear in projection methods.

Let's consider the intersection curves of two biquadratic Bézier surfaces $f(u, v)$ and $g(r, s)$, both with parameter domains $[0, 1]^2$. They are assumed to be given by their parametric representations with rational coefficients (control points). More precisely, these representations have the form

$$f(u, v) = \sum_{i=0}^{2} \sum_{j=0}^{2} \mathbf{c}_{i,j} B_i(u) B_j(v) \tag{3.1}$$

with certain rational control points $\mathbf{c}_{i,j} \in \mathbb{Q}^3$ and the quadratic Bernstein

polynomials $B_j(t) = \binom{2}{i} t^i (1-t)^{2-i}$ (and similarly for the second patch $g(r,s)$).

The intersection curve is defined by the system of three non–linear equations

$$f(u,v) = g(r,s) \tag{3.2}$$

which defines the intersection as a curve (in the generic case) in $[0,1]^4$. Similarly, self intersections of one of the patches are characterized by

$$f(u,v) = f(\bar{u},\bar{v}). \tag{3.3}$$

In this case, the set of solutions contains the 2–plane $u = u^*$, $v = v^*$ as a trivial component.

While these equations could be solved by using numerical methods, it is possible to compute the intersections by using *symbolic* computations, in order to avoid rounding errors and robustness problems.

The "generic" algorithm for computing the (self–) intersection curve(s), consists of three steps:

1. Find at least one point on each component of the intersection,

2. Trace the segments of the intersection curve,

3. Collect and convert the segments into a format that is suitable for further processing (depending on the application).

Several parts of the intersection curve may exist. Some possible types are shown in figure 3.10 in the parameter domain of a Bézier surface $f(u,v)$. Points with horizontal or vertical tangent are called *turning points*, and intersections with the boundaries of the patches generate *boundary points*. Note that also isolated points (where both surfaces touch each other) may exist.

The implicitization problem – which consists in finding an implicit equation (an algebraic representation) for a given parameterized rational surface – can be addressed by using several approaches, *e.g.* using resultants or Groebner bases [26, 27, 67]. However, the implicitization is very time consuming because of the degree of the implicit equation: for a generic parameterized surface of bi-degree (n_1,n_2), the implicit equation has degree $2n_1n_2$. Also,

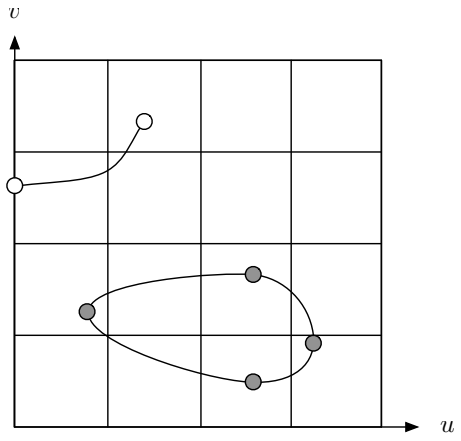


Figure 3.10: Intersection curves in one of the parameter domains. Boundary points are white ones and reversal points are gray ones.

all rational parametric curves and surfaces have an algebraic representation, but the reverse is not true. The relationship between the parametric and the algebraic representations can be very complex (problem of "phantom components"). Thus, we can try to find an algebraic approximation of a given parameterized surface for which the computation is more efficient and which contains less phantom components.

Considering a polynomial parameterized surface $f(u, v)$ with the domain $[0, 1]^2$, d a positive integer (the degree of the approximate implicit equation), $\epsilon \geq 0$ (the tolerance), following [31] (see [101] as well), the approximate implicitization problem consists in finding a non–zero polynomial $P \in \mathbb{R}[x, y, z]$ of degree d such that

$$\forall (u, v) \in [0, 1]^2, P\left(f(u, v) + \alpha(u, v)\,\mathbf{k}(u, v)\right) = 0 \tag{3.4}$$

with $|\alpha(u, v)| \leq \epsilon$ and $||\mathbf{k}(u, v)||_2 = 1$. Here, α is the error function and $\mathbf{k}$ is the direction for error measurement, *e.g.* , the unit normal direction of the surface patch.

The main question of the approximate implicitization problem is how to choose the degree. A key ingredient for this choice seems to be the topology, especially if the initial surface has self–intersections. The use of degree 4 was suggested by Tor Dokken, after several experiments he concluded that the algebraic surfaces of degree 4 provide sufficiently many degrees of freedom to

approximate most cases encountered in practice. In the case of a biquadratic surface, where the exact implicit equation has degree 8, using degree 4 seems to be a reasonable trade-off. ∎

3.4 Arrangements

Arrangements of geometric objects is a field of computational geometry which has been studied for years [1], initially with simple objects such as line segments [13], circular arcs and curves are still investigated [50, 62, 78, 110] and can be used for computing an arrangement of surfaces [84]. The current methods mainly use a sweep approach [13]. They focus on events, which are critical points for a projection direction. The events are sorted before a critical value and the order after this critical value is deduced from information at the critical points.

Example 3.7: The state of the art example is the one of computing the vertical decomposition induced by a set of line segments in the plane, that is, the first kind of arrangement of a collection of objects proposed using the Bentley-Ottman way of sweeping the plane with a line (see example 3.1).

A set $\mathcal{S}$ of segments induces a subdivision of the plane into regions, or cells, which are the connected components of $\mathbb{E}^2 \setminus \mathcal{S}$. The vertical decomposition of a set of line segments is obtained by subdividing each cell into elementary trapezoidal regions. It is a structure which depends upon the choice of a particular direction. Here let us assume this direction is that of the y-axis, which we call the vertical direction. When we want to refer to this direction, we speak of a y-decomposition. Let $\mathcal{S}$ be a set of n line segments in the plane. As previously, we suppose that the segments in $\mathcal{S}$ are in general position (meaning that no three segments have a common intersection) and that the abscissae of their endpoints are all distinct. In particular, this implies that no segment of $\mathcal{S}$ is vertical. From each point P in the plane, one can trace two vertical half-lines both upward and downward, $\Delta_1(P)$ and $\Delta_2(P)$. Let P_i $(i = 1, 2)$ be the first point of $\Delta_i(P)$ distinct from P where this half line meets a segment of $\mathcal{S}$. Should no such point exist, we make the convention that P_i is the point at infinity on the line $\Delta_i(P)$. Segments $[P, P_1]$ and $[P, P_2]$ are the walls stemming from the point P. Hence, the

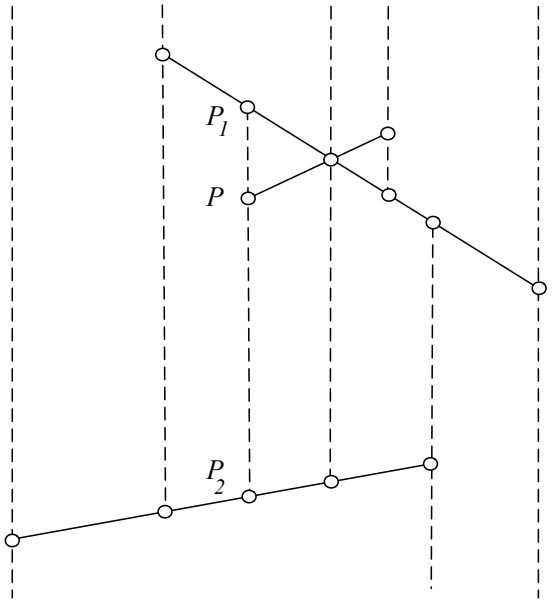


Figure 3.11: Walls in a vertical decomposition of line segments in the plane.

walls stemming from a point P are the maximal vertical segments that have P as an endpoint and whose relative interiors do not intersect segments of $\mathcal{S}$ as shown in figure 3.11.

The vertical decomposition of $\mathcal{S}$ can be described as a planar map whose vertices, edges, and regions subdivide the plane. The vertices of this map are the endpoints and intersection points of the segments of $\mathcal{S}$, and the endpoints of the walls. Each region in the map has the shape of a trapezoid, the two parallel sides of which are vertical. Some degenerate ones are triangular (with only one vertical side), or semi-infinite (bounded at top or bottom by a segment portion with two semi-infinite walls on both sides), or doubly infinite (a slab bounded by two vertical lines on either side), or even a half-plane (bounded by only one vertical line).

Each region of a vertical decomposition is thus a trapezoid, or a degenerate one, and its boundary has at most four sides, two of which are vertical. Each vertical side of a trapezoid consists of one or two walls stemming from the same point. The non vertical side of a trapezoid are respectively called ceiling and floor of the trapezoid. The floor or ceiling of a trapezoid is always included in some segment of S and its endpoints are vertices of the

vertical decomposition. Neither the floor nor the ceiling need to be edges of the vertical decomposition, they can however be made up of several edges of the planar map of the vertical decomposition. Indeed, several walls exterior to a trapezoid can butt against its floor or its ceiling, as is the case for the bottom cell in figure 3.11.

Let us directly begin with a semi-dynamic version of the algorithm, using an influence graph to maintain the current arrangement (in this case, the current vertical decomposition).

The algorithm maintains the set of regions $\mathcal{R}$ defined and without conflict over the current set of objects, together with the influence graph corresponding to the chronological sequence of objects currently inserted. The initial step processes a small set of objects. For instance, it can be the minimal number of objects needed to determine a region.

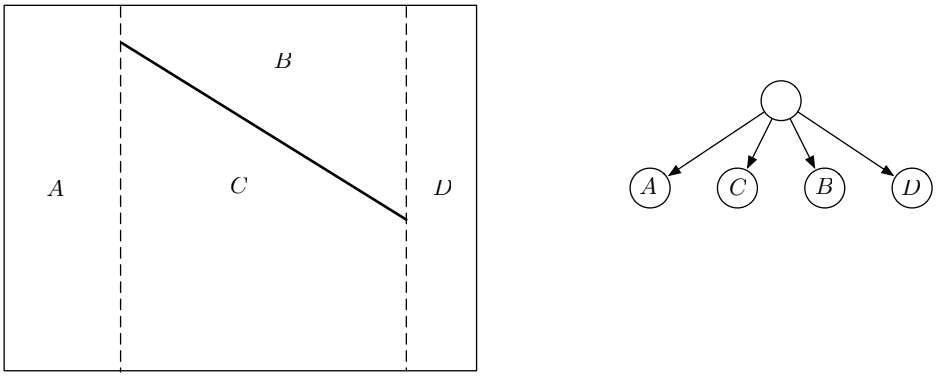
The algorithm computes the regions defined and without conflicts over the set of these initial objects. The influence graph is initialized by creating a root node, corresponding to a fictious region whose influence domain is the universe of objects in its entirety. Figure 3.13a illustrates the initial step.

In the current step, the object O is added to $\mathcal{R}$. The work can be divided into two phases: we first locate O and then update the data structures.

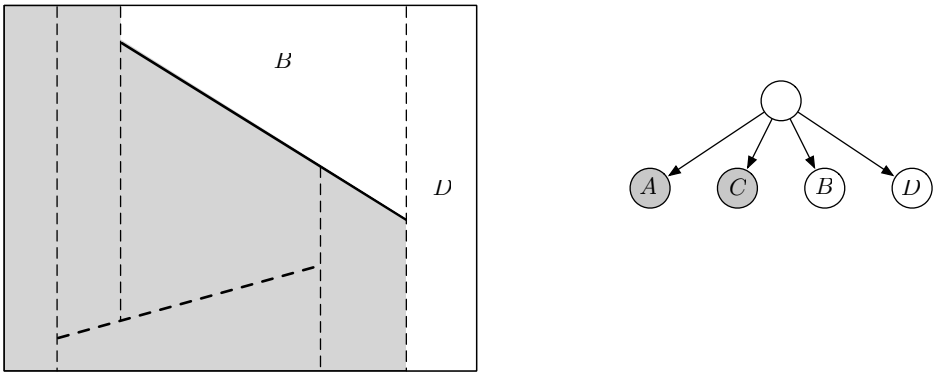
Locating: In this phase, we must find all the regions that conflict with the new object O. Starting from the root of the influence graph, we recursively visit all the nodes that conflict with O, and their children. The regions that conflict with O are said to be killed by O. Figure 3.13b illustrates the location phase.

Updating: We now have to update the data structure that represents the set of those regions defined and without conflict over the current subset of objects. We also have to update the influence graph accordingly. A leaf of the influence graph is created for each of the new regions. These are the regions created by O. Each of these leaves is linked to its parents. Figure 3.13c illustrates the updating phase.

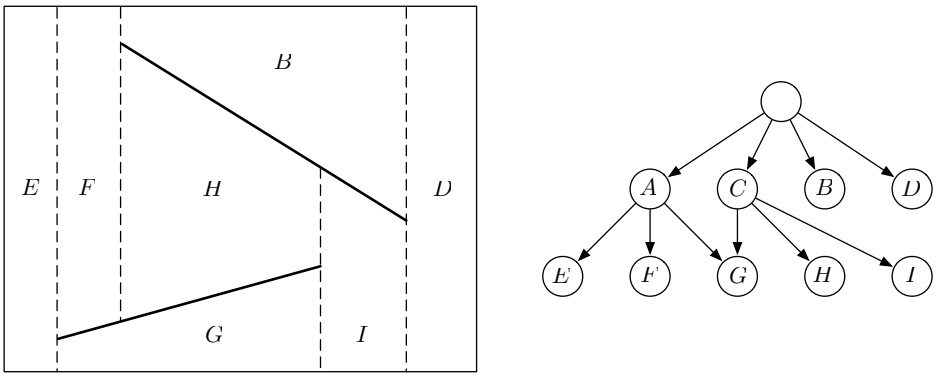
Using an influence graph naturally leads to the design of semi-dynamic algorithms, that is, no removal of any object is possible. We propose to show



(a) Initial step.



(b) Inserting a new object O: old regions are killed by O.



(c) Inserting a new object O: new regions are created by O.

Figure 3.12: Semi-dynamic decomposition of a line segment in the plane: first two steps.

how the combined use of both conflict and influence graph can yield a fully dynamic algorithm.

The general idea behind this approach is to maintain a data structure that meets the following two requirements: 1. It allows conflicts to be detected between any object and the regions defined over the current subset of objects. 2. After deleting an object, the structure is identical to what it would have been, had the deleted object never been inserted.

Such a structure is called an augmented influence graph, and can be implemented using an influence graph together with a conflict graph between the regions stored in the influence graph and the current set of objects.

We do not detail the removal of an object in great details since it requires to distinguish lots of configurations and refer to [16] instead, but we give an outlook of the method.

If $\Sigma = \{O_1, \ldots, O_n\}$ is the chronological sequence corresponding to the insertion of n objects, we denote by $\Sigma' = \{O_1, \ldots, O_{k-1}, O_{k+1}, \ldots, O_n\}$ the same sequence which corresponds to the incremental construction of the solution if the object O_k had never been inserted.

Again, in the current step, the object O_k is removed from $\mathcal{R}$. The work can be divided into two phases: we first locate O_k and then rebuild the data structures.

Locating: The phase is trivial, all the nodes that conflict with the object O_k to be deleted, or that are determined by a subset containing O_k, are visited together with their children. The algorithm identifies the destroyed nodes (which were directly or not only linked to a node corresponding to a region created by O_k) as well as unhooked nodes (the same type of nodes but with at least two parents). These regions are called critical regions.

Rebuilding: The algorithm maintains the data structures by retrieving the earlier step of the incremental construction of the vertical decomposition at rank $k-1$ and reinserts objects of Σ that have higher chronological rank $l > k$ by performing splits and joins on critical regions. ∎

More recently, sweep-line algorithms for computing an arrangement of arbitrary algebraic curves have emerged [34], making use of resultants to com-

pute roots when the sweep-line encounters an event. In [14] the authors present another context for computing an arrangement of a set of curves defined on a continuous two-dimensional parametric surface, while sweeping the parameter space.

When using sweep methods, events are treated when the sweep line encounters points of interest where a projection on a line becomes critical, reducing the dimension of the problem but increasing its computational difficulty (for instance by computing resultants and by lifting points in the case of implicit curves). Moreover the projection step onto a subspace of smaller dimension is systematically followed by a lifting operation to come back to the initial space. Most of the existing approaches rely on exact geometric computation models. When dealing with segments, this is not really an obstacle, but for general semi-algebraic objects, these operations are delicate from a numerical point of view since we are working at critical values. They require the manipulation of algebraic numbers, and their complexity could be a problem with large input polynomials.

Example 3.8: Already Bentley and Ottmann observed that the sweep-line algorithm can be used to handle arbitrary x-monotone curves (or x-monotone segments of arbitrary planar curves).

This example, shows how the authors in [50] adapt the general sweep scheme proposed by Bentley and Ottmann to handle algebraic curves in an arrangement process.

Two implicit assumptions are made by the "classical" algorithm: a pair of segments can intersect at most once, and two segments swap their relative position when they intersect. These assumptions do not necessarily hold for general curves, but one can easily remedy the situation:

- Instead of checking whether two curves intersect, we check whether they have an intersection point to the right of the current event point p_e. If there are several intersection points lying to the right of p_e, it is sufficient, at the current event, to consider only the leftmost one. However, if all intersection points are available, we can insert them all into the $\mathcal{X}$-structure.

- When we deal with an intersection event of two curves, we have to

consider the *multiplicity* of the intersection point. If the multiplicity is odd, the two curves swap their relative vertical positions and we proceed as in the case of line segments. If, however, the multiplicity is even, the two curves maintain their initial positions and no new adjacencies are created in the $\mathcal{Y}$-structure.

Suppose we wish to insert a planar curve C_i into an existing arrangement $\mathcal{A}_{i-1}$ of the curves $C_1, \ldots, C_{i-1}$. The insertion procedure of the first curve into an empty arrangement is trivial, so we will assume that $i > 1$ and $\mathcal{A}_{i-1}$ represents the arrangement of a non-empty set of curves. We will further assume that C_i is (weakly) x-monotone — if this is not the case, we will subdivide it into several x-monotone segments and insert each segment separately.

To insert an x-monotone curve C_i we execute the following procedure:

1. Locate C_i's left endpoint (and in case C_i is a vertical segment we start from its bottom endpoint) in $\mathcal{A}_{i-1}$ and act according to the type of the arrangement feature containing this endpoint:
   (a) If the endpoint lies on an existing vertex, we have to update the data associated with this vertex.
   (b) If it lies on an edge, we have to split this edge, introducing a new arrangement vertex associated with the endpoint.
   (c) If the endpoint is contained in the interior of a face, we construct a new vertex associated with the endpoint within the face.
2. Traverse the *zone* of C_i, the set of arrangement faces in $\mathcal{A}_{i-1}$ that C_i crosses (see figure 3.13 for an illustration). Each time we discover an intersection along C_i with one of the existing arrangement elements, we create a new arrangement vertex and cut C_i into two subcurves at this point. We also have to split the edges and faces of $\mathcal{A}_{i-1}$ that C_i crosses. We continue the process with the right subcurve of C_i until reaching C_i's right endpoint.
3. In case C_i's right endpoint lies on an existing vertex, update the data associated with this vertex. Otherwise we add a new arrangement vertex representing this endpoint (in case the endpoint lies on an existing edge, we will also have to split this edge). We take special care of the

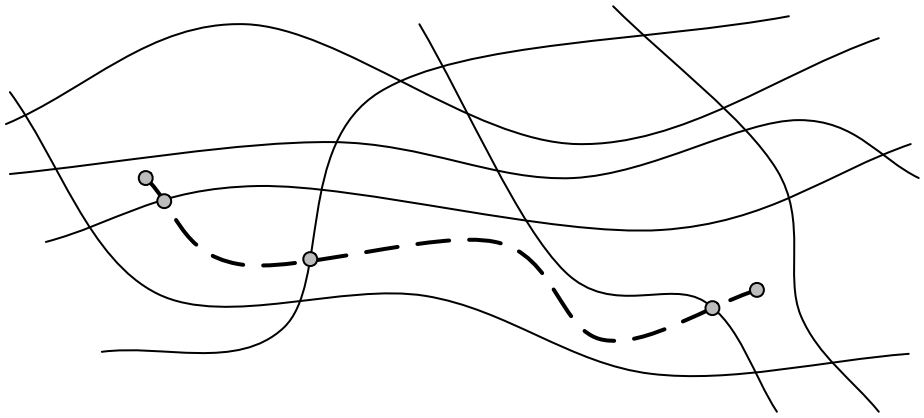


Figure 3.13: The insertion process of a new curve into an existing arrangement. Newly introduced vertices are marked.

case where C_i lies entirely within a face of $\mathcal{A}_{i-1}$ — that is, both its endpoints lie in the same face and it crosses no existing arrangement edge — as we have to initiate a new hole in the relevant face in this case. ∎

The use of subdivision methods has recently emerged such as in [17] where interval arithmetic is used to classify cells in the subdivision process. Subdivision methods are also very efficient for isolating the roots of polynomial equations, which appear in geometric problems [43, 55, 80, 97]. They have also been extended for the approximation of one or two dimensional objects [3, 70, 75]. Finally, we mention an attempt at computing elements of an arrangement of implicit curves using interval arithmetic in a subdivision process [66].

Back to our rotor design example, figure 3.14 shows an arrangement of circles defining the rotor.

Vertices of the arrangement are intersection points between the objects. Some extra vertices can appear when used to build the arrangement such as critical points for a direction. Regions are defined by a set of vertices and a set of connecting edges which are bounded curve segments. Their orientation defines the interior of the structure. This structure, sometimes called doubly connected edge list or Dcel for short allows a fast traversal of

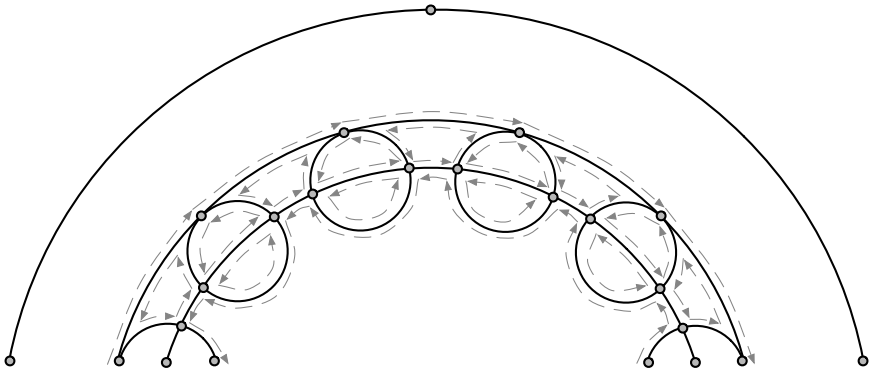


Figure 3.14: An arrangement of circles defining the disc brake.

the arrangement structure.

Bibliographical notes

The book "Algorithmic geometry" [16] serves as a reference for the classification of methods used in the field of the same name. It outlines the sweep schemes in many applications and details associated data structures together with the complexity of their query. The illustration of the sweep scheme comes from the work of Jean-Pierre Técourt [100]. The discussion of intersection methods comes from our previous considerations of the subject with Stéphane Chau *e.g.* in [25]. The illustration of the arrangement computation of curves using sweep scheme is inspired by [50].

Chapter 4

A generic arrangement algorithm

Chapter 2 shows that geometric modeling aims at combining curves together in order to represent solids either constructively or by their boundary. It is therefore crucial to know how a collection of curves decompose the ambient space into a set of connected regions. These regions should be maintainable under boolean operations such as intersection, union or difference, and should be representable by their boundary.

An arrangement allows to solve such problems. It is a high level algorithm making use of both topology, intersection and self-intersection algorithms to decompose a given collection of objects in an input space into a set of vertices, curved edges and faces describing regions that can be intersected, united or subtracted.

The first contribution of this thesis is a generic approach to arrangement computation using a subdivision scheme.

The method is generic in several respects. First, it is independent of the dimension, even though it will be specialized in dimension 2 for the computation of an arrangement of curves in the plane (see chapter 5) and in dimension 3 for the computation of an arrangement surfaces in the space (see chapter 6). Note that the same method has been successfully applied in 4-dimensional space parameter of two parametric surfaces in an intersection context. Second, it is heterogeneous and specialized for various representations such as implicit, parametric or piecewise-linear. Finally, it can be

applied either dynamically to maintain an arrangement structure under the insertion of new objects or the deletion of existing objects, or statically, considering a collection of objects and computing the result in a single pass. This genericity is made possible by the flexibility of the subdivision approach.

The use of a subdivision scheme, as opposed to a sweep scheme, allows to avoid costly projections at critical points as well as subsequent numerical errors, by enclosing these critical parts in a region in which the configuration can be deduced from information on its boundary.

Whatever the dimension, the type of the objects or the computation method, the algorithm has a very good numerical behavior. First, some computations that may require approximate tools are made only once to ensure the consistency of the method. Second, the use of segmentation structures brings necessary conditions for the conflict of regions that allow to filter the use of algebraic solvers and reduce the algorithm complexity.

We describe the dynamic version of the algorithm but its static counterpart can easily be derived from it. Indeed, each time an object is processed, the same computation can be achieved considering a set of objects instead. As a consequence, only the first section is common to the both versions whereas the following ones only make sense in a dynamic context. The insertion operation of the algorithm contains the parts which are not related to the type of input objects, assuming that type related functions will be found in a specialization of it. Inserting an object o_k into a structure built for a set $\mathcal{O}_{k-1}$ can be handled in four phases:

- **Computing regions.** In this phase, regions are computed from the topology of o_k independently of other objects of $\mathcal{O}_{k-1}$.

- **Segmenting the boundary.** In this phase, computed regions are equipped with additional data structures to help the introduction in the current arrangement $\mathcal{A}_k$.

- **Locating conflicts.** In this phase, newly computed regions are checked for conflict with regions defining the current arrangement.

- **Updating regions.** In this phase, conflicts are dealt with, possibly leading to new regions which are inserted in the data structure.

4.1 Computing regions

To match the generic framework proposed in chapter 3, we consider the object $\mathcal{O}$ together with a domain $\mathcal{D}_0$ (generally its bounding box). In a dynamic context, this section explains how to compute the set of regions $\mathcal{F}$ defined by $\mathcal{O}$ in $\mathcal{D}_0$.

Remark 4.1: In a static context, consider that $\mathcal{O}$ is a set of objects and $\mathcal{D}_0$ a domain enclosing the objects (or at least a part of each one of them) in which the arrangement computation has to be handled. •

The subdivision process decomposes the initial domain into sub-domains in such a way that the structure (or the topology) of the objects inside these sub-domains is uniquely determined from information computed on their boundary. For that purpose, we need to check the existence and unicity of some characteristic points inside these domains. The method exploits, as a main ingredient, solvers which isolate the real roots of polynomial equations defining the objects. They enclose distinct solutions into boxes which are disjoint one from the other. This is the only requirement that we ask to these external solvers.

Regions defined by an object within a given domain $\mathcal{D}_0$ cannot be computed directly from its representation. To be able to compute a set of regions, from an object in a given domain, we have to ensure that we are in a configuration where we are able to deduce its topology from information on the border of the bounding domain.

When an object is not in one of these configurations, it is subdivided into smaller parts. This process is iterated until one of these configurations is detected. An object in such a configuration is said to be *regular*. The subdivision is then driven by a *regularity test*.

There is a strong link between determining whether a cell has been subdivided enough and computing the regions defined by the object within the cell. These two type-related functionalities being dependent one from another, they are left for the specialization of an algorithm computing an arrangement of objects of type t. There is however a generic procedure to get the set of regions defined by an object in a given domain.

To each object, we associate a hierarchical structure (a 2^d-tree where d is the dimension of the input objects, *e.g.* , a quadtree when $\mathcal{D}_0$ is planar, an octree when $\mathcal{D}_0$ is spatial *etc*) used to keep track of the subdivision process which allows to deduce regions. The root of the tree stores the domain $\mathcal{D}_0$. A list of cells is initialized with the root node. While this list is not empty, we check its first element for regularity: if $\mathcal{O}$ is deemed regular in the cell, regions are computed from its topology within the cell and stored in the corresponding node of the tree. Else, the cell is subdivided into 2^d children which are appended to the list of cells to be checked for regularity and the tree is updated accordingly.

Once all cells have been processed, the leaves of the tree contain sub-regions whose union constitutes the regions defined by $\mathcal{O}$. To compute this union, these regions are merged traversing the tree from its leaves to its root in a process called *fusion*. The subdivision algorithm, summarized in algorithm 4.1 ends up with the root node representing the input domain $\mathcal{D}_0$ containing the regions determined by $\mathcal{O}$.

Algorithm 4.1: A generic subdivision algorithm.

Input: an object $\mathcal{O}$ and a box $\mathcal{D}_0 \subset \mathbb{R}^n$.

Output: a list of regions defined by $\mathcal{O}$.

Create a tree $\mathcal{Q}$ and set its root to $\mathcal{D}_0$;

Create a list of cells $\mathcal{L}$ and initialize it with $[\mathcal{D}_0]$;

while $\mathcal{L} \neq \emptyset$ **do**
    $c =$ `pop(`$\mathcal{L}$`)` ;
    if `regular(`$\mathcal{O}, c$`)` **then**
        $\mathcal{Q} \leftarrow$ `topology(`$\mathcal{O}, c$`)` ;
    else
        $\mathcal{L} \leftarrow$ `subdivide(`$\mathcal{O}, c$`)` ;
    end
end
return `fusion(`$\mathcal{Q}$`)` ;

The following operations remain to be clarified:

regularity: the specific operation which checks if regions can be computed from an object within a cell of the subdivision, *i.e.* if the object is regular in the cell.

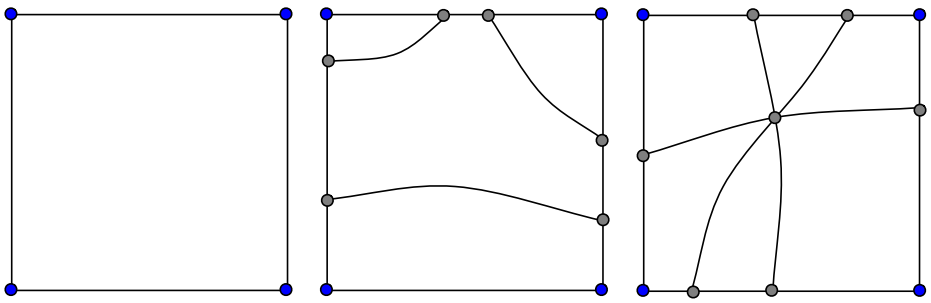


Figure 4.1: Regular cells in the plane.

subdivide: the generic operation which subdivides a cell applying dynamic programming to save computation effort.

topology: the specific operation which computes regions in a regular cell.

fusion: the generic operation which merges regions stored in each node of the tree.

4.1.1 Regularity

The regularity test allows to determine if regions can be computed in a cell from information on its boundary. It is a representation dependent operation and therefore provided in one specialization of this generic algorithm.

Example 4.1: To illustrate how the regularity test behaves, let us consider the configurations, shown in figure 4.1, in which the object is said to be regular, in a specialization of this generic algorithm to the case of curves.

The first case is the easiest configuration in which, the region is the whole cell. The regularity test just ensures that $\mathcal{O}$ does not intersects the cell.

The second case is a cell in which the curve has no x-critical (or no y-critical) point. In this case, the object is deemed regular and the topology of the curve inside the cell is uniquely determined from its intersection with the boundary. The connecting algorithm used to get the curve segments from points on the border of the cell will be described *e.g.* in section 5.1. The set of the regions defined by the curve is obtained from its topology as shown later.

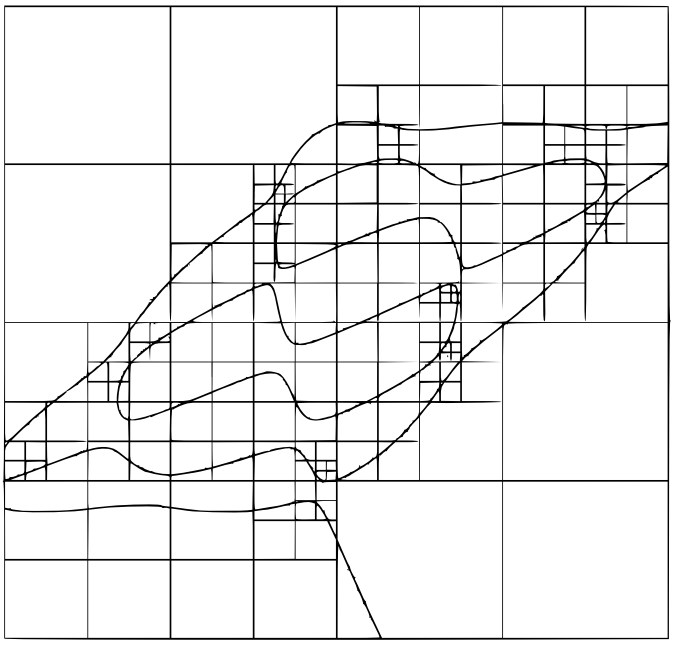


Figure 4.2: Subdivision result in the plane.

In the last case, all the branches of the object are intersecting at a unique (singular) point of the cell, in a star-shaped configuration. This case implies to be able to compute the number of branches stemming out from a self-intersection point so that the regularity test ensures that no other curve segment intersects the cell. Again, see *e.g.* section 5.1 for an algorithm to test such a configuration and connect singular regions.

Figure 4.2 shows the result of the subdivision driven by a simple regularity criterion which accepts empty cells or cells containing at most one monotonic curve segment in both directions. ∎

Testing an object in a cell for regularity involves many interrogation on the curves such as computing their critical points or their intersection with a cell of subdivision. These operations will be supplied in a specialization of the generic subdivision arrangement algorithm using only the subdivision solvers defined in chapter 1.

Remark 4.2: This strategy of subdivision can be enriched to deal with some degenerated cases such as intersection points of more than 2 curves in

a static context. It can also be optimized by requiring a limited number of active objects or of branches of these objects in the cell. If for instance, we require at most one branch per cell, the region computation will be simplified but the depth of the subdivision might increase, depending of the geometric configuration. •

Remark 4.3: Even though the regularity criteria are evoked in the case of curves, the method naturally extends to higher dimension (3 or more), as sketched in chapter 6. •

4.1.2 Subdivision

When a cell is not deemed regular, it is subdivided into smaller parts, generally more likely to be regular. When we subdivide the cell, we compute the intersection of the active objects in the cell (one object in a dynamic configuration, several objects in a static configuration) with the new boundary and update the geometric information attached to the cell such as points of interest on the border or inside the cell.

Example 4.2: As an example, a planar subdivision cell is defined by its coordinates (minimums and maximums in each direction), a list of intersection points of the objects that it contains with its border, a list of critical points (*e.g.* critical points or singularities of implicit curves) computed for the objects that it contains and a list of intersection points between the objects that it contains. When a cell is subdivided, this information is inherited to its children, simply by locating the corresponding vertices in child cells. ■

The subdivision reducing the size of a cell, already computed points will be found in child cells and can therefore be distributed during the subdivision. We call this process vertical inheritance, since points of interest are distributed from a parent, to its children, in a tree hierarchy of cells. This inheritance is processed by locating the points in the children.

Another way to save computation effort is to consider adjacency relationships between child cells. Indeed, once a cell has been checked for regularity, generated points on the border of the cells can be inherited to its neighborhood. This process takes place inside one level of the hierarchical tree, from

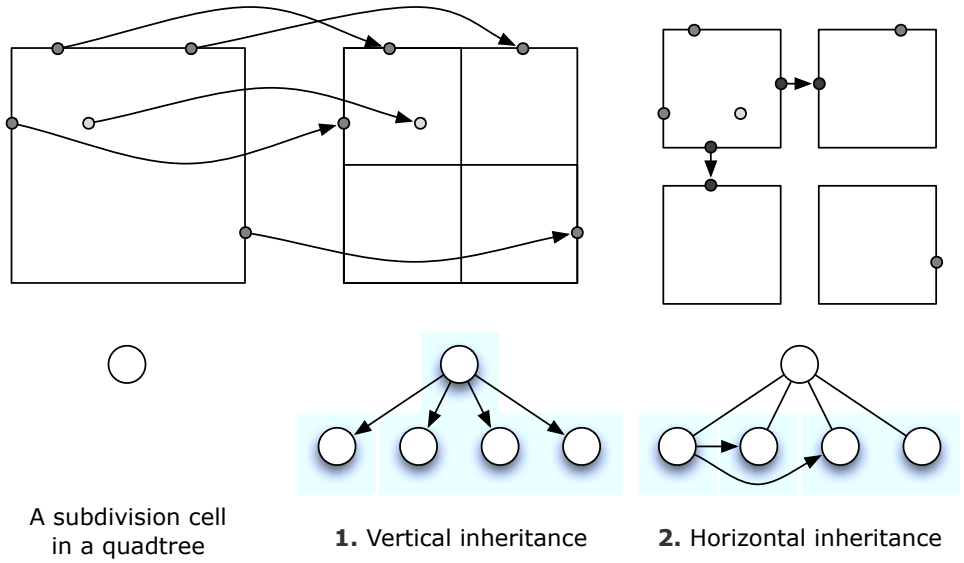


Figure 4.3: Inheritance of information in the plane.

a sibling to another, it is therefore called horizontal inheritance.

Remark 4.4: Things behave the same way when subdividing in higher space, except additional levels of inheritance are to be considered. In three dimensions for example, points of interest inside the box and points computed on the edges of the boxes are inherited following as well as intersection curves on each facet of the box. •

Remark 4.5: Beyond the performance profit, this inheritance ensures the information shared by different cells to be consistent, especially if solvers used to compute the intersections of an object with a cell are approximate, as it may be the case when using subdivision solvers. •

4.1.3 Topology

The construction of regions is based on a connection algorithm, which general idea is illustrated in figure 4.4 for the case of curves. Regions are constructed while turning around the border of the cell and connecting intersection points of the curve with the border of the cell together in loops. The orientation is a paramount aspect chosen so that a face lies on the left of an edge, turning around the cell is then performed in counterclockwise

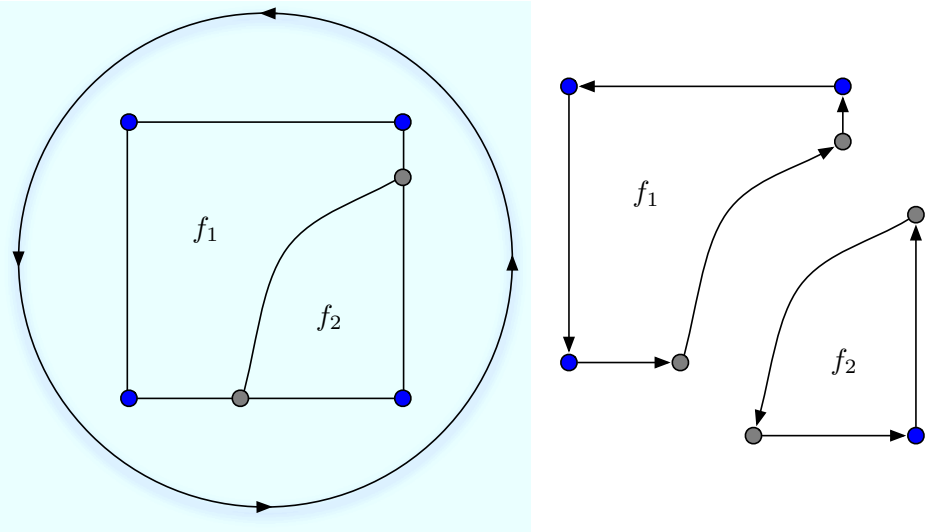


Figure 4.4: Generic scheme to compute a region from information on the border of a subdivision cell.

order.

It is even more easy in the case of a star-shaped curve, since each intersection point of the object and the boundary of the cell is connected together with the singular point lying in the cell. This requires to check whether the number of branches stemming out from a singular point strictly corresponds to the number of intersection of the curve with the border of the cell. This test is provided in a specialization of the algorithm (*e.g.* 5.1).

4.1.4 Fusion

At the end of the subdivision process, a tree contains regions computed in regular cells in its leave nodes, internal nodes keep track of the subdivision structure. To get the set of regions defined by one object, or by a set of objects, omitting the subdivision process which locally ensures a correct topology in regular cells, these small local regions have to be merged to get global regions. To do so, we traverse the tree from its leaves to its root, merging the regions inside a level of the tree and across levels, in a process called *fusion*.

To take care of being consistent regarding adjacency relationships children

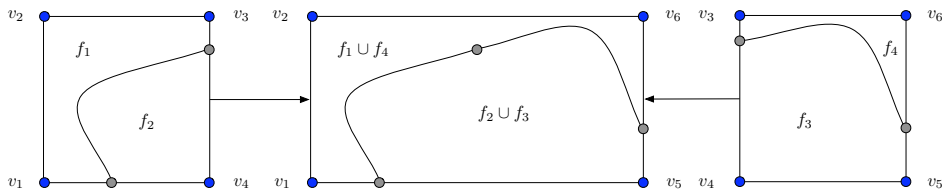


Figure 4.5: Fusion of regions.

nodes of an internal node are merged in the order illustrated in figure 4.6: the two cells located in top nodes, the two cells located in bottom nodes and the resulting top cell together with the resulting bottom cell. Regions determined in the corresponding cells are merged together, resulting in a new set of regions which are the unions of adjacent regions as shown in figure 4.5.

This algorithm brings local regions determined in regular cells associated to leave nodes, up to the root, computing their union if they are adjacent across the levels of the tree. The root node finally contains regions determined by the object or the set of objects for which the subdivision process has been initiated.

Once these regions have been computed, they are inserted in the augmented influence graph, the data structure used to represent the decomposition of the space (see definition in section 3.1.2).

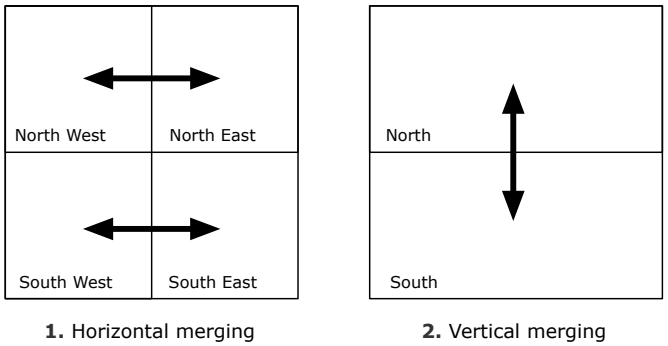


Figure 4.6: Merging schema.

4.2 Segmenting the boundary of a region

The previous subdivision scheme assures the transition from the representation of an object to its geometry, producing regions, either considering objects one by one (in a static context) or all together (in a dynamic context).

This section focuses on defining in great details the structure of a region and associated data structures used to facilitate the manipulation of regions.

A region is traditionally defined by a set of elements of incremental dimensions regarding the one of the input space: vertices, edges, faces and so on with adjacency relationships. When computing an arrangement of algebraic curves for example, regions are faces which edges are curved segment keeping their continuous representation, the vertices being some markers that help to somehow provide a discretization of these continuous objects.

The set of vertices of an arrangement is mainly defined by intersection operations on the input objects. In some cases, stronger conditions on the edges can be assumed such as a monotony requirement. In this case, considering critical points for both directions as vertices will turn the edges into monotone ones. Also, some objects may have a representation which can induce degenerate points such as isolated or self-intersection points, usually referred to as singularities, that, if considered as vertices, will provide a better accuracy in the description of regions and a certified topology.

This data structure can be enriched to enhance further operations in the case of a dynamic arrangement algorithm which maintains its solution under insertion or removal of objects, for which conflict detection is a paramount operation which needs to be performed efficiently.

Finding all the regions that conflict with a new object is a frequent operation in the arrangement computation so we need to pay special attention to its complexity. Indeed, a naive tree traversal of the augmented influence graph $\mathcal{I}_a$ where each region (stored in a node) is checked for intersection with the regions (stored in newly created nodes) to be inserted would lead to a $O(p^2)$ complexity, where p is the number of nodes in $\mathcal{I}_a$. To efficiently find out the set of regions of $\mathcal{I}_a$ which conflict with regions created by o_k we build a structure called the region segmentation.

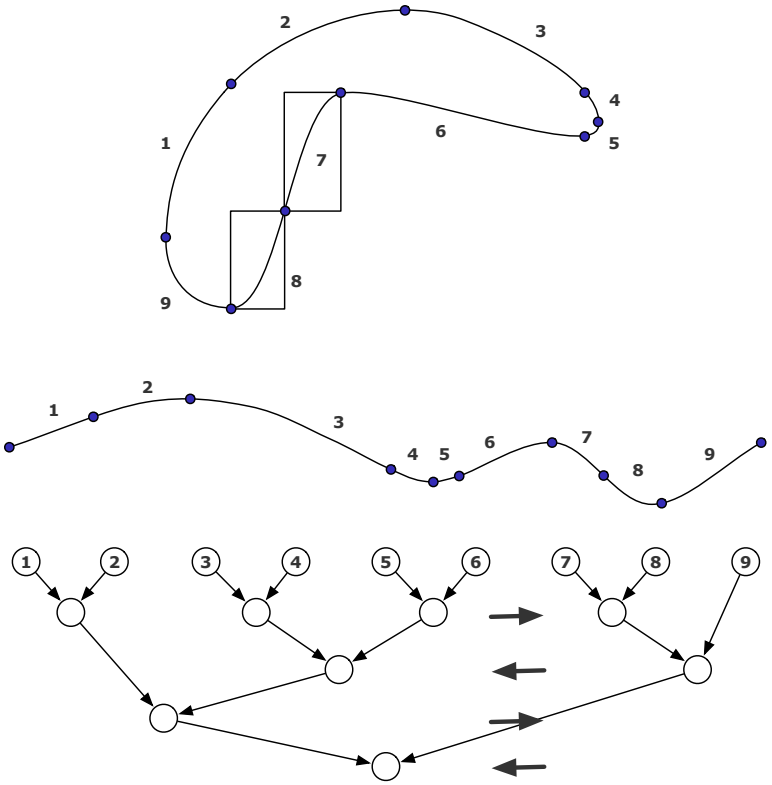


Figure 4.7: Building the region segmentation.

A *region segmentation* is a balanced tree data structure in which each node has two children (internal nodes), or none (leaves). Leaf nodes contain the edges defining the boundary of the region as well as their bounding boxes, and internal nodes are associated to the union of the boxes of their children. This way, the root of the tree defining the segmentation is associated to a *domain of influence*, a subset of the bounding box from which the region has been computed, containing the whole region.

For more efficiency when querying the tree, we propose a simple procedure to build a balanced tree from the list of edges defining a region. As shown in figure 4.7, to build the segmentation, the list of edges is "flattened", ordered and traversed to build the tree. First, a node is created for each edge and the bounding box of the edge is associated to it, yielding a list of nodes. Second, this list is traversed to build the tree. Since we want the tree to be balanced, the traversal is performed from left to right at step k, then from right to left at step $k + 1$. Each time a couple of nodes is found in

the list, we create a node containing the union of their bounding boxes and re-parent the couple of nodes to it, until no more nodes exist in the list. This structure leads to an efficient algorithm to check whether two regions conflict together and helps dealing with them.

4.3 Locating conflicts

Once regions have been processed to build the segmentation, it is very easy to find out whether two regions are intersecting, whatever the type of objects defining their edges.

To get the list of conflicting zones, we simply compare the segmentations of the regions stored in the nodes of the augmented influence graph with the segmentation of the region currently inserted.

Algorithm 4.2: Querying region segmentations for conflict.

Input: two segmentation nodes n_1 and n_2

Output: a list $\mathcal{L}$ of conflict zones

if *!*`intersects(`n_1*,* n_2`)` **then**
  return $\mathcal{L}$;
end
if `isLeaf(`n_1`)` *and* `isLeaf(`n_2`)` **then**
  $\mathcal{L} \ll$ `pair(`n_1*,* n_2`)` ;
else if `isLeaf(`n_1`)` *and* *!*`isLeaf(`n_2`)` **then**
  $\mathcal{L} \ll$ `query(`n_1*, left(*n_2*)*`)` ;
  $\mathcal{L} \ll$ `query(`n_1*, right(*n_2*)*`)` ;
else if *!*`isLeaf(`n_1`)` *and* `isLeaf(`n_2`)` **then**
  $\mathcal{L} \ll$ `query(` *left(*n_1*)*, n_2`)` ;
  $\mathcal{L} \ll$ `query(`*right(*n_1*)*, n_2`)` ;
else
  $\mathcal{L} \ll$ `query(`*left(*n_1*), left(*n_2*)*`)` ;
  $\mathcal{L} \ll$ `query(`*left(*n_1*), right(*n_2*)*`)` ;
  $\mathcal{L} \ll$ `query(`*right(*n_1*), left(*n_2*)*`)` ;
  $\mathcal{L} \ll$ `query(`*right(*n_1*), right(*n_2*)*`)` ;
return $\mathcal{L}$;

To do so, we “intersect” the respective segmentations associated to two regions, that is, beginning with the roots, we check the nodes and recursively

proceed to their children as long as their associated boxes do intersect. If two bounding boxes associated to leave nodes (containing the edges) do intersect, the algorithm 4.2, provides a list of conflict zones, in which actual intersection points can be computed using representation specific procedures.

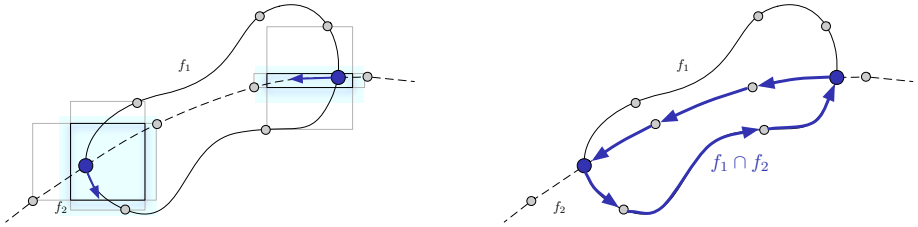
If the resulting list is not empty, the intersection of the curve segments are computed using type specific intersection methods according to the type of the objects defining the edges in question. If several intersection points are found, we refine the boundary segmentations in order to have at most one intersection per box.

If the resulting list of pairs of intersecting boxes is empty and if there is no inclusion of a region in the other, there is no conflict and the region can be inserted into $\mathcal{I}_a$. The inclusion test simply consists in locating one point of the region to be inserted. If the latter falls inside the other region, and since we already know there is no intersection of edges, we can conclude that the inserted region lies inside the other one. It is then inserted in the graph as a child node of the one representing the other region. To locate a point in a region, we only have to count the number of intersections of a half-line stemming out from the point in an arbitrary direction. The point is inside the region if this number of intersections is odd, the point is outside the region otherwise. The intersection test can be carried out in a reduced complexity using the region segmentation. Indeed, we only test the half-line for intersection with an edge of the region if the latter intersects its bounding box from the segmentation.

This query on the respective segmentations of two regions provides an efficient test to check regions for intersection and does not require any extra algebraic computation.

4.4 Updating regions

This section addresses the problem of resolving conflicts between regions. This resolution consists in dividing conflicting regions in a set of regions which union covers the conflicting regions. This set of regions is constituted of the intersection of conflicting regions and of the difference of this intersection with original regions.



(a) Computing navigation information. (b) Walk-about border of regions.

Figure 4.8: A generic boolean operation.

When two nodes of the region segmentations do intersect, a conflict is detected, yielding a conflict zone (or box). The intersection of the conflicting regions is computed in three phases. First, intersection points between regions are computed in the conflict box. Second, navigation information are computed for each intersection point, as in [111] (see figure 4.9a). Finally, a walkabout from an intersection point to another along the edges of the intersecting regions is performed to construct the intersection (figure 4.9b).

Computing navigation information for each intersection point consists in examining intersections of the edges with the conflict box, to know which edge to follow from an intersection point to another during the walkabout. To do so, we compute intersection points of each edge of each region (which are oriented counter-clockwise for their outside border and clockwise for their inside border if holes exist) with the bounding box.

When computing the intersection of regions, the walkabout proceeds along the edge whose intersection point with the box is to the left of the other, considering the orientation of edges. This can be easily obtained by sorting intersection points on the boundary of the conflict box, see figure 4.9a and 4.9b.

The resulting intersection regions are segmented and then inserted in the augmented influence graph as a child node of the regions from which they have been computed.

Chapter 5

Specialization for curves

In chapter 4, we have presented a generic algorithm. Generic means that its overall behavior will always be the same, *assuming* some functionalities. The algorithm therefore provides an abstraction of representation related functions. A generic algorithm can never be used out of the box, without providing these specific functionalities, it has to be *specialized*.

To deal with the global problem, the generic arrangement algorithm produces easier local problems. This chapter shows how these local problems are chosen by providing a specific regularity test which drives the generic subdivision process and how they are dealt with by providing a specific topology computation to compute local regions in these regular cells.

We emphasize that the output topology and arrangement are guaranteed to be correct. Although, in section 5.1, we focus on the implicit case which requires special attention, the specialization is also provided for parametric curves in section 5.2 or piecewise linear curves in section 5.4 without much additional work and no theoretical difficulties.

For each type of curve, we first provide a regularity test, that is, we want to determine whether the topology of a curve $\mathcal{C}$ can be computed inside an arbitrary rectangular domain $\mathcal{D}$ provided by the generic arrangement algorithm. The method isolates singular points from regular parts and deals with them independently. In the case of an implicit curve, the topology near singular points is guaranteed through topological degree computation. In

either case the topology inside regions is recovered from information on the boundary of a cell of the subdivision.

For each kind of domain, we also provide a *connection algorithm* that computes a piecewise approximation of the curve inside simple domains of that type.

In order to ensure the generic arrangement algorithm to be able to reconstruct the global topology in $\mathcal{D}_0$ from which $\mathcal{D}$ is originating, we have to ensure that the approximations on the $\mathcal{D}_i$ agree on the boundaries. Our connections algorithms have this property at no extra cost.

Finally, we provide intersection techniques needed to compute boolean operations on resulting regions in the framework of our dynamic arrangement algorithm.

5.1 Implicit curves

This section shows how the topology computation of an implicit curve is used in order to get an arrangement of a set of such objects.

The implicit curves we manipulate are defined by squarefree polynomials in $\mathbb{Q}[x, y]$. For $f \in \mathbb{Q}[x, y]$, $\mathcal{Z}(f) = \{(x, y) \in \mathbb{R}^2 | f(x, y) = 0\}$ will denote its zero set. But when we deal with a single curve, we will just refer to it as $\mathcal{C}$ and to its equation as f. The rectangular domain in which we carry out all our computations is denoted by $\mathcal{D} := [a, b] \times [c, d] \subset \mathbb{R}^2$.

We use enveloping techniques, which allows us to compute with fixed precision numbers. To analyze the curve $\mathcal{C}$ defined by the polynomial $f \in \mathbb{Q}[x, y]$ on a domain $\mathcal{D} = I \times J$, we convert f to the Bernstein basis on the domain $\mathcal{D}$ using exact arithmetic:

$$f(x, y) = \sum_{i,j} \gamma_{i,j} \, B^i_{d_x}(x; I) \, B^j_{d_y}(y; J) \tag{5.1}$$

we round up and down to the nearest machine precision number:

$$\underline{\gamma_{i,j}} \leq \gamma_{i,j} \leq \overline{\gamma_{i,j}} \tag{5.2}$$

so that, on $\mathcal{D}$ we have:

$$\underline{f}(x,y) \leq f(x,y) \leq \overline{f}(x,y) \tag{5.3}$$

The set of *singular points* of $\mathcal{C}$ is denoted $\mathcal{S} := \{(x,y) \in \mathbb{R}^2 | f(x,y) = \partial_x f(x,y) = \partial_y f(x,y) = 0\}$.

The set of *critical* or *extremal points* of f is denoted $\mathcal{Z}_e(f) := \{(x,y) \in \mathbb{R}^2 | \partial_x f(x,y) = \partial_y f(x,y) = 0\}$.

We recall that a tangent to the curve $\mathcal{C}$ is a line, which intersects $\mathcal{C}$ with multiplicity ≥ 2. In particular, any line through a singular point of $\mathcal{C}$ is tangent to $\mathcal{C}$.

For a subset $S \subset \mathbb{R}^2$, we denote by $\overset{\circ}{S}$ its *interior*, by $\overline{S}$ its *closure*, and by ∂S its *boundary*. We call *domain* any closed set $\mathcal{D}$ such that $\overline{\overset{\circ}{\mathcal{D}}} = \mathcal{D}$ and $\mathcal{D}$ is simply connected. We call *region* any open set R which is a connected component of the complement of an algebraic curve.

We call *branch* (relative to a domain $\mathcal{D}$), any smooth closed segment whose endpoints are on $\partial\mathcal{D}$.

We call *half branch* at a point $p \in \overset{\circ}{\mathcal{D}}$ or half branch originating from $p \in \overset{\circ}{\mathcal{D}}$, any smooth closed segment which has one endpoint on $\partial\mathcal{D}$ and whose other endpoint is p.

We call *loop*, any smooth closed curve which does not intersect $\partial\mathcal{D}$.

We distinguish three different types of simple domains: x-regular domains, y-regular domains and simply singular domains.

Definition 5.1: A domain $\mathcal{D}$ is *x-regular* (resp. *y-regular*) for $\mathcal{C}$ if $\mathcal{C}$ is smooth in $\mathcal{D}$ and it has no vertical (resp. horizontal) tangents. This is algebraically formulated as the following condition: $\mathcal{Z}(f, \partial_y f) \cap \mathcal{D} = \emptyset$ (resp. $\mathcal{Z}(f, \partial_x f) \cap \mathcal{D} = \emptyset$).

We might equivalently say that the curve $\mathcal{C}$ is x-regular (resp. y-regular) in $\mathcal{D}$ instead of saying that $\mathcal{D}$ is x-regular (resp. y-regular) for $\mathcal{C}$.

Remark 5.1: Pay attention to the fact that x-regularity is a condition on

the partial derivative along y. It ensures that the orthogonal projection on the x-axis is locally surjective. The same remark applies to y regularity. •

Finally we say for short that a curve is *regular* in $\mathcal{D}$, or equivalently that $\mathcal{D}$ is *regular* for $\mathcal{C}$ if $\mathcal{C}$ is x-regular *or* y-regular in $\mathcal{D}$.

Definition 5.2: A domain $\mathcal{D}$ is *simply singular* for $\mathcal{C}$ if $\mathcal{S} \cap \mathcal{D} = \{p\}$ and if the number n of half branches of $\mathcal{C}$ at the singular point p is equal to $\sharp(\partial\mathcal{D} \cap \mathcal{C})$, the number of points of $\mathcal{C}$ on the boundary of $\mathcal{D}$.

5.1.1 Regularity

The regularity test ensures the generic algorithm that regions can be computed in a cell of subdivision from the topology of the object(s) inside the cell. We distinguish the case of non singular (simply regular) domains and the one of singular domains.

5.1.1.1 Regular domains

We are going to show that if $\mathcal{C}$ is x-regular in $\mathcal{D}$, then its topology can be deduced from its intersection with the boundary $\partial\mathcal{D}$. By symmetry the same applies when $\mathcal{C}$ is y-regular.

We only require $\partial\mathcal{D} \cap \mathcal{C}$ to be 0-dimensional. This is a very mild requirement that can be easily taken care of when choosing a partition of the initial domain.

Remark 5.2: This is well defined because we required that $\partial_y f$ does not vanish at any point of $\mathcal{C}$ in $\mathcal{D}$. •

Definition 5.3: For a point $p \in \mathcal{C} \cap \partial\mathcal{D}$, and for a sufficiently small neighborhood U of p, by the implicit function theorem, $\mathcal{C}$ is a function graph over the x-axis because $\partial_y f(p) \neq 0$. We define the local right branch at p relative to U as the portion of $\mathcal{C}$ in the half plane $x > x_p$. We define the local left branch at p relative to U as the portion of $\mathcal{C}$ in the half plane $x < x_p$.

Definition 5.4: For a point $p \in \mathcal{C} \cap \partial\mathcal{D}$, we define its x-index.

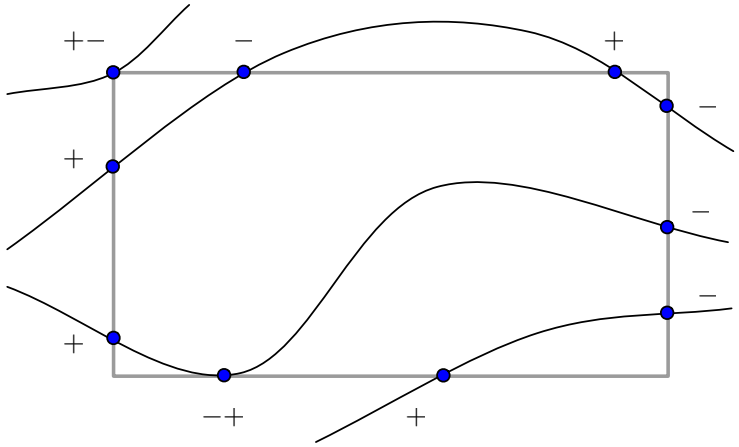


Figure 5.1: x-indexes of an x-regular domain.

$+$ if $\mathcal{C}$ enters $\mathcal{D}$ locally: there exists a local left (resp. right) tangent lying outside (resp. inside) $\mathcal{D}$.

$-$ if $\mathcal{C}$ exits $\mathcal{D}$ locally: there exists a local left (resp. right) tangent lying inside (resp. outside) $\mathcal{D}$.

$+-$ if $\mathcal{C}$ is tangent to $\mathcal{D}$ and does not enter it locally: $\mathcal{C} - \{p\}$ locally lies outside $\mathcal{C}$.

$-+$ if $\mathcal{C}$ is tangent to $\mathcal{D}$ and does not exit it locally: $\mathcal{C} \subset \mathcal{D}$.

Remark 5.3: This is well defined because if there exists a local left (resp. right) tangent lying outside (resp. inside) $\mathcal{D}$, then there cannot exist a local left (resp. right) tangent lying inside (resp. outside) $\mathcal{D}$. Moreover, we necessarily fall into one of these cases because $\partial\mathcal{D} \cap \mathcal{C}$ is 0-dimensional. •

These conditions can be effectively tested using the sign s_y of $\partial_y f$, the order k of the first x derivative of f that does not vanish, and the sign s_x of $\partial_x^k f$. k is well defined because if all these partial derivatives were 0, the whole horizontal line would be included in $\mathcal{C}$ which would mean $\mathcal{C} \cap \partial\mathcal{D}$ is not 0-dimensional.

Remark 5.4: As $\mathcal{C}$ is assumed x-regular (with no vertical tangent) in $\mathcal{D} = [a, b] \times [c, d]$ we can immediately see that if $p \in](a, c), (a, d)[$, we have x-index$(p) = +$. And if $p \in](b, c), (b, d)[$, its x-index is $-$. •

In the following the points with double index (+- or -+) are considered as double points, one with "smaller x component" than the other (although they correspond to a single point that has only one x component). The one with smaller x component gets the left part of the double index, and the one to its right (bigger x component) gets the right part.

Lemma 5.1: If $\mathcal{C}$ is x-regular in $\mathcal{D}$, then a branch of $\mathcal{C} \cap \mathcal{D}$ connects a point p of x-index $+$ to a point q of x-index $-$, such that $x_p < x_q$. $\Diamond$

Proof 5.1: As the curve is x-regular, it has no vertical tangent and thus no closed loop in $\mathcal{D}$. Consequently, each of the interior connected components of $\mathcal{C} \cap \mathcal{D}$ intersects $\partial\mathcal{D}$ in two distinct points $p, q \in \mathcal{C} \cap \partial\mathcal{D}$ (with $x_p \leq x_q$).

Assume that the x-index of p, q are the same. Suppose that this index is $+$. Then for an analytic parameterization $s \in [0,1] \mapsto (x(s), y(s))$ of the branch $[p, q]$ with $(x(0), y(0)) = p$, $(x(1), y(1)) = q$, we have $\partial_s x(0) > 0$, $\partial_s x(1) < 0$. This implies that for a value $0 < s_0 < 1$, $x(s_0) > x(1) = x_q \geq x(0) = x_p$ and that there exists $s_0' \in]0,1[$ such that $x(s_0') = x(1)$. We deduce that $\partial_s x(s)$ vanishes in $[0,1]$ and that the branch $[p, q]$ of $\mathcal{C}$ has a vertical tangent, which is excluded by hypothesis. If the index of p and q is $-$, we exchange the role of p and q and obtain the same contradiction. As $\partial_s x(s) > 0$ for $s \in [0,1]$, we have $x_p < x_q$, which proves the lemma. $\square$

Lemma 5.2: Suppose that $\mathcal{C}$ is x-regular in $\mathcal{D}$ and let p, q be two consecutive points of $\mathcal{C} \cap \partial\mathcal{D}$ with: q such that x_q is minimal among the points with x-index$= -$, and $x_p < x_q$, then p, q belong to the same branch of $\mathcal{C} \cap \mathcal{D}$. $\Diamond$

Proof 5.2: Suppose that p and q are not on the same branch. Let p' the other endpoint of the branch going to q. Let q' this other endpoint of the branch starting from p. By lemma 5.1, x-index$(p') = +$ and $x_{p'} < x_q$. By that same lemma, x-index$(q') = -$ and $x_p < x_{q'}$.

The branch (p', q) separates $\mathcal{D}$ in two connected components. We call C_r the one whose boundary $B_r = \partial C_r$ contains the point p.

Because (p', q) and (p, q') do not intersect, p and q' are in the same connected component of $\mathcal{D} - (p', q)$ and in B_r.

Consider the sub-boundary $\{x \geq x_q\} \cap B_r$. It must be connected. Otherwise the branch (p', q) would intersect $x = x_q$ in two distinct points and the curve

would have a x-critical point in between. We denote by $(q, \tilde{q})$, the endpoints of $\{x \geq x_q\} \cap B_r$ (possibly with $q = \tilde{q}$). We decompose B_r as the union of arcs $B_r = (p', q) \cup (q, \tilde{q}) \cup (\tilde{q}, p')$ with $(q, \tilde{q}) \subset \partial\mathcal{D}$, $(\tilde{q}, p') \subset \partial\mathcal{D}$.

By minimality of x_q, we have $x_{q'} \geq x_q$ so that $q' \in \{x \geq x_q\} \cap B_r = (q, \tilde{q})$. Because $x_p < x_q$ and $p \in \partial D$, we have $p \in (\tilde{q}, p')$.

This proves that p is in-between p' and q' and q' in-between p and q on $\partial\mathcal{D}$. Therefore, p and q cannot be consecutive points of $\mathcal{C}$ on $\partial\mathcal{D}$. By way of contradiction, we conclude that p and q must be on the same branch of $\mathcal{C}$.□

Proposition 5.1: Let $\mathcal{C} = \mathcal{Z}(f)$. If $\mathcal{D}$ is a x-regular domain, the topology of $\mathcal{C}$ in $\mathcal{D}$ is uniquely determined by its intersection $\mathcal{C} \cap \partial\mathcal{D}$ with the boundary of $\mathcal{D}$.

Proof 5.3: We prove the proposition by induction on the number $N(\mathcal{C})$ of points on $\mathcal{C} \cap \partial\mathcal{D}$. We denote this set of points by $\mathcal{L}$. Since the curve has no vertical tangent in $\mathcal{D}$ and has no closed loop, each of the connected components of $\mathcal{C} \cap \overset{\circ}{\mathcal{D}}$ have exactly two distinct endpoints on $\partial\mathcal{D}$. Thus if $N(\mathcal{C}) = 0$, then there is no branch of $\mathcal{C}$ in $\mathcal{D}$. Assume now that $N(\mathcal{C}) > 0$, and let us find two consecutive points p, q of $\mathcal{L}$ with x-index$(p) = +$, x-index$(q) = -$, $x_p < x_q$ and x_q minimal. By lemma 5.2, the points p, q are the endpoints of the branch of $\mathcal{C}$. Removing this branch from $\mathcal{C}$, we obtain a new curve $\mathcal{C}'$ which is still x-regular and such that $N(\mathcal{C}') < N(\mathcal{C})$. We conclude by induction hypothesis, that the topology of $\mathcal{C}'$ and thus of $\mathcal{C}$ is uniquely determined. □

Proposition 5.2: If $\mathcal{C}$ has at most one x-critical or y-critical point $\in \mathcal{D}$, which is also smooth, then its topology in $\mathcal{D}$ is uniquely determined by its intersection with the boundary of $\mathcal{D}$.

Proof 5.4: Suppose $\mathcal{C}$ has at most one x-critical point in $\mathcal{D}$, which is smooth, then the curve is smooth in $\mathcal{D}$ and has no closed loop inside $\mathcal{D}$ (otherwise the number of x-critical points would be at least 2). Therefore, the branches are intersecting $\partial\mathcal{D}$ in two points. If there is no x-critical point on a branch, by Lemma 5.1 their x-index $\in \{-, +\}$ are distinct. If the branch has a x-critical point of even multiplicity, then the x-index of the end-points of the branch in $\mathcal{C}$ are the same. If there are only two points

of $\mathcal{C}$ on ∂D, then this branch is connecting the two points. As the curve is smooth, the branches are not intersecting. If there are more points, and thus more than 2 branches, the branch with the even x-critical point is separating the set of branches into two disjoints subsets of branches with no x-critical points. Changing the orientation of the x-axis if necessary, we can find consecutive points p, q on ∂D which satisfies the hypothesis of lemma 5.2. By this lemma, they are necessarily on the same branch of one of these two subsets. Removing this branch from $\mathcal{C}$ and processing recursively in this way, we end up either with no point on $\partial\mathcal{D}$ or two points on $\partial\mathcal{D}$ with the same x-index. These points are necessarily connected by the branch containing the x-critical point of $\mathcal{C}$ in $\mathcal{D}$. □

5.1.1.2 Singular domains

Let us now deal with simple singular domains. We will assume here that $\mathcal{D}$ contains a unique critical point p of f and that the curve passes through it (*i.e.* it is a singular point of $\mathcal{C}$).

We explain how using topological degree, [76] one can count the number of half branches of $\mathcal{C}$ at p and check if it is the same as the number of points in $\partial\mathcal{D} \cap \mathcal{C}$.

Topological Degree. We recall the definition of the topological degree in two dimensions and how it can be computed. See [76, 98] for more details.

Let $\mathcal{D}$ be a bounded open domain of $\mathbb{R}^2$ and $F = (f_1, f_2) : \mathcal{D} \to \mathbb{R}^2$ a bivariate function which is two times continuously differentiable in $\mathcal{D}$.

A point $p \in \mathbb{R}^2$ is said to be a *regular value* of F on $\mathcal{D}$ if the roots of the equation $F(x, y) = p$ in $\mathcal{D}$ are simple roots, *i.e.* the determinant of the Jacobian J_F of F at these roots is nonzero) where the Jacobian matrix of $F_n = (f_1, \ldots, f_n) \in \mathbb{R}^2$ in $x \in \mathcal{D}$ is

$$J_{F_n} = \nabla F_n(x) = \left(\frac{\partial f_i}{\partial x_j}(x)\right)_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}} \tag{5.4}$$

Definition 5.5 (Topological degree): Let $p \in \mathbb{R}^2$ and suppose further that the roots of the equation $F(x, y) = p$, are not located on the boundary

$\partial\mathcal{D}$. The *topological degree of F at p relative to $\mathcal{D}$*, denoted by $\deg[F, \mathcal{D}, p]$, is defined by

$$\deg[F, \mathcal{D}, p] = \sum_{\mathbf{x}\in\mathcal{D}:F(\mathbf{x})=p} \text{sign}\left(\det(J_F(\mathbf{x}))\right)$$

for p a regular value of F on $\mathcal{D}$ in the connected component of $\mathbb{R}^2 - F(\partial\mathcal{D})$ containing p. If p is not regular, $\deg[F, \mathcal{D}, p]$ is defined as the limit of $\deg[F, \mathcal{D}, p\epsilon]$ when $p\epsilon \to p$.

It can be proved that this construction does not depend on the regular value q in the same connected component of $\mathbb{R}^2 - F(\partial\mathcal{D})$ as p [76]. If p is a regular value of F on $\mathcal{D}$, we can take $q = p$.

Remark 5.5: The topological degree has a geometric interpretation known as the degree of the "Gauss map". It is the number of times $F(p)$ goes around $F(\mathcal{D})$ when p goes around $\mathcal{D}$ one time. And it is negative when F reverses the orientation of $\mathcal{D}$. The red arrows in fig. 5.2 picture the $F(p)$ on the boundary. This viewpoint allows to use the strong geometric intuition behind the gradient field when F is the gradient map of f. •

Let us now give a more explicit formula for computing this topological degree, which involves only information on the boundary of $\mathcal{D}$.

Proposition 5.3: [98] Assume here that the boundary $\mathcal{D}$ is a polygon and that it is decomposed in reverse clock-wise order into the union of segments

$$\partial\mathcal{D} = \cup_{i=1}^{g}[p_i, p_{i+1}], \quad p_{g+1} = p_1,$$

in such a way that one of the component f_{σ_i} ($\sigma_i \in \{1, 2\}$) of $F = (f_1, f_2)$ has a constant sign ($\neq 0$) on $[p_i, p_{i+1}]$. Then, for p a regular value:

$$\deg[F, \mathcal{D}, p] = \frac{1}{8}\sum_{i=1}^{g}(-1)^{\sigma_i - 1}\begin{vmatrix} \text{sign}(f_{\sigma_i}(p_i)) & \text{sign}(f_{\sigma_i}(p_{i+1})) \\ \text{sign}(f_{\sigma_i+1}(p_i)) & \text{sign}(f_{\sigma_i+1}(p_{i+1})) \end{vmatrix} \quad (5.5)$$

where $f_1 = f_3$ and sign(x) denotes the sign of x.

Thus in order to compute the topological degree of F on a domain $\mathcal{D}$ bounded

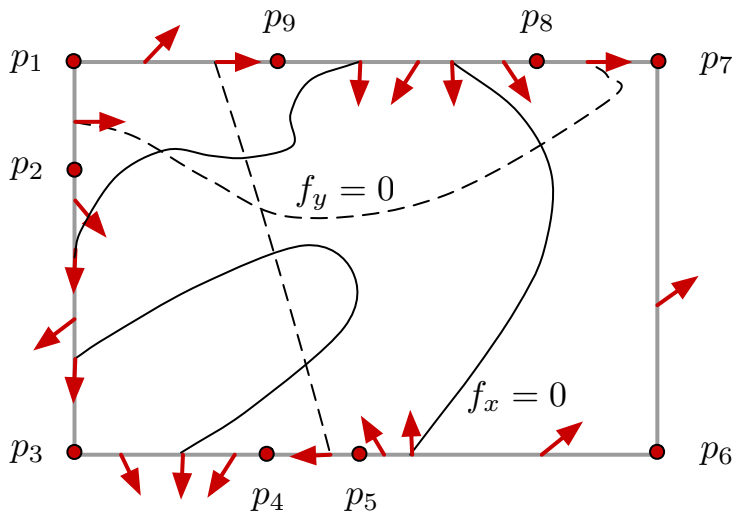


Figure 5.2: Computing the topological degree.

by a polygon, we need to separate the roots of f_1 from the roots of f_2 on $\partial\mathcal{D}$ by points $p_1, \ldots, p_{g+1}$ at which we compute the sign of f_1 and f_2. This will be performed on each segment of the boundary of $\mathcal{D}$, by a univariate root isolation method working simultaneously on f_1 and f_2.

Figure 5.2 shows a sequence of points $p_1, \ldots, p_9$, which decomposes ∂D into segments on which one of the two functions ($f_1 = 0$ and $f_2 = 0$ are represented by the plain and dash curves) has a constant sign. Computing the sign of these functions and applying formula (5.5) yields the topological degree of $F = (f_1, f_2)$ on $\mathcal{D}$ at p.

Counting the number of branches. Let us consider a curve $\mathcal{C}$ in a domain $\mathcal{D} \subset \mathbb{R}^2$, defined by the equation $f(x, y) = 0$ with $f(x, y) \in \mathbb{R}[x, y]$. Let $\nabla f = (\partial_x f, \partial_y f)$ be the gradient of f. A point $p \in \mathcal{C}$ is singular if $\nabla f(p) = 0$. We define a real half branch of $\mathcal{C}$ at p, as a connected component of $\mathcal{C} - \{p\} \cap \mathcal{D}(p, \epsilon)$ for $\epsilon > 0$ small enough.

The topological degree of ∇f can be used to count the number of half branches at a singular point, based on the following theorem:

Theorem 5.1: (Khimshiashvili [8, 72, 99]) Suppose that p is the only root of $\nabla f = 0$ in $\mathcal{D}$. Then the number N of real half branches at p of the

curve defined by $f(x,y) = f(p)$ is

$$N = 2\,(1 - \deg[\nabla f, \mathcal{D}, p]). \tag{5.6}$$

We will denote by $N(f, \mathcal{D})$ the number given by formula (5.6).

In order to count the number of branches of $\mathcal{C}$ at a singular point $p \in \mathcal{C}$, first we isolate the singular point p in a domain $\mathcal{D}$, so that ∇f does not vanishes elsewhere in $\mathcal{D}$. Then we compute the topological degree $\deg[\nabla f, \mathcal{D}, p]$, as described previously, by isolating the roots of $\partial_x f$ and $\partial_y f$ on $\partial\mathcal{D}$.

Let us describe now the algorithm used to compute the topological degree of ∇f in a domain $\mathcal{D} = [a,b] \times [c,d]$. According to formula (5.5), this reduces to separating the roots of $\partial_x f$ $\partial_y f$ on the boundary of $\mathcal{D}$, which consists in 2 horizontal and vertical segments. The problem can thus be transformed into isolating the roots of univariate polynomials on a given interval. Hereafter, these polynomials will be called $g_1(t), g_2(t)$ and the interval $[u,v] \subset \mathbb{R}$. For instance, one of the 4 cases to consider will be $g_1(t) = \partial_x f(t,c), g_2(t) = \partial_y f(t,c)$, $u = a, v = b$. We recall briefly the subdivision method described in [46, 82, 86] and section 1.2.2, which can be used for this purpose. First we express our polynomials $g_1(t)$, $g_2(t)$ of degree d_1, d_2 in the Bernstein bases $(B^i_{d_k}(t;u,v))_{i=0,\ldots,d_k}$ $(k = 1,2)$, on the interval $[u,v]$:

$$g_k = \sum_{i=0}^{d_k} \lambda_{k,i}\, B^i_{d_k}(t;u,v), k = 1,2,$$

where $B^i_d(t;u,v) = \binom{d}{i}(t-u)^i(v-t)^d(v-u)^{-d}$. The number of sign variations of the sequence $\lambda_k = [\lambda_{k,0}, \ldots, \lambda_{k,d_k}]$ $(k = 1,2)$ is denoted $V(g_k;[u,v])$. By a variant of Descartes rule [12], it bounds the number of roots of g_k on the interval $[u,v]$ and is equal modulo 2 to it. Thus if $V(g_k;[u,v]) = 0$, g_k has no root in the interval $[u,v]$, if $V(g_k;[u,v]) = 1$, g_k has exactly one root in the interval $[u,v]$. This is the main ingredient of the algorithm [46], which splits the interval using de Casteljau subdivision algorithm [48] if $V(g_k;[u,v]) > 1$; stores the interval if $V(g_k;[u,v]) = 1$ and removes it otherwise. It iterates the process on each sub-intervals until the number of sign variation is 0 or 1. The complexity analysis of the algorithm is described in [46]. See also [36].

In our case, we need to compute intervals on which one of the polynomial g_1 or g_2 has a constant sign. Thus we replace the subdivision test by the following: if $V(g_1;[u,v]) = 0$ or $V(g_2;[u,v]) = 0$, we store the interval $[u,v]$; otherwise we split it and compute the Bernstein representation of g_k $(k = 1, 2)$ on the two sub-intervals using De Casteljau algorithm and repeat the process.

This yields the following algorithm for computing the topological degree of $\nabla f = (f_1(x,y), f_2(x,y))$ on $\mathcal{D}$:

Algorithm 5.1: Topological degree of (f_1, f_2).

Input: a polynomial $f(x,y) \in \mathbb{Q}[x,y]$ and a domain $\mathcal{D} = [a,b] \times [c,d]$
Output: N the topological degree of ∇f on $\mathcal{D}$ at $(0,0)$
$\mathcal{B} := \{\}$ (a circular list representing the boundary $\partial\mathcal{D}$) ;
foreach *side segment* I *of the box* $\mathcal{D}$ **do**
  Compute the restriction $g_1(t)$ (resp. $g_2(t)$) of f_1 (resp. f_2) on this side segment I and its representation in the corresponding Bernstein basis ;
  $\mathcal{L} := \{I\}$;
  while $\mathcal{L} \neq \emptyset$ **do**
    pop up an interval $[p,q]$ from $\mathcal{L}$;
    if $V(g_1;p,q) = 0$ *or* $V(g_2;p,q) = 0$ **then**
      $\mathcal{B} \leftarrow$ `clockwise`$([p,q])$;
    else
      $\mathcal{L} \leftarrow$ `split`$([p,q])$;
    end
  end
end
Compute N given by formula (5.5) for the points in the circular list $\mathcal{B}$;

If we assume that $\partial_x f$ and $\partial_y f$ have no common root on the boundary of $\mathcal{D}$, it can be proved (by the same arguments as those used in [12, 46, 82]) that this algorithm terminates and outputs a sequence of intervals on which one of the functions g_1, g_2 has no sign variation. The complexity analysis of this method is described in [86]. This analysis can be improved by exploiting the recent results in [46].

5.1.2 Topology

The regularity test ensures that we are able to locally compute the topology of a curve within a given domain. The way of computing this topology

comes together with the regularity test. Again, we distinguish the singular and non singular configurations of a curve inside a subdivision cell.

5.1.2.1 Regular domains

Propositions 5.1 and 5.2 lead to the following algorithm to connect points on the boundary $\partial\mathcal{D}$:

Algorithm 5.2: Connection for a regular domain.

Input: an algebraic curve $\mathcal{C}$ and a domain $\mathcal{D} = [a, b] \times [c, d] \subset \mathbb{R}^2$ such that $\mathcal{C}$ has no vertical tangent in $\mathcal{D}$

Output: the set $\mathcal{B}$ of branches of $\mathcal{C}$ in $\mathcal{D}$

Isolate the points $\mathcal{C} \cap \partial\mathcal{D}$ and compute their x-index ;

Order the points of $\mathcal{C} \cap \partial\mathcal{D}$ with non-zero x-indexes clockwise and store them in the circular list $\mathcal{L}$;

while $\mathcal{L} \neq \emptyset$ **do**

  Choose q such that x_q is minimal in $\mathcal{L}$ with x-index $-$;

  Take the point p that follows or precedes q in $\mathcal{L}$ such that $x_p < x_q$ (thus x-index$(p) = +$) ;

  $\mathcal{B} \leftarrow \mathfrak{b} = [p, q]$;

  $\mathcal{L} = \mathcal{L} \setminus p$;

  $\mathcal{L} = \mathcal{L} \setminus q$;

end

Notice that a sufficient condition for the x (resp. y) regularity of f in a domain $\mathcal{D}$ is that the coefficients of ∂_y (resp. $\partial_x f$) in the Bernstein basis on $\mathcal{D}$ are all > 0 or < 0. In this case the connection algorithm can be simplified even further. See [3] for more details.

5.1.2.2 Singular domains

Finally we prove that the topology in a simple singular domain $\mathcal{D}$ is conic and write a connection algorithm for theses domains.

Let $A \subset \mathbb{R}^n$ and $p \in \mathbb{R}^n$. We call *cone* over A with center p the set $p \star A := \bigcup_{q \in A} [p, q]$.

Proposition 5.4: Let $\mathcal{D}$ be a convex, simply singular domain, *i.e.* $\mathcal{D}$ is convex such that there is a unique singular point s and no other critical point of f in $\mathcal{D}$, and such that the number of half branches of $\mathcal{C}$ at s is $\sharp(\partial\mathcal{D} \cap \mathcal{C})$. Then the topology of $\mathcal{D}$ is conic, *i.e.* for any point p in the inside $\mathcal{D}$, $\mathcal{Z}(f) \cap \mathcal{D}$ can be deformed into $p \star (\partial\mathcal{D} \cap \mathcal{C})$.

Proof 5.5: s is the unique critical point of f in $\mathcal{D}$. If the endpoint of a half branch at s is not on $\partial\mathcal{D}$, the half branch has to be a closed loop inside $\mathcal{D}$. In that case, f would be extremal at some point p ($\neq s$) inside the loop, and p would be another critical point of f inside $\mathcal{D}$. Thus, by way of contradiction, the endpoints of half branches at s have to be on $\partial\mathcal{D}$.

The number of half branches at s is exactly $\sharp(\partial\mathcal{D} \cap \mathcal{C})$. As no two half branches can have the same endpoint on $\partial\mathcal{D}$ (that would be another singular point in $\mathcal{D}$), all points on $\partial\mathcal{D}$ are endpoints of half branches at s. Thus, at this point, we know that the connected component of s inside $\mathcal{D}$ is conic.

But in fact, there is no other connected component. Suppose we have another connected component α of $\mathcal{C}$ intersecting $\mathcal{D}$. As all points of $\partial\mathcal{D} \cap \mathcal{C}$ are connected to s, we have $\alpha \subset \mathcal{D}$. α is a smooth 1-dimensional manifold because s is the only singular point. Therefore α is a closed loop inside $\mathcal{D}$ (s might be inside it). We look at the complement of $\mathcal{C}$ in $\mathbb{R}^2$, it has a bounded connected component because one of them is inside the loop α. As f vanishes on the boundary of this component, f has an extremum inside it. This extremum cannot be s as it is in the complement of f, which is impossible. Thus, $\mathcal{C} \cap \mathcal{D}$ is connected.

This concludes our argument as we have proved that $\mathcal{C} \cap \mathcal{D}$ is equal to the connected component of s inside $\mathcal{D}$ and that it has the topology of a cone over $\partial\mathcal{D} \cap \mathcal{C}$ which is what we claimed. □

Remark 5.6: We do not have to suppose that $\mathcal{D}$ is convex, simply connected would suffice. But we only work with convex sets (boxes) and the denomination "conic topology" originates from the convex case. •

In the end the connection algorithm is extremely simple. We just proved that the topology inside these domains is conic, that is $\mathcal{C} \cap \mathcal{D}$ can be deformed into a cone over $\mathcal{C} \cap \partial\mathcal{D}$. Therefore the connection algorithm for (convex)

simply singular domains is to first compute the points q_i of $\mathcal{C} \cap \partial\mathcal{D}$, compute the singular point s inside $\mathcal{D}$ and finally for every q_i, connect q_i to s by a half branch segment $\mathfrak{b}_i = [s, q_i]$.

Algorithm 5.3: Connection for a singular domain.

Input: an algebraic curve $\mathcal{C}$ and a domain $\mathcal{D} = [a, b] \times [c, d] \subset \mathbb{R}^2$ such that $\mathcal{C}$ has only one singular point s in $\mathcal{D}$

Output: the set $\mathcal{B}$ of branches of $\mathcal{C}$ in $\mathcal{D}$

$\mathcal{B} := \{\}$;

Isolate the points $Q = \{\mathcal{C} \cap \partial\mathcal{D}\}$;

forall $q_i \in Q$ **do**

Connect s to q_i by a half branch segment $\mathfrak{b}_i = [s, q_i]$;

$\mathcal{B} \leftarrow \mathfrak{b}_i$;

end

Using the method described in section 5.1.2, we obtain the topology of the curve within a regular cell of subdivision, applying the connection algorithm 5.2. The topology of local regions within the regular cell is easily deduced from the topology of the curves lying in the cell together with its corner points. The edges of regions are oriented counterclockwise to ease up following steps of the arrangement algorithm and specify on which side of the border lies the interior of a region.

After the segmentation step, the detection of conflicts reduces either to intersecting regular segments of two different objects or to testing that an endpoint of a regular segment of an object belongs to a given region. To this end, our favorite polynomial solver is used to solve the bivariate polynomial equations f_k, f_l representing the the objects in potential conflict in the region of interest.

The specialization of the generic arrangement algorithm to the case of implicit curves has been completely implemented with the algebraic geometric modeling software (see chapter 8). The efficiency of the topology algorithm presented here allows a real time manipulation of algebraic objects within the software, whereas current solutions usually only propose ray tracing algorithm for visualization. Chapter 9 gives some significant illustrations.

The dynamic part of the arrangement algorithm has lead to a daemon in the modeler which looks for insertions and removal of objects to maintain

its data structure. The generic part has lead to an abstract algorithm implemented following the template method and visitor design patterns [54]. We have put a special emphasis on keeping the structure accessible to the user. Using the model-view-controller pattern, data structures such as the augmented influence graph and the various quadtrees can be interactively displayed and queried for point location (see figure 9.13).

5.2 Parametric curves

The family of parametrized objects counts a large amount of representations. We present the case of polynomial rational curves and the one of B-spline curves, giving a hint on how to compute the elements needed for the specialization of our generic algorithm to these representations.

We denote by σ_k the parameterization of the curve $\mathcal{C}$ that we analyze.

5.2.1 Regularity

The regularity test for parametric curves also distinguishes singular cases (a self-intersecting curve) and non-singular cases.

Empty cells. It is possible to compute intersection points of a uniform rational curve (see definition in section 2.1.2) with any line segment parallel to the axis, by solving the following univariate problems:

$$\begin{cases} y(t) - y_{min} = 0 & \text{and} \quad x_{min} \leq x(t) \leq x_{max} \\ y(t) - y_{max} = 0 & \text{and} \quad x_{min} \leq x(t) \leq x_{max} \\ x(t) - x_{min} = 0 & \text{and} \quad y_{min} \leq y(t) \leq y_{max} \\ x(t) - x_{max} = 0 & \text{and} \quad y_{min} \leq y(t) \leq y_{max} \end{cases}$$

Should no such point exist, we ensure that the cell is empty.

Non-singular cells. In order to meet the configurations presented in figure 4.1, we first partition the interval $[t_{min}, t_{max}]$ in intervals where $x'(t) \neq 0$ and $y'(t) \neq 0$ on the interior. On each one of these intervals, the curve is

x and y monotonic. The corresponding bounding boxes of these curve segments are defined by image by σ_k of the end points of the interval. This builds a monotonous segmentation of the curve which allows us to efficiently test its monotony.

Critical points can be characterized as follows:

$$\begin{cases} x'(t)w(t) - x(t)w'(t) = 0 \\ y'(t)w(t) - y(t)w'(t) = 0 \end{cases}$$

The curve $\mathcal{C}$ is deemed regular in $\mathcal{D}$ if it intersects it twice and features at most on critical point in $\mathcal{D}$.

Singular cells. Next, we localize which pairs of bounding boxes of two non-consecutive segments intersect, using the segmentation structure described in section 4.2. If we find two such boxes, which are the images of the intervals I and I', we test if there exists $(t, s) \in I \times I'$ with $t \neq s$ such that $x(t) = x(s)$ and $y(t) = y(s)$. This reduces to solving a bivariate polynomial system of equations.

Self-intersection points can be characterized as follows: $\exists (t, s), t \neq s$ verifying

$$\begin{cases} (x(t)w(s) - x(s)w(t))/(t-s) &= 0 \\ (y(t)w(s) - y(s)w(t))/(t-s) &= 0 \end{cases}$$

The curve $\mathcal{C}$ is deemed regular in $\mathcal{D}$ if it intersects it four times and features at most on singular point in $\mathcal{D}$.

5.2.2 Topology

Again, we finally turn around these points on the border of the cell in clockwise order and connect them to points of interest inside the cell, such as self-intersection points or points where a vertical or horizontal tangent exist.

To compute the intersection points of two parametric curves σ_k, σ_l (with $k \neq l$), we solve the bivariate system $\sigma_k(t) = \sigma_l(s)$, with t, s in the intervals $\subset [0, 1]$ corresponding to the curve segments. Here again, we can use again our

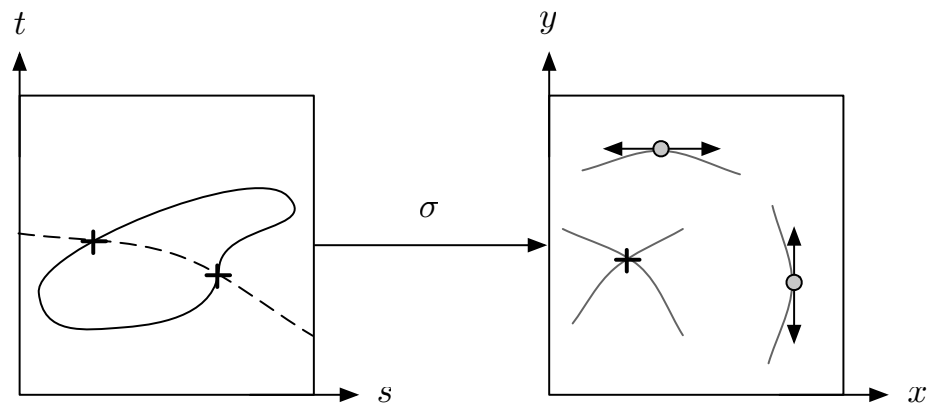


Figure 5.3: Mapping of an implicit curve defined in a parameter space.

polynomial solver (chapter 1) for this purpose. This allows to detect conflicts between regions which edges are made up of parametric curve segments.

5.3 Image of an implicit curve

Another situation that will happen in following investigations is described here. Consider the case, illustrated in figure 5.3, of a curve defined implicitly by $f(s,t)$ in a parameter space. A key ingredient that we will have to perform is to analyze the curve in the parameter space to reach useful conclusions concerning its mapping by some parameterization *e.g.* $\sigma = (x(s,t), y(s,t))$ such as the identification of critical or singular points.

As an example x-critical points are defined for a couple of parameters (s,t) where $\partial_u x = 0$, for u a local parameter of the implicit curve such that $f(s,t) = 0$ is equivalent to $f(s(u), t(u)) = 0$. With such a local parameterization,

$$\partial_u x = 0 \Leftrightarrow \partial_s x(s,t)\ \partial_u s + \partial_t x(s,t)\ \partial_u t = 0 \tag{5.7}$$

In particular, $(\partial_u s, \partial_u t)$ is the tangent vector to the curve $f(s,t) = 0$, and is orthogonal to the gradient $\nabla f(s,t)$ of f, so we have the following relation:

$$(\partial_u s, \partial_u t) \perp \nabla x(s,t) \Leftrightarrow \partial_s x(s,t)\ \partial_t f(s,t) - \partial_t x(s,t)\ \partial_s f(s,t) = 0 \tag{5.8}$$

providing a formula which enables us to compute such points.

Similarly, we obtain the following formulation for y-critical points:

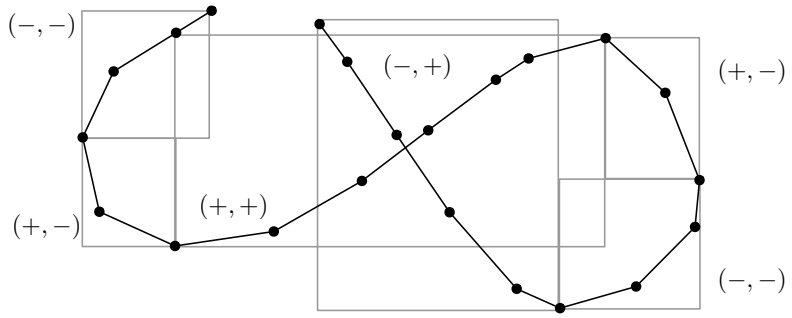
$$\partial_s y(s,t)\ \partial_t f(s,t) - \partial_t y(s,t)\ \partial_s f(s,t) = 0 \tag{5.9}$$

Singular points are the images of points either verifing both equations 5.8 and 5.9 or lying at the intersection of $f(s,t)$ and $g(s,t)$, where g is the curve of self-intersection, *i.e.* the set of points having a common image by the parameterization σ. Note for example on the figure, where the self-intersection curve is dashed, that the two cross marked points in the parameter space both map to the cross marked singular point in the xy-space.

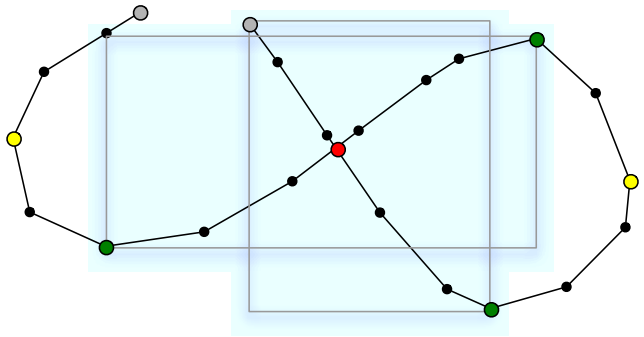
5.4 Piecewise linear curves

When computing an arrangement of piecewise linear curves, the only challenge is to be able to check if the line segments defining a curve are regular in a cell of subdivision. Indeed, computing the intersection of a linear curve segment with a cell border poses no problem since they have the same representation and detecting self-intersection points is not difficult either since it consists in intersecting two line segments. Besides, the topology of the curve is included in its representation so we just have to deduce regions from it in an efficient way.

The treatment of piecewise linear curves $p_0, \ldots, p_s$ (with $p_i \in \mathbb{R}^2$) is similar to the case of parametric curves and is illustrated in figure 5.4. First, to each segment p_i, p_{i+1} we associate a so-called "monotony code" (corresponding to the sign of the coordinates of the vector $p_i\overrightarrow{p_{i+1}}$. Then, segments are gathered in a *monotonous segmentation*, a balanced binary tree data structure, similar to the region segmentation which is queried the same way to compute self-intersection points. The only difference between the region segmentation and the monotonous segmentation is that in the latter, curve segments are gathered with regard of their monotony code, $M_k = M(t_k, t_{k+1}) = (\text{sign}(x(t_{k+1}) - x(t_k)), \text{sign}(y(t_{k+1}) - y(t_k)))$, figure 5.4(a). These segmentations are then compared to find non adjacent intersecting nodes (comparing the coordinates of their associated edges' bound-



(a) Segmenting the curve.



(b) Querying the segmentation.

Figure 5.4: A generic procedure to compute features of a piecewise linear curves.

ing boxes) in which, intersection points are computed using usual methods 5.4(b).

Chapter 6

Specialization for surfaces

The generic approach for computing an arrangement of objects presented in chapter 4 is easily specialized in 2-dimensional space (as shown in chapter 5). Thanks to its subdivision scheme, we can specialize it in 3-dimensional space as well. Even though this specialization is not yet implemented, we sketch our proposition here.

Again, the specialization consists in providing regularity criteria which ensure that the topology of a surface $\mathcal{S}$ is well defined in a cell of subdivision (a domain $\mathcal{D}$ which a cube in the space), together with corresponding connection algorithms. These requirements are the same for both the static and dynamic versions of the generic arrangement algorithm.

To dynamically deal with a collection of surfaces, the specialization must also provide a way to check whether two objects are intersecting or not, and, if so, the topology of their intersection. It is shown that the generic efficient scheme to segment the boundary of regions also applies when computing an arrangement in 3 dimensions.

Section 6.1 explores the case of implicit surfaces, following a classification of the different regularity cases that can happen. In particular, intersection or self-intersection curves of implicit surfaces are defined by spatial implicit curves. It is therefore important to be able to compute the topology of such a curve and be able to analyze how it behaves in a given domain.

Section 6.2 explains how the generic algorithm can be specialized to the

case of parametric surfaces. Since their topology computation does not pose any problem, the challenging question is to efficiently compute intersection and self-intersection of such objects and guarantee their topology thanks to a regularity criterion so that regions can be computed in resulting regular cells.

6.1 Implicit surfaces

In this section, we consider a surface $\mathcal{S}$ defined by the equation $f(x, y, z) = 0$, with $f \in \mathbb{Q}[x, y, z]$. We assume that f is squarefree, that is f has no irreducible factors of multiplicity ≥ 2. For more details, see [3].

For a subset $S \subset \mathbb{R}^2$, we denote by $\overset{\circ}{S}$ its *interior*, by $\overline{S}$ its *closure*, and by ∂S its *boundary*.

We call *domain* any closed set $\mathcal{D}$ such that $\overline{\overset{\circ}{\mathcal{D}}} = \mathcal{D}$ and $\mathcal{D}$ is simply connected.

We call *region* any open set R which is a connected component of the complement of an algebraic surface.

Unlike the 2 dimensional case, the shape of the surface and of its singular locus can be really complicated. Topologically we can characterize the situation as follows:

- Near a 2-dimensional stratum the topology is the same as a hyperplane.
- Near a 1-dimensional stratum the topology is the same as a cylinder.
- Near a 0-dimensional stratum the topology is the same as a cone.

Moreover, we know that only one of these three situations, illustrated in figure 6.1 can and will happen locally. So we just have to design a solution for each one of the above three cases.

The regularity criterion to deduce regions from the topology of a possibly singular surface consists in ensuring that the surface locally (in a box of subdivision) corresponds to one of the criteria shown in figure 6.1. In addition to these configurations we obviously consider the case of an empty cell where the topology is trivially obtained from the corners of the cell.

The first case illustrated corresponds to a regular surface patch. The second case corresponds to a regular intersection of two patches of a surface. The

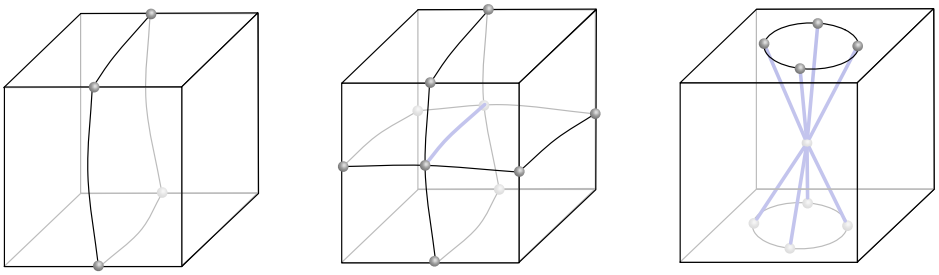


Figure 6.1: Regularity criterion for surfaces.

third case corresponds to two or more surface patches touching at a singular point.

The critical points in the x-direction (resp. y-direction and z-direction) can be obtained by (resp.) solving:

$$\left\{\begin{array}{rcl} f_k(x,y,z) & = & 0 \\ \partial_y f_k(x,y,z) & = & 0 \\ \partial_z f_k(x,y,z) & = & 0 \end{array}\right. \quad \left\{\begin{array}{rcl} f_k(x,y,z) & = & 0 \\ \partial_x f_k(x,y,z) & = & 0 \\ \partial_z f_k(x,y,z) & = & 0 \end{array}\right. \quad \left\{\begin{array}{rcl} f_k(x,y,z) & = & 0 \\ \partial_x f_k(x,y,z) & = & 0 \\ \partial_y f_k(x,y,z) & = & 0 \end{array}\right.$$

To compute singular points (isolated or self-intersection points), we solve (*e.g.* using the multivariate subdivision solver):

$$\left\{\begin{array}{rcl} f_k(x,y,z) & = & 0 \\ \partial_x f_k(x,y,z) & = & 0 \\ \partial_y f_k(x,y,z) & = & 0 \\ \partial_z f_k(x,y,z) & = & 0 \end{array}\right.$$

The intersection points of two surfaces defined by the polynomials f_k, f_l in a cell are obtained by computing the spatial implicit curve:

$$\left\{\begin{array}{rcl} f_k(x,y,z) & = & 0 \\ f_l(x,y,z) & = & 0 \end{array}\right.$$

Intersection curves of the surface patch with facets of the cell are trivially obtained by a variable substitution in the polynomial defining the surface.

The polar variety will be used to check the subdivision cells for regularity.

Definition 6.1 (Polar variety): The polar variety $\mathcal{P}_f$ of a surface $\mathcal{S}$ defined by the polynomial equation $f(x, y, z) = 0$ cuts the surface at the points of self-intersection and the points that have vertical tangents:

$$\begin{cases} f(x,y,z) &= 0 \\ \partial_z f(x,y,z) &= 0 \end{cases} \tag{6.1}$$

6.1.1 Regularity

In this section, we consider a surface $\mathcal{S}$ in $\mathbb{R}^3$, defined by the equation $f(x, y, z) = 0$ with $f \in \mathbb{Q}[x, y, z]$ and a domain $\mathcal{D} = [a, b] \times [c, d] \times [e, f] \subset \mathbb{R}^3$.

To check a cell for regularity we will use the 2-dimensional planar regularity criterion of section 5.1.1 for each facet of the cube, for each configuration identified here before, and also analyze the silhouette of the surface in the domain using the polar variety. The next section explains how to compute the topology of such a curve, and also applies for any three dimensional curve defined by two polynomials.

6.1.1.1 Spatial implicit curves

Let $\mathcal{C}$ be a curve of $\mathbb{R}^3$ defined by the two polynomials $f(x, y, z)$ and $g(x, y, z)$. We assume that (f, g) is radical or equivalently that the resultant of f and g with respect to z after a generic change of coordinate is squarefree.

The tangent vector on $\mathcal{C}$ serves as an indicator of the topological features of the curve. While it is computationally prohibitive to compute the tangent vector field at each point of $\mathcal{C}$, we can reach some useful conclusion about the topology of the curve by looking at the tangent vector field defined by:

$$\mathbf{t} = \bigtriangledown(f) \wedge \bigtriangledown(g) = \begin{vmatrix} \mathbf{e}_x & \mathbf{e}_y & \mathbf{e}_z \\ \partial_x f & \partial_y f & \partial_z f \\ \partial_x g & \partial_y g & \partial_z g \end{vmatrix} \tag{6.2}$$

where $\mathbf{e}_x$, $\mathbf{e}_y$ and $\mathbf{e}_z$ are the unit vectors along the principle axis x, y and z.

Remark 6.1: Singularities on the curve can be easily characterized, as $\mathbf{t}$ vanishes at those points. •

Similar to the 2D case, we can represent f, g and each component of $\mathbf{t}$ in the Bernstein basis for the domain $\mathcal{D}$.

As we will see, the sign changes of the resulting Bernstein coefficients will make it possible to test the regularity of the curve with minimal effort.

The following is a regularity criterion that applies for space curves which allows us to deduce the topology of $\mathcal{C}$ in the domain $\mathcal{D}$:

Proposition 6.1: [75] Let $\mathcal{C}$ be a 3D spatial curve defined by $f = 0$ and $g = 0$. If 1. $\mathbf{t}_x(\mathbf{x}) \neq 0$ on $\mathcal{D}$, and 2. $\partial_y h \neq 0$ on z-faces, and $\partial_z h \neq 0$ and it has the same sign on both y-faces of B, for $h = f$ or $h = g$, then the topology of $\mathcal{C}$ is uniquely determined from the points $\mathcal{C} \cap \partial B$.

A similar criterion applies by symmetry, exchanging the roles of the x, y, z coordinates. If one of these criteria applies with $\mathbf{t}_i(x) \neq 0$ on $\mathcal{D}$ (for $i = x, y, z$), we will say that $\mathcal{C}$ is i-regular on $\mathcal{D}$.

From a practical point of view, the test $\mathbf{t}_i(x) \neq 0$ or $\partial_i(h) \neq 0$ for $i = x, y$ or z, $h = f$ or g can be replaced by the stronger condition that their coefficients in the Bernstein basis of $\mathcal{D}$ have a constant sign, which is straightforward to check. Similarly, such a property on the faces of $\mathcal{D}$ is also direct to check, since the coefficients of a polynomial on a facet form a subset of the coefficients of this polynomial in the box.

In addition to these tests, we also test whether both surfaces defining the curve $\mathcal{C}$ penetrate the cell, since a point on the curve must lie on both surfaces. This test could be done by looking at the sign changes of the Bernstein coefficients of the surfaces with respect to that cell. If no sign change occurs, we can rule out the possibility that the cell contains any portion of the curve $\mathcal{C}$, and thus terminate the subdivision early. In this case, we will also say that the cell is regular.

This regularity criterion is sufficient for us to uniquely construct the topological graph $\mathcal{G}$ of $\mathcal{C}$ within $\mathcal{D}$. Without loss of generality, we suppose that the curve $\mathcal{C}$ is x-regular in $\mathcal{D}$. Hence, there is no singularity of $\mathcal{C}$ in $\mathcal{D}$. Furthermore, this also guarantees that there is no "turning-back" of the curve tangent along the x-direction, so the mapping of $\mathcal{C}$ onto the x axis is injective. Intuitively, the mapped curve should be a series of non-overlapping

line segments, whose end points correspond to the intersections between the curve $\mathcal{C}$ and the cell. Such a mapping is injective.

This property leads us to a unique way to connect those intersection points, once they are computed in order to obtain a graph representing the topology of $\mathcal{C}$, similarly to the two dimensional method (see section 5.1.2).

In order to apply this algorithm, we need to compute the points of $\mathcal{C} \cap \mathcal{D}$, that is to solve a bivariate system for each facet of $\mathcal{D}$. This is performed by applying the algorithm described in section 1.2.2.

The special treatment of points of $\mathcal{C}$ on an edge of $\mathcal{D}$ or where $\mathcal{C}$ is tangent to a face requires the computation of tangency information at these points. This is performed by evaluating the derivatives of the defining equations of $\mathcal{C}$ at these points.

Collecting these properties, we obtain the subdivision algorithm 6.1, which subdivides the domain $\mathcal{D}$ until some size ϵ, while the curve is not regular in current subdivision cells. It relies on a bivariate solver, for computing the intersection of the curve with the faces of the box.

Algorithm 6.1: Topology of an implicit spatial curve.

Input: A curve $\mathcal{C}$ defined by equations $f_1 = 0, f_2 = 0, \ldots, f_k = 0$, a domain $\mathcal{D} = [a_0, b_0] \times [a_1, b_1] \times [a_2, b_2] \subset \mathbb{R}^3$ and $\epsilon > 0$

Output: A set of points p and a set of arcs connecting them

for $1 \leq i < j \leq k$ **do**
  Compute the Bernstein coefficients of the x, y, z coordinates of $\nabla f_i \wedge \nabla f_j$ in $\mathcal{D}$;
  Check that they are of the same sign for one of the coordinates (say x) ;
  Check the x-regularity condition on the facets of $\mathcal{D}$;
end
if *such a pair* (i, j) *satisfying the previous regularity condition exists* **then**
  Compute the points of $\mathcal{C} \cap \partial\mathcal{D}$ and connect them ;
else if $|\mathcal{D}| > \epsilon$ **then**
  Subdivide $\mathcal{D}$ and proceed recursively on each sub-domain ;
else
  Compute the points $\mathcal{C} \cap \partial B$ and connect them to $p \in \overset{\circ}{B}$;
end

Proposition 6.2: For $\epsilon > 0$ small enough, the graph of points and arcs computed by the algorithm has the same topology as $\mathcal{C} \cap B$.

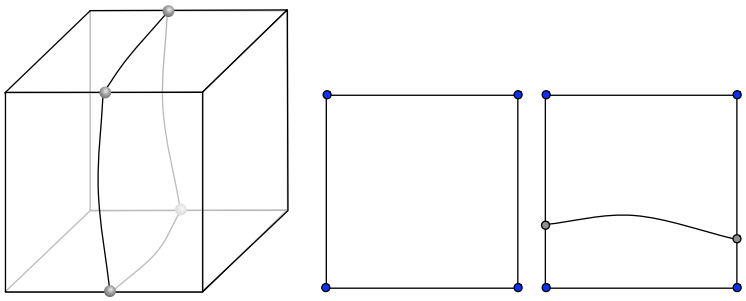


Figure 6.2: Regularity criterion for a 2-dimensional stratum.

6.1.1.2 2-dimensional stratum

Definition 6.2: The surface $\mathcal{S}$ is 2-*regular* in the domain $\mathcal{D}$ if the intersection curves of the surface patch with the cell $\mathcal{S} \cap \partial\mathcal{D}$ are regular and if no critical point lie in the cell $\mathcal{S} \cap \overset{\circ}{\mathcal{D}} = \emptyset$.

The regularity test in the case of a 2-dimensional stratum in a domain $\mathcal{D}$ performs as follows:

1. Check each facet for 2-dimensional regularity using the following criteria: (a) empty cell (b) regular cell with one curve segment
2. Check that the polar variety $\mathcal{P}$ does not intersect $\mathcal{D}$ ($\mathcal{P} \cap \mathcal{D} = \emptyset$).

6.1.1.3 1-dimensional stratum

Definition 6.3: The surface $\mathcal{S}$ is 1-*regular* in the domain $\mathcal{D}$ if the intersection curves of the surface patch with the cell $\mathcal{S} \cap \partial\mathcal{D}$ are regular and if $\mathcal{S} \cap \overset{\circ}{\mathcal{D}} \neq \emptyset$ and $\mathcal{P}$ is regular in $\mathcal{D}$.

The regularity test in the case of a 1-dimensional stratum in a domain $\mathcal{D}$ performs as follows:

1. Check each facet for 2-dimensional regularity using the following criteria: (a) regular cell with one curve segment (b) star shaped singularity with two curve segments
2. Check that the polar variety $\mathcal{P}$ features only one regular branch in $\mathcal{D}$ ($\mathcal{P} \cap \mathcal{D} \neq \emptyset$).

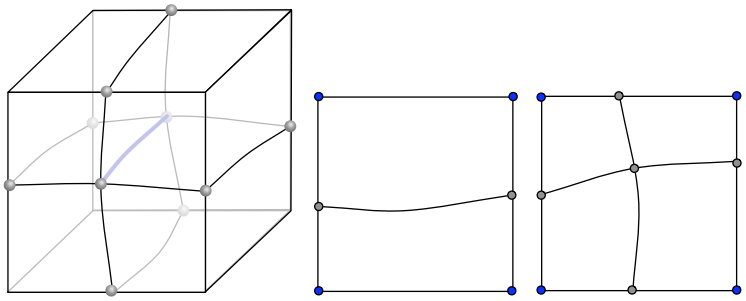


Figure 6.3: Regularity criterion for a 1-dimensional stratum.

6.1.1.4 0-dimensional stratum

Definition 6.4: The surface $\mathcal{S}$ is 0-*regular* in the domain $\mathcal{D}$ if the intersection curves of the surface patch with the cell $\mathcal{S} \cap \partial\mathcal{D}$ are regular and if $\Pi_z(\mathcal{P}) = Res_z(f, \partial_z f)$ is regular and $\mathcal{P}$ has only one singular point in $\overset{\circ}{\mathcal{D}}$.

Remark 6.2: $\mathcal{P}$ corresponds to the silhouette of the surface with respect to the z axis. This could be a problem if some intersection points of the curve with the border of the cell $\mathcal{P} \cap \partial\mathcal{D}$ coincide with the (unique) singular locus of $\mathcal{P}$ for the z coordinate. A more general curve to characterize the silhouette of $\mathcal{S}$ avoiding this problem can be obtained by *shearing*. In this case, we would define $\mathcal{P} = \{f = 0 \cap \lambda_1\partial_x f + \lambda_2\partial_y f + \lambda_3\partial_z f = 0\}$ for random $\lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}$ and note Δ the corresponding shearing direction. •

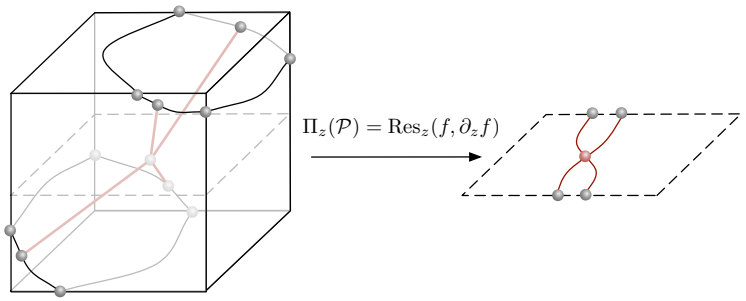


Figure 6.4: Regularity criterion for a 0-dimensional stratum.

The regularity test in the case of a 0-dimensional stratum in a domain $\mathcal{D}$ performs as follows:

1. Check each facet for 2-dimensional regularity using the following criteria: (a) empty cell (b) regular cell with one curve segment
2. Check that the polar variety features only one singular point in $\mathcal{D}$.
3. Compute the projection $\Pi_z(\mathcal{P}) = Res_z(f, \partial_z f)$ where z is the z-coordinate of s
4. Check that $\Pi_z(\mathcal{P})$ is regular in the 2-dimensional sense with a star shaped configuration

6.1.2 Topology

The regularity test ensures that we are able to compute the topology of a surface within a given domain. The way of computing this topology comes together with the regularity test. Again, we distinguish the 2-dimensional, 1-dimensional and 0-dimensional strata configurations of a surface inside a subdivision cell.

6.1.2.1 2-dimensional stratum

If a surface $\mathcal{S}$ is 2-regular in the domain $\mathcal{D}$ then its topology is uniquely determined from information on the border $\partial\mathcal{D}$ of the domain $\mathcal{D}$.

In such a configuration, the polar variety does not appear in $\mathcal{D}$. The implicit functions theorem applies and each patch can be seen as a graph of a function over a planar domain. To each of these patches corresponds a loop for the intersection of the surface with the boundary of $\mathcal{D}$. In order to determine these loops, it is sufficient to analyze the branches and their connections on the facets of $\mathcal{D}$.

Algorithm 6.2: Connection for a 2-dimensional stratum.

Input: A cell $\mathcal{D}$ and a 2-dimensional stratum of $\mathcal{S}$ in $\mathcal{D}$

Output: The boundary $\mathcal{B}$ of the patch of $\mathcal{S} \cap \mathcal{D}$

Compute the set $V = \mathcal{S} \cap \partial\mathcal{D}$ to get the vertices of the patch ;

Apply connection algorithm 5.2 to get the edges of the patch ;

Connect the edges together to get the face defined by the patch ;

6.1.2.2 1-dimensional stratum

If a surface $\mathcal{S}$ is 1-regular in the domain $\mathcal{D}$ then its topology is uniquely determined from information on the border $\partial\mathcal{D}$ of $\mathcal{D}$.

Since there cannot be any isolated point in $\mathcal{D}$ and the polar variety is a regular curve, the two surface patches intersect along the polar variety. For two facets, intersection curves self-intersect at the endpoints of the polar variety curve segment in a star shaped configuration. By connecting these singular points for two facets we obtain the topology of the self-intersection of the two surface patches that we connect with curves of intersection on each facet.

Algorithm 6.3: Connection for a 1-dimensional stratum.

Input: A cell $\mathcal{D}$ and a 1-dimensional stratum of $\mathcal{S}$ in $\mathcal{D}$
Output: The boundary $\mathcal{B}$ of the patch of $\mathcal{S} \cap \mathcal{D}$
Compute the set $V = \mathcal{S} \cap \partial\mathcal{D}$ to get the vertices of the patch ;
Apply connection algorithm 5.2 on regular faces of $\mathcal{D}$;
Apply connection algorithm 5.3 on singular faces of $\mathcal{D}$;
Connect the edges together to get the face defined by the patch ;

6.1.2.3 0-dimensional stratum

If a surface $\mathcal{S}$ is 0-regular in the domain $\mathcal{D}$ then its topology is uniquely determined from information on the border $\partial\mathcal{D}$ of the domain $\mathcal{D}$ and from its unique singular point in $\mathcal{D}$.

As in the 2-dimensional case, the problem is to locally ensure the topology by counting the number of features stemming out from the unique singular point s within the cell. To this end, we compute a projection of the polar variety on the plane of same z-coordinate as s. The latter contains the singular point as well as branches stemming out from it, corresponding to the projection of the silhouette of $\mathcal{S}$. By applying the regularity test of section 5.1.1.2, we ensure the topology within the cell by connecting s to points of interest on the border of the cell.

In some cases, it could be difficult to "see" the polar variety and thus, to apply the regularity criteria on its projection. In such cases, an alternative

consists in locally defining a box of some very small size ϵ, centered around the singular point and counting the number of connected components computed from the current object(s) with its boundary. If this number coincides with the number of features computed for the current cell of subdivision, we ensure a conic structure. This test can be used as a heuristic, choosing an appropriate ϵ depending on the context.

Algorithm 6.4: Connection for a 1-dimensional stratum.

Input: A cell $\mathcal{D}$ and a 0-dimensional stratum of $\mathcal{S}$ in $\mathcal{D}$

Output: The boundary $\mathcal{B}$ of the patch of $\mathcal{S} \cap \mathcal{D}$

Compute the set $V = \mathcal{S} \cap \partial\mathcal{D}$ to get the vertices of the patch ;

Compute the singular point s of $\mathcal{S}$ in $\mathcal{D}$;

Compute the projection of the polar curve $\Pi_z(\mathcal{P}) = Res_z(f, \partial_z f)$ where z is the z-coordinate of s ;

Apply connection algorithm 5.2 on regular faces of $\mathcal{D}$;

Apply connection algorithm 5.3 on singular faces of $\mathcal{D}$;

Connect the edges together to get the face defined by the patch ;

Connect the edges endpoints to s and create incident faces accordingly ;

6.2 Parametric surfaces

As mentioned in chapter 4, the arrangement algorithm driven by a subdivision scheme can be specialized to any kind of curve or surface in any dimension as long as one is able to provide regularity criteria to drive the subdivision process together with the corresponding topology computation algorithm that is used to locally compute regions in the cells of subdivision starting from the topology of the object(s) lying in the cell. We remind that these local regions are then merged following a generic scheme which performs unions of these piecewise entities through a hierarchical data structure.

The topology of a surface patch itself is easy to recover by nature when computing with parameterized models given in evaluation. In particular, rendering this kind of surface is an easy task.

To match the set of configurations shown in figure 6.1, we have to compute: 1. intersection points of the surface patch with the cell edges 2. intersection

curves of the surface patch with the cell faces 3. self-intersection curves of the surface patch 4. singular points of the surface patch. In either case, we want to benefit from the exact nature of the model. This can be achieved by performing the computations in the parameter space of the surface, *i.e.* the domain $\mathcal{D}_p = [s_{smin}, s_{max}] \times [t_{min}, t_{max}]$.

The set of intersection points with the edges of a subdivision cell $\mathcal{D}$ for a polynomial rational surface (defined in section 2.2.2) can easily be obtained by respectively retrieving the sets

$$\begin{aligned} v_{x_i,y_j} &= \{(x(s,t), y(s,t), z(s,t)) \in \mathcal{D} \mid (s,t) \in \mathcal{D}_p, x(u,v) = x_i, y(u,v) = y_i\} \\ v_{x_i,z_j} &= \{(x(s,t), y(s,t), z(s,t)) \in \mathcal{D} \mid (s,t) \in \mathcal{D}_p, x(u,v) = x_i, z(u,v) = z_i\} \\ v_{y_i,z_j} &= \{(x(s,t), y(s,t), z(s,t)) \in \mathcal{D} \mid (s,t) \in \mathcal{D}_p, y(u,v) = y_i, z(u,v) = z_i\} \end{aligned}$$

where $x_i \in \{x_{min}, x_{max}\}$, $y_i \in \{y_{min}, y_{max}\}$ and $z_i \in \{z_{min}, z_{max}\}$.

The same way, an intuitive computation of the intersection curves to get the edges of the patch on each facet of the subdivision cell is:

$$\begin{aligned} e_{x_i} &= \{(y(s,t), z(s,t)) \mid (s,t) \in \mathcal{D}_p, x(s,t) = x_i\} \\ e_{y_i} &= \{(x(s,t), z(s,t)) \mid (s,t) \in \mathcal{D}_p, y(s,t) = y_i\} \\ e_{z_i} &= \{(x(s,t), y(s,t)) \mid (s,t) \in \mathcal{D}_p, z(s,t) = z_i\} \end{aligned}$$

where $x_i \in \{x_{min}, x_{max}\}$, $y_i \in \{y_{min}, y_{max}\}$ and $z_i \in \{z_{min}, z_{max}\}$.

For each facets of the cube, we perform the analysis using the implicit representation of the intersection curves of the surface with the cell in the parameter space of the surface patch. To this end, we use the framework exposed in section 5.3. This representation allows the computation with low degree polynomials.

Remark 6.3: The intersection curves of the surface patch with the border of the subdivision cell $\mathcal{D}$ can also be expressed by the mean of planar implicit curves in euclidean space using resultant computation. Indeed for the faces of coordinates $x_i \in \{x_{min}, x_{max}\}$

$$\begin{cases} x(s,t) - x_i = 0 \\ y(s,t) - y = 0 \\ z(s,t) - z = 0 \end{cases}$$

leads to the system

$$\begin{cases} f(s,t,y,z) = 0 \\ g(s,t,y,z) = 0 \\ h(s,t,y,z) = 0 \end{cases}$$

the resultant $r_x(y,z) = \mathrm{Res}_{s,t}(f,g,h)$ of which gives an implicit curve in the yz-plane. Note that this computation provides high-degree polynomials but seems reasonable for surfaces of bidegree $(2,2)$. •

We define the tangent vectors at $\mathcal{C}$

$$T_u(u,v) = \begin{pmatrix} \partial_u x(u,v) \\ \partial_u y(u,v) \\ \partial_u z(u,v) \end{pmatrix} \quad T_v(u,v) = \begin{pmatrix} \partial_v x(u,v) \\ \partial_v y(u,v) \\ \partial_v z(u,v) \end{pmatrix}$$

and the normal vector

$$N(u,v) = T_u(u,v) \wedge T_v(u,v)$$

The critical points for x (respectively y, z) are expressed by the systems

$$\begin{cases} N_y(u,v) = 0 \\ N_z(u,v) = 0 \end{cases} \quad \begin{cases} N_x(u,v) = 0 \\ N_z(u,v) = 0 \end{cases} \quad \begin{cases} N_x(u,v) = 0 \\ N_y(u,v) = 0 \end{cases}$$

The set of singular points is expressed by the system

$$\begin{cases} N_x(u,v) = 0 \\ N_y(u,v) = 0 \\ N_z(u,v) = 0 \end{cases}$$

The problem is then to compute either intersection curves in a static context or self-intersection curves in a dynamic context. Even though a very fast computation is proposed by Jean Pascal Pavone in [53], we opt for the method of Stephane Chau [25] for the following reasons.

The first method is a sampling algorithm, which, driven by an injectivity criteria ends up in computing the self-intersection curves using an approximation of the shape and a triangle intersection procedure. The result does not provide any topological guarantee but note that is however allows a

real time computation of the self-intersection curve for a result more than sufficient in many CAGD contexts where computations are performed on approximations. For similar reasons, we do not choose the method proposed in [101].

On the contrary, the second method, in its resultant based approach provides an implicit representation of the intersection and self-intersection curves of two parametric surfaces in their parameter space. Using such a method allows us the reuse of our 2-dimension criteria for checking implicit curves for regularity, assuming the fact that we are able to guarantee the result. This is done by recursively checking that the images of the boxes defining the respective parameter spaces do not intersect in the euclidean space.

The regularity test then occurs in the parameter space of the surfaces by considering the curves corresponding to the polar varieties $N_x(u,v)$, $N_y(u,v)$ and $N_z(u,v)$ respectively for each direction x, y and z together with self-intersection or intersection curves. Let us see how to obtain these curves using the second method.

In this computation, we use the resultant to compute the intersection locus between $f(u,v)$ and $g(r,s)$. We consider the algebraic variety

$$\mathcal{C} = \{(u,v,r,s) \mid f(u,v) = g(r,s)\} \tag{6.3}$$

we denote $\mathcal{D}_p$ its parameter space and we will suppose that $\mathcal{C} \cap \mathcal{D}_p$ is a curve.

We remind some basics about resultants. Let f_1, f_2 and f_3 be three polynomials in two variables with given monomial supports and N the number of terms of these 3 supports. For each $i \in \{1,2,3\}$ we denote by $\text{coeffs}(f_i)$ the sequence of the coefficients of f_i. The resultant of f_1, f_2 and f_3 is, by definition, an irreducible polynomial R in N variables with the property, that

$$R(\text{coeffs}(f_1), \text{coeffs}(f_2), \text{coeffs}(f_3)) = 0 \tag{6.4}$$

if and only if these 3 polynomials have a common root in a specified domain $\mathcal{D}$. For a more precise description of resultants, see *e.g.* [19, 26, 27].

In our application to surface–surface–intersection, the resultant can be used as a projection operator. Indeed, if f_1, f_2 and f_3 are the three components of $f(u,v) - g(r,s)$ which are considered as polynomials in the two variables

r and s, then the resultant of f_1, f_2 and f_3 is a polynomial $R(u, v)$ and it gives an implicit plane curve which corresponds to the projection of $\mathcal{C}$ in the (u, v) parameters. More precisely, if f_1, f_2 and f_3 are generic, then the two sets

$$\big\{(u, v) \in [0, 1]^2 \mid R(u, v) = 0\big\} \tag{6.5}$$

and

$$\big\{(u, v) \in [0, 1]^2 \mid \exists (r, s) \in \mathcal{D} : f(u, v) = g(r, s)\big\} \tag{6.6}$$

are identical. Several families of multivariate resultants have been studied and some implementations are available, see [22, 71].

In what follows, we show how to characterize the intersection locus, useful *e.g.* in a dynamic context or the self-intersection, useful *e.g.* in a static context.

Application to the intersection problem. A strategy to describe the intersection between $f(u, v)$ and $g(r, s)$ consists in projecting $\mathcal{C}$ on a plane (by using the resultant). Many authors propose to project $\mathcal{C}$ on the (u, v) (or (r, s)) plane and then the resulted plane curve is traced (see [52] and [69] for the tracing method) and is lifted to the three dimensional space by the corresponding parameterization. Note that this method can give some unwanted components (the so called "phantom components") which are not in $f([0, 1]^2) \cap g([0, 1]^2)$. So, another step is needed to cut off the extraneous branches. This last part can be done with a solver for multivariate polynomial systems (see [80]) or an inversion of parameterization (see [20]).

As an alternative to these existing approaches, we propose to project the set $\mathcal{C}$ onto the (u, r) space. Note that, in the equations $f(u, v) = g(r, s)$, the two variables v and s are separated, so they can be eliminated via a simple resultant computation. It turns out that such a resultant can be computed via the determinant of a Bezoutian matrix (see [44] or [45]). First, consider the $(3, 3)$ determinant:

$$b = \det\left(f(u, v) - g(r, s), \frac{f(u, v) - f(u, v_1)}{v - v_1}, \frac{g(r, s) - g(r, s_1)}{s - s_1}\right) \tag{6.7}$$

The determinant b is a polynomial and its monomial support with respect to (v, s) is $\mathcal{S} = \{1, v, s, vs\}$ and similarly for (v_1, s_1), where $\mathcal{S}_1 =$

$\{1, v_1, s_1, v_1 s_1\}$. So, a monomial of b is a product of an element of $\mathcal{S}$ and of an element of $\mathcal{S}_1$. Then, we form the 4×4 matrix whose entries are the coefficients of b indexed by the product of the two sets $\mathcal{S}$ and $\mathcal{S}_1$. This matrix contains only the variables u and r and is called the Bezoutian matrix. In our case, its determinant is a polynomial in (u, r) equal to the desired resultant $R(u, r)$ ($\deg(R)$=24 and $\deg_u(R)$=$\deg_r(R)$=16) and it gives an implicit curve which corresponds to the projection of $\mathcal{C}$ in the (u, r) space.

Then, we analyze the topology of this curve (see [57] and [90]) and we trace it (see [52] and [69]). Finally, for each $(u_0, r_0) \in [0, 1]^2$ such that $R(u_0, r_0) = 0$, we can determine if there exists a pair $(v_0, s_0) \in [0, 1]^2$ such that $f(u_0, v_0) = g(r_0, s_0)$ (solving a polynomial system of three equations with two separated unknowns of bi-degree (2,2)) and thus we can avoid the problem of the phantom components. We lift the obtained points in the 3D space to give the intersection locus. Note that this method can also give the projection of $\mathcal{C}$ in the (v, s) space by the same kind of computation.

Application to the self-intersection problem. In the parameter domain $\mathcal{D}_p$, the self–intersection curve of the first patch forms the set:

$$\{(u_1, v_1, u_2, v_2) \in \mathcal{D}_p \mid (u_1, v_1) \neq (u_2, v_2) \text{ and } f(u_1, v_1) = f(u_2, v_2)\} \quad (6.8)$$

This locus is the real trace of a complex curve. We assume that it is either empty or of dimension 0 or 1. We do not consider degenerate cases, such as a plane which is covered twice.

We use the following change of coordinates to discard the unwanted trivial component $(u_1, v_1) = (u_2, v_2)$. Let (u_2, v_1) be a pair of parameters in $[0, 1]^2$, $(l, k) \in \mathbb{R}^2$ and let $u_1 = u_2 + l$, $v_2 = v_1 + lk$. If we suppose that we have $(u_1, v_1) \neq (u_2, v_2)$, then $l \neq 0$. Hence $f(u_1, v_1) = f(u_2, v_2)$ if and only if $f(u_2 + l, v_1) = f(u_2, v_1 + lk)$. We suppose now that (u_2, v_1, l, k) verifies this last relation.

Let $\tilde{T}(u_2, v_1, l, k)$ be the polynomial $\frac{1}{l}\left[f(u_2 + l, v_1) - g(u_2, v_2 + lk)\right]$, its degree in (u_2, v_1, l, k) is $(2, 2, 1, 2)$ and the monomial support with respect to

(l, k) contains only $k^2 l, k, l$ and 1. We can decrease the degree introducing:

$$T(u_2, v_1, m, k) = m\tilde{T}(u_2, v_1, \frac{1}{m}, k). \tag{6.9}$$

Then in $T(u_2, v_1, m, k)$, the monomial support in (m, k) consists only of $1, m, k^2$ and km. So, we can write T in a matrix form:

$$T(u_2, v_1, m, k) = \begin{pmatrix} a_1(u_2, v_1) & b_1(u_2, v_1) & c_1(u_2, v_1) & d_1(u_2, v_1) \\ a_2(u_2, v_1) & b_2(u_2, v_1) & c_2(u_2, v_1) & d_2(u_2, v_1) \\ a_3(u_2, v_1) & b_3(u_2, v_1) & c_3(u_2, v_1) & d_3(u_2, v_1) \end{pmatrix} \begin{pmatrix} 1 \\ m \\ k^2 \\ km \end{pmatrix}$$

By Cramer's rule, we get

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

with

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

and

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

Lemma 6.1: Let $Q(u_2, v_1)$ be the polynomial $Q = D_4^2 D_1 - D_2^2 D_3$. The curve $\{(u_2, v_1) \in [0, 1]^2 \mid Q(u_2, v_1) = 0\}$ defined implicitly is the projection of the self–intersection locus (given by the set (6.8) but in $\mathbb{C}^4$) into the parameters domain $(u_2, v_1) \in [0, 1]^2$. ◊

Proof 6.1: $Q(u_2, v_1) = 0$ is the only algebraic relation (of minimal degree) between u_2 and v_1 such that $\forall (u_2, v_1) \in [0, 1]^2, Q(u_2, v_1) = 0 \Rightarrow \exists (m, k) \in \mathbb{C}^2, T(u_2, v_1, m, k) = 0$. □

This lemma provides a method to compute the self–intersection locus. For every point (u_2, v_1) on the curve $Q(u_2, v_1) = 0$, we obtain by continuation the corresponding point $(u_1, v_2) \in [0, 1]^2$ if it exists. So it suffices to characterize the bounds of these curve segments.

6.3 Piecewise linear surfaces

The topology of a piecewise linear surface is provided together with its representation. The only thing we have to do in order to compute an arrangement of such objects is to efficiently carry out intersection and self-intersection tests. Again, to this end, we can build a monotonous segmentation of the surface with respect to a direction.

Once (self-)intersection points and curves have been computed we just have to follow adjacency relationships to deem the cell regular or not. Topology of regions is then trivially obtained from the representation of the surface.

Chapter 7

Applications

This chapter evokes some major applications of the arrangement algorithm. First, the well known problem of defining regions in a trimming process of parametric surfaces can benefit from a static version of the generic subdivision algorithm in the parameter space of the surface. This works either when trimming a surface by another one using an intersection algorithm or when trimming a self-intersecting surface by removing its "bad" regions. Second, we show how the medial curve computation of rational curves can be embedded into an incremental process to dynamically maintain a Voronoi diagram of rational curves, using arrangement computation.

7.1 Trimming of parametric surfaces

In most CAD systems, a trimmed parametric surface is defined by two things: the control information of the surface itself and the control information of its trimming curves, which are usually defined as parametric curves in the parameter space of the surface. Often, trimmed parametric surfaces cause problems in the context of data exchange between CAD systems, surface evaluation/rendering, and grid/mesh generation.

The problem of trimming surfaces has been approached by two distinct ways. The first family of methods consists in computing a piecewise model which union of patches corresponds to the trimmed surface (*e.g.* in [9]). The other

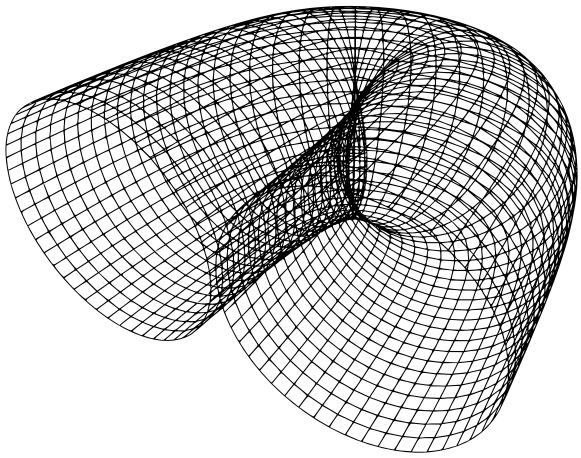


Figure 7.1: A self-intersecting pipe obtained in a CAD process.

major family of methods use ray tracing techniques (possibly exploiting the GPU, [61, 74]) to render the surface resulting from the trimming process.

Trimmed surfaces are now used as a standard tool for modeling complex objects, in various areas, such as computer animation or CAD modeling. Several software systems (Maya, Lightwave, Catia, see chapter 2) have specific tools to define and to render trimmed surfaces. The most popular technique to define a trimmed surface is to introduce a base parametric surface and to specify the trimming area by a closed parametric curve. The orientation of the trimming curve determines its inner and outer parts. The trimming techniques using Nurbs and other parametric objects usually require a re-parameterization step in order to obtain a correct visual result.

Although the trimming technique based on parametric curves is very popular, it has some severe limitations. For instance, the trimming curve can not self-intersect, and trimming area is a simple hole. It means that for every hole one wants to model, he/she needs to define the corresponding trimming curve on the surface. Furthermore, it is well known that set-theoretic operations on B-Rep models and on parametric surfaces suffer from the lack of robustness, and unwanted holes and cracks often appear when performing trimming operations.

Using an arrangement, one can easily remedy these limitations.

Another context in which the problem is even more crucial is the verification

of the output models of CAD constructions such as ones found in software cited above. Let us consider for example the pipe shown in figure 7.1, obtained by extruding a base curve along a path. To be able to use this model in a simulation or as a part of a design process that could serve as an input for a CAM system, it is very important to check whether the model self-intersects or not. If so, identifying and removing the wrong regions created by the self-intersection is important.

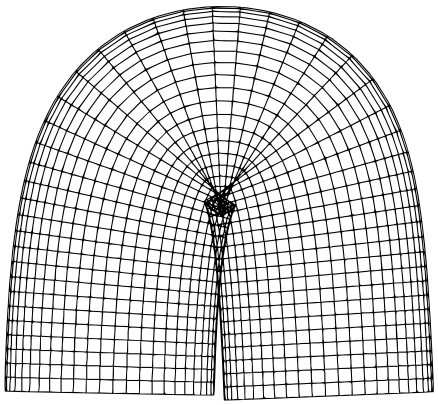
Again, using an arrangement, one can remedy this situation.

Figure 7.3b shows the self intersection curves in the parameter space of our pipe, which top view is shown in figure 7.3a.

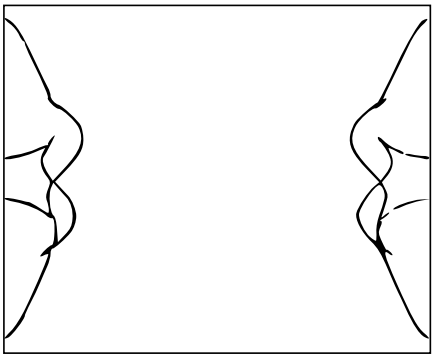
Let us outline the method which allows to compute the trimmed regions. This is an adaptation of the algorithm 4.1 to handle the verification of the mapping of the regions in euclidean space.

The arrangement is run in the parameter space of the parametric surface and objects are self-intersection curves which can have any representation among implicit, parametric or piecewise linear.

Remark 7.1: Note that thanks to the genericity of our subdivision arrangement method, one can obtain the regions whatever the technique used to get the self-intersection curves. •



(a) Computing the self-intersection curves as implicit curves in parameter space.



(b) The (intersecting) self-intersection curves can be dealt with in the parameter space using an arrangement.

Figure 7.2: Trimming self-intersection parts of a parametric surface using arrangements.

Remark 7.2: Since self-intersection curves of an object or intersection curves of two objects that define the trimming are computed for once and for all, there is no need to maintain the arrangement structure dynamically. The static version of the arrangement algorithm is therefore completely adapted.•

Algorithm 7.1: A generic subdivision algorithm.

Input: A set of curves $\mathcal{C}$ and the parameter space $\mathcal{D}_0 \subset \mathbb{R}^2$.

Output: a list of regions defined by $\mathcal{C}$.

Create a quadtree $\mathcal{Q}$ and set its root to $\mathcal{D}_0$;

Create a list of cells $\mathcal{L}$ and initialize it with $[\mathcal{D}_0]$;

while $\mathcal{L} \neq \emptyset$ **do**

  $c =$ `pop(`$\mathcal{L}$`)` ;

  if `regular(`$\mathcal{C}, c$`)` *and* `map_regular(`$\mathcal{C}, c$`)` **then**

    $\mathcal{Q} \leftarrow$ `topology(`$\mathcal{O}, c$`)` ;

  else

    $\mathcal{L} \leftarrow$ `subdivide(`$\mathcal{O}, c$`)` ;

  end

end

return `fusion(`$\mathcal{Q}$`)` ;

The regularity test of algorithm 7.1 is enriched with another one which has different meaning depending on whether the trimming process implies one or two objects:

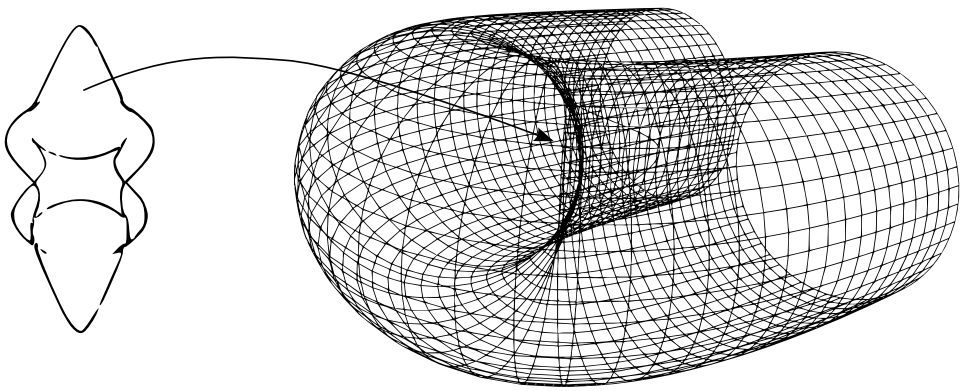


Figure 7.3: Computing the arrangement in parameter space controlling the mapping of defined regions allows to obtain trimmed regions even with self-intersection curves in degenerate configuration.

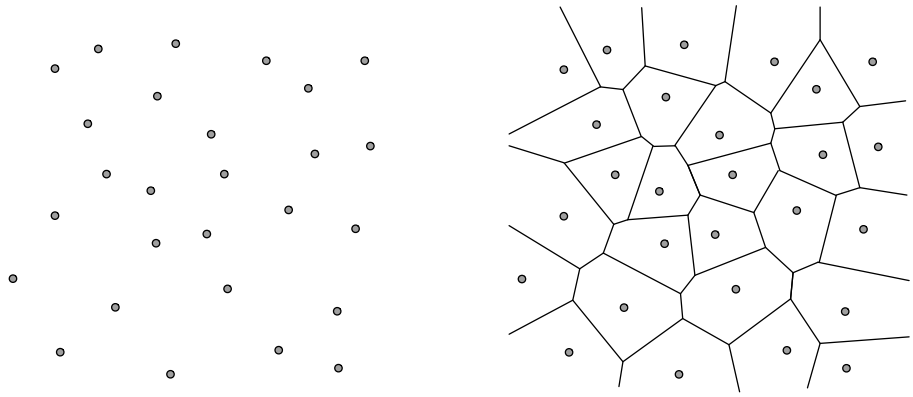
- In the case of a self-intersection process, this additional regularity test is optional.
- In the case of an intersection process, this additional regularity test checks whether the images $s_1(c)$ and $s_2(c)$ of the subdivision cell c by the maps defining surfaces S_1 and S_2 do not intersect each other.

Whenever these tests are fulfilled and the curves are regular as defined in section 5.1.1, we obtain the regions defined by the (self-)intersection which can be queried by various means (such as point location) and is exact by exploiting the map whenever a query is performed.

7.2 Voronoi diagram of rational curves

Computing the Voronoi diagram of a set of objects is a famous problem with many applications, especially in motion planning. If the problem is well handled when the input is a set of points (called *Voronoi sites*, see figure 7.4), computing the diagram of a set of curves causes many problems, overviewed in [7].

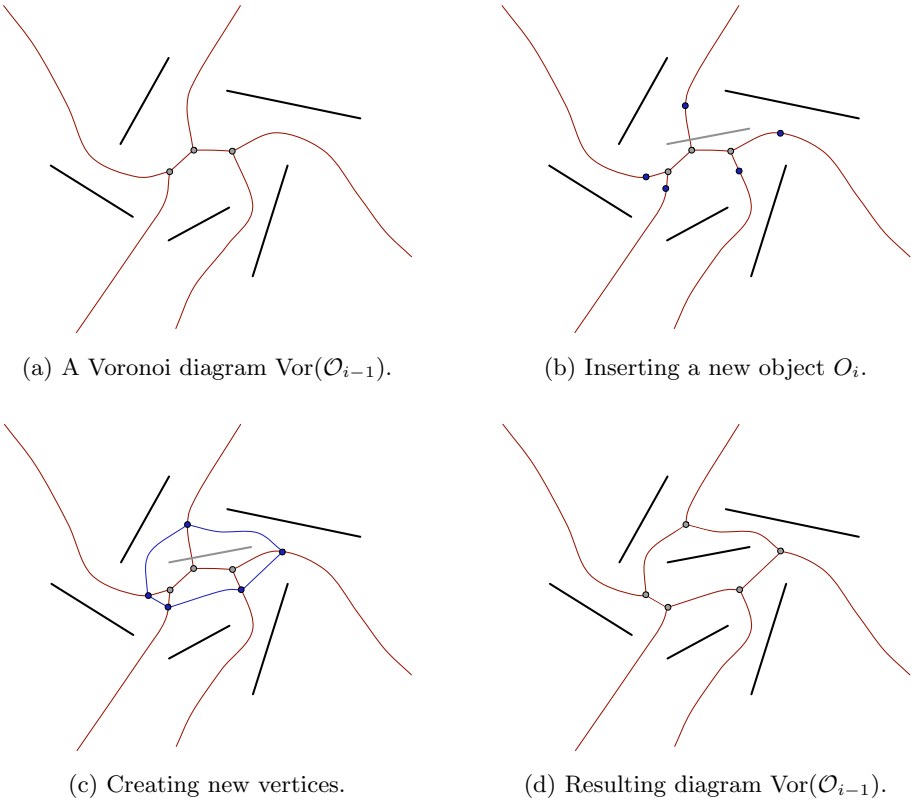
Given a set $\mathcal{O}$ of n objects, the associated Voronoi diagram subdivides $\mathbb{R}^d$ into regions, each region consisting of the points that are closest to one of the objects. The set of all interior points that have equal minimum distance



(a) Set of input points $\mathcal{P}$.

(b) The Voronoi diagram Vor($\mathcal{P}$).

Figure 7.4: The Voronoi diagram of a set of points.



(a) A Voronoi diagram Vor($\mathcal{O}_{i-1}$).

(b) Inserting a new object O_i.

(c) Creating new vertices.

(d) Resulting diagram Vor($\mathcal{O}_{i-1}$).

Figure 7.5: Incremental construction of a Voronoi diagram of line segments.

from at least two objects is called the skeleton Skel($\mathcal{O}$) and the Voronoi diagram is denoted Vor($\mathcal{O}$).

Incremental algorithms (see discussion in chapter 3) can be used to dynamically maintain the Voronoi diagram of a set of objects. When adding a new object O_i into a structure Vor($\mathcal{O}_i$) (figure 7.5(b)), the algorithm basically behaves as follows: 1. Compute the conflict skeleton (see figure 7.5(c)). 2. Create a new vertex at each point of the conflict skeleton. 3. Remove vertices, edges and portions of edges that belong to Skel($\mathcal{O}_{i-1}$) $\cap$ Vor($O_i, \mathcal{O}_i$). 4. Connect the new vertices as to form the boundary of the new region (see figure 7.5(d)).

Affine diagrams, *i.e.* diagrams whose cells are convex polygons, are well understood. The situation is very different for curved Voronoi diagrams, which arise in various contexts where the objects are not punctual and/or

the distance is not the Euclidean distance.

The distance metric used to compute curved diagram is usually a medial object computation. The problem of bisector curves has been studied in [39, 40, 41, 42, 47] (see figure 7.6).

Computing the diagram of non trivial input curved objects implies to (1) efficiently carry out many operations such as computing the bisector curve of two input objects (2) finding all intersections between two input objects. For (1), we rely on the formulas given by Elber in [40]. For (2), we solve systems of arbitrary polynomial equation [80].

Let us overview how to use the dynamic subdivision arrangement algorithm in order to compute a Voronoi diagram of a set of rational curves.

1. Each time an object O_k is inserted, we retrieve the set of regions which conflicts with O_k. This is done by traversing the current augmented influence graph, checking for conflicts each node using the segmentation structure associated to the region contained in the node. This yields a set of conflicting regions $\{R_i\}$.

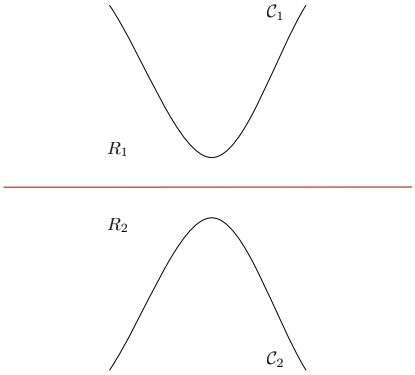
2. For each site O_i of which R_i is the corresponding region, we compute the medial curve M_i of O_k and O_i and intersect it with R_i (that is, we consider the half-part of M_i which lies in R_i). M_i is an implicit curve in the parameter space of O_k and O_i. We insert M_i into the current arrangement, yielding the new region R_k^i. Note that this insertion reduces R_i ($R_i = R_i - R_k^i$).

3. Finally, we merge the set of regions created by O_k to get the region corresponding to this site ($R_k = \bigcup R_k^i$).

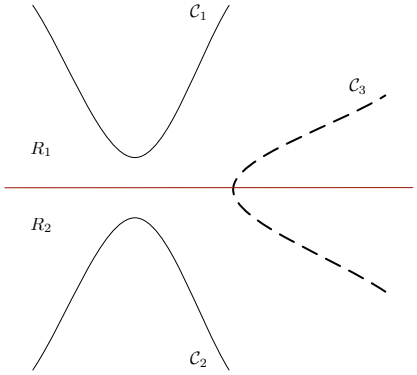
Example 7.1: Figure 7.6(a) shows the medial curve of two curves $\mathcal{C}_1$ and $\mathcal{C}_2$. It divides the space into two regions R_1 and R_2. If $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) is a Voronoi site, R_1 (resp. R_2) is the corresponding region. The Voronoi diagram corresponding to the set of objects $\mathcal{O}_2$ with chronological sequence $\Sigma = \{\mathcal{C}_1, \mathcal{C}_2\}$ is denoted $\mathrm{Vor}(\mathcal{O}_2)$.

1. $\mathcal{C}_3$ is inserted into $\mathrm{Vor}(\mathcal{O}_2)$. The set of regions conflicting with it is $\{R_1, R_2\}$ (see figure 7.6(b)).

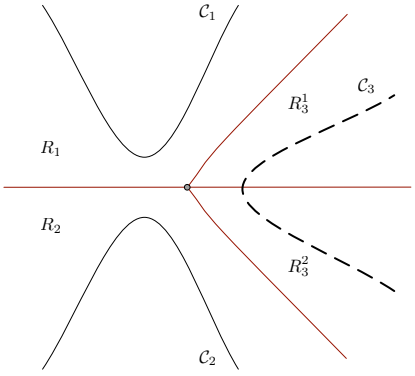
2. The medial curve of $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) and $\mathcal{C}_3$ is computed and intersected



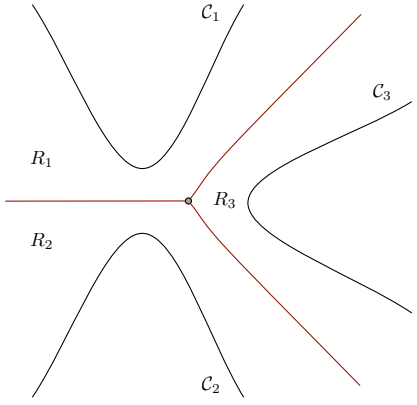
(a) The medial curve of two rational curves defines two regions.



(b) Inserting a new object in the diagram: retrieve regions in conflict.



(c) Compute corresponding medial curves.



(d) Maintain the arrangement of medial curves.

Figure 7.6: A step in the dynamic computation of a Voronoi diagram of rational curves.

with R_1 (resp. R_2). The yields the upper (resp. lower) half curve M_1 (resp. M_2) in figure 7.6(c). M_1 (resp. M_2) is inserted in the current arrangement to obtain the region R_3^1 (resp. R_3^2) (see figure 7.6(c)).

3. R_3^1 and R_3^2 are merged to get R_3, the Voronoi region defined by $\mathcal{C}_3$. ∎

Remark 7.3: [64] addresses the problem of computing a Voronoi cell of rational planar curves. It contains in particular some information for computing the intersection of medial curves by splitting them into monotone pieces, applying trimming constraints and a lower envelope computation. •

Chapter 8

Axel algebraic geometric modeler

Axel [108] is an algebraic geometric modeler which allows the visualization and manipulation of geometric objects with algebraic representation such as implicit or parametric curves or surfaces.

Its main features are topology, interpolation, approximation, intersection, self-intersection and arrangement computation of implicit and parametric curves and surfaces.

We expose objects and tools both from a user point of view, presenting the software graphical, script and file interfaces, and from a developer point of view, presenting internal hierarchies of data structures used to achieve both genericity and efficiency, as well as the plugin system which allows anyone to extend the software's capabilities or wrap an external library to use the application as its graphical frontend.

Current geometric modelers focus on discrete representations which present many advantages: they are easy to understand, easy to manipulate, easy to render. For all these reasons, meshes are models of choice for many users from beginners to confirmed designers and if they are suitable in many situations they present one major weakness: they are approximate and the amount of data is closely related to the desired accuracy.

On the other hand, industry or researchers use non linear geometric models to represent physical shapes or phenomena. Therefore, models need a more accurate representation which implies a computational difficulty but a re-

duced amount of data. Implicit or parametric curves and surfaces are such geometric objects with a very compact representation.

Computer Algebra Systems (CAS) have an obvious need of visualization facilities for such objects. Even though some existing solutions allow to display mathematical objects, they do not propose an interaction with it, they have no edition support. Maple^{TM} and Mathematica^{TM} are such applications with built in interactive plotting functionalities but they give unsatisfying results mainly for singular points or singular curves. Singular [59] is dedicated to polynomial computations with special emphasize on the needs of handling singularities. It has a connection with Surf which provides static images and lacks interaction. We finally mention [51], one of the first independent graphic tool aimed at being driven by an external CAS for the visualization of mathematical surfaces.

The visualization of geometric objects having mathematical representation such as curves or surfaces has been investigated for many years in many research areas such as computational or algebraic geometry. Existing methods usually depend on the nature of objects: while parametric models may be evaluated on grids, the visualization of implicit surfaces has led to two types of solutions. The most widespread one is the ray tracing method [32, 73] which is convenient in the case of implicit curves or surfaces. The principal inconvenient is that produced images are static, *i.e.* they are computed for a given viewpoint and changing either position or orientation of the object or of the user implies to recompute a picture. The other solution consists in computing a piecewise linear approximation of the object and then to dynamically display it. While marching cube methods [79, 91] usually are time consuming and may produce uncertified results, new techniques guarantee the topological structure of curves and surfaces even in singular cases [4, 5, 35, 57].

Axel aims at providing a unified framework for both the visualization and manipulation of geometric objects with algebraic representation, manipulating exact data to provide certified results even when the context requires an approximation.

The application is designed to be either used standalone or connected together with external tools and therefore become a graphical frontend. Its

modular architecture using plugins makes easy the wrapping of external libraries or the connection with other tools.

Following sections show the constant interaction between the geometric entity and its algebraic representation which is dynamically manipulated in the modeling environment.

8.1 User perspective

This section first overviews objects and tools provided in Axel. Then, it shows how the user can interact with them through the graphical, script or file interfaces.

8.1.1 Objects

Among elementary geometric entities such as points, planes *etc.* Axel aims at providing an interface for the visualization and manipulation of curves and surfaces with various representations: implicit, parametric or piecewise linear.

8.1.1.1 Curves

Curves in two or three dimensions are widely used in computational mathematics and geometric modeling either to build surfaces or to model surface sections. They are the main ingredient of technical drawing software.

Implicit curves. Let $f(x, y)$ be a polynomial in two variables with coefficients in $\mathbb{Q}$. If f is non-zero, then the real zero set of f is an algebraic variety of dimension one or zero (a plane curve or isolated point(s)). Its visualization is not straight-forward because of the difficulty to compute a set of points (x_i, y_i) such that $f(x_i, y_i) = 0$. Moreover, detecting and correctly handling singularities (*i.e.* points (x_0, y_0) such that $f(x_0, y_0) = \partial_x f(x_0, y_0) = \partial_y f(x_0, y_0) = 0$) represent a higher difficulty. For all these reasons, the small amount of software used by mathematicians for the visualization of implicit planar curves give unsatisfied results (especially in the