Université de Montréal

Reliable Computation for Geometric Models

par
Di Jiang

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures et postdoctorales
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

Août 2008

Université de Montréal

Faculté des études supérieures et postdoctorales

Cette thèse intitulée :

Reliable Computation for Geometric Models

présentée par :

Di Jiang

a été évaluée par un jury composé des personnes suivantes :

Victor Ostromoukhov
président-rapporteur

Neil Stewart
directeur de recherche

Pierre Poulin
membre du jury

Wolfram Luther
examinateur externe

Abraham Broer
représentant du doyen de la FES

Résumé

La modélisation géométrique est devenue un domaine de recherche et de développement central à un vaste champ d'applications. Avec la forte croissance de la puissance de calcul des ordinateurs, la simulation par ordinateur a commencé à jouer un rôle important dans plusieurs domaines de recherche reliés à la modélisation géométrique, de l'ingénierie traditionnelle à la simulation de chirurgie virtuelle.

À cause de l'usage de représentations de précision finie, l'absence de robustesse numérique en calcul scientifique est un phénomène bien connu et répandu. De nombreuses approches différentes ont été proposées pour résoudre ce problème. Les nombres en virgule flottante (IEEE 754/854) [PH98, Ove01] sont les substituts standards pour les nombres réels en calculs informatisés, et la plupart des logiciels de modélisation de solides, incluant les systèmes de conception assistée par ordinateur (CAO), sont basés sur des méthodes de modélisation géométrique qui fonctionnent en utilisant l'arithmétique en virgule flottante. Mais cette dernière, appliquée naïvement, peut causer l'échec d'axiomes géométriques. L'analyse inverse d'erreur (*backward error analysis*), maintenant standard, est un outil très utile qui peut nous aider à surmonter ce problème : elle nous permet de distinguer les algorithmes qui, en présence d'incertitudes dans les données, ont produit des résultats aussi bien que nous pouvions espérer.

L'impact de l'absence de robustesse dans le domaine de la modélisation géométrique a été ouvertement reconnu et il y a eu beaucoup d'attention pour améliorer la fiabilité. D'un autre côté, il existe plusieurs représentations en modélisation géométrique et, même si chacune parvient à bien modéliser certaines propriétés, aucune d'elles n'est suffisamment générale pour satisfaire tous les prérequis qui pourraient être souhaitables d'une représentation. Ainsi, pour des problèmes géométriques différents, l'absence de robustesse tend à se manifester de différentes façons et nous devons chercher la méthode appropriée pour chaque problème : une solution universelle n'existe pas.

Le but de cette thèse est d'étudier le calcul informatisé fiable en modélisation géométrique. En particulier, nous abordons trois problèmes reliés à la robustesse en modélisation géométrique :

1. *L'arithmétique en virgule flottante pour des problèmes de géométrie informatique avec des données incertaines (*Floating-point arithmetic for computational-geometry problems with uncertain data*).*

   Dans ce travail, trois exemples (résolution de systèmes d'équations linéaires, le problème de l'enveloppe convexe planaire et un problème d'objet extrudé en trois dimensions) sont présentés pour expliquer notre méthode pour accomplir l'analyse inverse d'erreur. Aussi, notre exposition illustre le fait que l'analyse inverse d'erreur ne prétend pas surmonter le problème de précision finie, et que des situations en géométrie informatique sont exactement parallèles à d'autres domaines informatiques.

2. *Jonction fiable de surfaces pour des modèles combinant maillages et surfaces paramétriques (*Reliable joining of surfaces for combined mesh-surface models*).*

   L'opérateur de jonction est un important opérateur primitif pour les opérations booléennes. Notre motivation pour ce travail est de chercher un algorithme de jonction fiable pour les *patches* combinant maillages et surfaces paramétriques, prenant en considération un critère d'erreur sur la normale. Deux mesures d'erreur sont définies pour guider la procédure de jonction. En utilisant le théorème de l'extension de Whitney, la qualité de la jonction calculée peut être garantie.

3. *Robustesse d'opérations booléennes sur les modèles de surface de subdivision (*Robustness of boolean operations on subdivision-surface models.*)*

   Les surfaces de subdivision sont de plus en plus fréquemment utilisées comme représentation de rechange, à la place des surfaces B-splines rationnelles non uniformes coupées (*trimmed NURBS*), pour la modélisation géométrique dû à leurs avantages intrinsèques. En particulier, elles permettent d'éviter le problème difficile de faire correspondre les bordures des *patches* coupées. Ce travail décrit un algorithme pour effectuer des opérations booléennes, basé sur l'usage des maillages limites, dans le cas où les objets en entrée sont définis en termes de maillages triangulaires et de subdivision de Loop. Ce travail se concentre sur la robustesse, incluant des bornes d'erreurs et des méthodes numériques pour la validation *a posteriori* de la forme topologique.

Mots-clés :
calcul informatisé fiable, arithmétique en virgule flottante, robustesse, stabilité, analyse inverse

d’erreur, maillage de surfaces, jonction, opération booléenne, modèles d’interrogation de forme, erreur de vecteurs normaux, surfaces de subdivision.

Abstract

Geometric modeling has become a central area of research and development that involves diverse applications. In fact, because of greatly increased computer power, computer simulation has started playing an important role in many geometric-modeling related research domains, from traditional engineering design to virtual surgery simulation.

Due to the use of finite-precision representation, numerical nonrobustness in scientific computing is a well-known and widespread phenomenon. Several different approaches have been proposed for this problem. Floating-point numbers (IEEE 754/854) [PH98, Ove01] are the standard substitute for real numbers in computations, and most solid modelers, including CAD (Computer Aided Design) systems, are based on geometric-modeling methods that operate using floating-point arithmetic. But naively applied floating-point arithmetic can cause axioms of geometry to fail. The now-standard backward error analysis is a very useful tool that can help to overcome this problem: it permits us to distinguish those algorithms which, given the presence of uncertainties in the data, have done as well as we can hope for.

The impact of nonrobustness in the domain of geometric modeling has been widely acknowledged, and much attention has been paid to improving reliability. On the other hand, many different geometric modeling representations exist, and although each succeeds in modeling certain properties well, none of them is general enough to satisfy all the requirements that could be demanded of a representation. Therefore, for different geometric problems, nonrobustness tends to manifest itself in different ways, and we must seek an appropriate method for each problem: a universal solution does not exist.

The goal of this thesis is to study reliable computation for geometric models. More specifically, we will address three related robustness problems in geometric modeling:

1. *Floating-point arithmetic for computational-geometry problems with uncertain data.*

   In this work three examples (solving linear equations, the planar convex-hull problem

and a three-dimensional extruded-objects problem) are presented to explain our method of performing backward error analysis. Also, our exposition illustrates the fact that backward error analysis does not pretend to overcome the problem of finite precision, and that situations in computational geometry are exactly parallel to other computational areas.

2. *Reliable joining of surfaces for combined mesh-surface models.*

   The joining operator is a very important primitive operator for Boolean operations. Our motivation for this work is to seek a reliable joining algorithm for combined mesh-surface patches, taking into account a normal error criterion. Two error measures are defined to guide the joining procedure. By using the Whitney extension theorem, the quality of the computed joining result can be guaranteed.

3. *Robustness of Boolean operations on subdivision-surface models.*

   Subdivision surfaces are more and more frequently used as an alternative representation, in place of trimmed-NURBS, for geometric modeling due to their intrinsic advantages. In particular, they permit us to avoid the difficulties in matching boundaries of trimmed patches. This work describes an algorithm to perform Boolean operations, based on the use of limit meshes, in the case when input objects are defined in terms of triangular meshes and Loop subdivision. The focus of the work is on robustness, including error bounds and numerical methods for the *a posteriori* validation of topological form of the produced result.

Keywords:
reliable computing, floating-point arithmetic, robustness, stability, backward error analysis, surface mesh, joining, Boolean operation, shape-interrogation models, normal-vector error, subdivision surfaces.

Table of Contents

List of Figures

Acknowledgment

I would like to express my sincere appreciation to Professor Neil F. Stewart, my research supervisor, for his invaluable guidance, his encouragement and his great support throughout all the course of the work. Without him this work would not be possible.

I am particularly grateful to Professor Wolfram Luther for accepting to be my external examiner. My thanks also goes to Professor Pierre Poulin and Professor Victor Ostromoukhov for accepting to be the jury members for this thesis.

I wish to to thank also all the people in the Laboratoire d'Informatique Graphique de l'Université de Montréal (LIGUM) for the help they offered, the wonderful environment they provided, and all the activities we have done together.

Finally, I wish to express my special acknowledgment to my parents, for their support and encouragement, and my husband, François Duranleau, for his support, his encouragement, his comprehension and his company throughout the years of this work.

Preface

This Ph.D. thesis is a thesis by articles. The main part of the thesis is composed of three accepted (to appear), published, and submitted articles. To better present each individual work, we choose to retain for each paper the complete version as it is (will be) in the respective publication. This leads us to two referencing systems in the thesis. For each paper (Ch. 3, 4, 5), its own references are provided together with the paper: each reference entry is assigned a running number in square brackets as the in-text marker (e.g. [1]). Also, a bibliography chapter is given at the end of the thesis, and in this case the reference markers are an abbreviation of the authors' name plus year of publication (e.g. [ABA02]). This is the format for the references for all the other chapters in the thesis. There are certain overlaps between the bibliography chapter and the references of the three individual articles. Another remark about the bibliography chapter is that the references for websites are given in lowercase, e.g. [g-b].

Chapter 1

Introduction

With the greatly improved computational techniques and the powerful machines available, computer-aided methods have come to be involved in almost every aspect of life:

"*Physicists use computers to solve complicated equations modeling everything from the expansion of the universe to the microstructure of the atom, and to test their theories against experimental data. Chemists and biologists use computers to determine the molecular structure of proteins. Medical researchers use computers for imaging techniques and for the statistical analysis of experimental and clinical observations. Atmospheric scientists use numerical computing to process huge quantities of data and to solve equations to predict the weather. Electronics engineers design ever faster, smaller, and more reliable computers using numerical simulation of electronic circuits. Modern airplane and spacecraft design depends heavily on computer modeling...*" [Ove01]

In fact, all fields of science and engineering now rely heavily on numerical computation. But one question has inevitably to be asked: *can we trust these numerical computational results*? We do not want our surgery simulation software to turn out to be a source for medical accidents [FGG03]. The following example gives an idea of how bad things can get if not enough attention is given to verification of correctness. Figure 1.1 shows the result of an implemented algorithm for a simple planar convex hull problem[1]. The point on the lower left corner which clearly belongs to the convex hull has been ignored, and left outside of the resulting hull. The cause of this failure is the naive use of floating point arithmetic on a two-dimensional orientation predicate.

[1]The *convex hull* of a finite point set S in the plane is the smallest polygon containing the set and such that the vertices of the polygon are points of S [KS86].

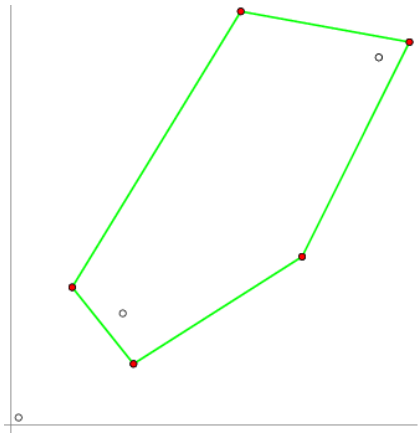


Figure 1.1: An example of failed convex-hull algorithm due to the naive use of floating-point arithmetic [KMP+04].

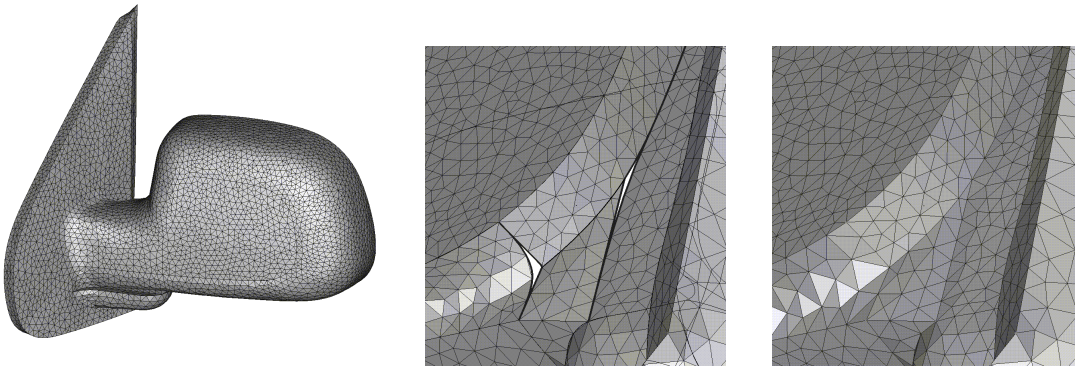
Fortunately, numerical non-robustness in scientific computing is a widely recognized phenomenon. In particular, the goal of reliable computation has attracted many researchers in the area of geometric modeling.

Two main factors, amongst others, explain the origins of the errors that contribute to nonrobustness: the use of floating-point arithmetic and uncertainty in the input data. Often, designers of geometric algorithms avoid the problem of computational error by assuming the real random access machine (RAM) as the model of computation [PS85]. The real RAM allows real numbers to be represented exactly and provides exact arithmetic operations. Unfortunately, often floating-point arithmetic is substituted for exact real arithmetic and special cases are ignored [For93]. Naively applied floating-point arithmetic can cause disastrous results, as illustrated in the previous example (Figure 1.1).

Backward error analysis has become a standard error-analysis method. In the presence of uncertainties in the input data, which is the usual case, it can help to distinguish algorithms that overcome the error problem *to whatever extent it is possible to do so*. In such situations expensive methods, such as exact arithmetic, are not necessary, provided a stable algorithm has been applied. The application of the backward error analysis will be presented in Ch. 3, with detailed examples provided.

For different geometric problems, non-robustness manifests itself in different manners. The phenomena include random system crashes, inconsistent states (e.g. the geometric data inconsistent with the topological data), models that contain cracks, holes and overlaps, etc. [Yap01]. This, in turn, means that we have to seek appropriate methods for each problem: a universal solution does not exist. The following example (Fig. 1.2) is a typical “dirty” geometric model:

an exterior-mirror model with small cracks (left), with the zoom-in on the problematic area (middle) and the repaired result (right) [SWC00].



original model zoom-in of the problem area repaired result

Figure 1.2: An example of a "dirty" geometric model [SWC00].

Depending on the underlying geometric representation used for describing the model, different techniques can be used to eliminate the error in the result, each with its own advantages and weaknesses.

NURBS (details in Ch. 2) have become a *de facto* industry standard for the representation, design, and data exchange of geometric information processed by computer. Trimmed-NURBS (details in Ch. 2) offer greater flexibility than traditional NURBS for the design of very sophisticated objects, and they have become a very powerful tool used in most commercial modeling systems. The errors illustrated in Fig. 1.2 may come from inconsistent information, e.g. trimmed-curve mismatch problems. On the other hand, in most cases, for the purpose of rendering, the trimmed patches need to be transformed into a polygonal representation. The error, at this stage, may come from the approximation procedure, and a joining (sewing/merging) operation can be used to fix the problem. But even in the case that maximum auxiliary information is available, *i.e.* even if we have both trimmed-NURBS and the (triangular) mesh information, a simple joining operation may not produce a satisfying result. Discussion of this problem will be presented in Ch. 4, where an algorithm, which produces a result satisfying two error criteria by using the Whitney extension theorem, will be presented.

NURBS information is not always available in practical applications, e.g. finite-element analysis. Further, a simple polygonal representation (polygon soup) itself is often insufficient for the manipulation of complex geometric models. Therefore, subdivision-surface models (details in Ch. 2) become a convenient representation. In fact, with the increasing popularity of subdivision-surface models, more and more modelers have begun to use them as an alterna-

tive to trimmed-NURBS, due to their simplicity, generality and efficiency for smooth surface construction [BK04]. Subdivision-surface models do not have the trimming difficulties and the error-prone conversion procedure (from trimmed-NURBS to polygonal meshes) associated with NURBS. Complex models based on subdivision surfaces can be formed using Boolean operations. The related robustness issues of such Boolean operations on subdivision-surface models is the next problem we considered (Ch. 5).

The remainder of the thesis is organized as follows. A short overview of the research area of geometric modeling is given in Ch. 2. It contains two parts: reliable computation (sources of error and error analysis methods), and geometric modeling, which presents the geometric representations, geometric operations and the related robustness issues. The main part of the thesis (Ch. 3, 4, 5) is composed of three accepted (to appear), published or submitted articles, each of which forms an individual chapter, with a preceding short summary. Chapter 3 describes our work on floating-point arithmetic for computational-geometry problems with uncertain data. Our work on reliable joining of surfaces for combined mesh-surface models is given in Chapter 4. Chapter 5 discusses the problem of robustness of Boolean operations on subdivision-surface models. We conclude in Chapter 6, where we also mention promising possibilities for future work.

Chapter 2

Reliable computation and geometric modeling

Problems of robustness are a major cause for concern in the implementation of algorithms relating to geometry. Most geometric algorithms are a mix of numerical and combinatorial computations, and the approximate nature of the former often leads to inconsistencies that hinder the ability to construct a satisfactory result [Hof89]. In this chapter an overview of the problems of reliable computation for geometric models will be given, and the related geometric modeling topics, including geometric representations and Boolean operations, will also be presented.

2.1 Finite precision representation and reliable computation

Numerical nonrobustness in scientific computing is a well-known and widespread phenomenon. The root cause is the use of *finite-precision numbers*, e.g. floating-point representation, to represent real numbers, with precision usually fixed by the machine word size (e.g. 24 bits). A number of approaches to the finite-precision problem have been advocated in academia. Hoffmann [Hof01] categorizes these into three strategies: exact arithmetic, symbolic reasoning and interval computation. Exact arithmetic is very expensive, and performance can be badly affected if it is used exclusively, so filtered exact arithmetic is usually preferred [SD07]. Another proposed method related to exact arithmetic is the exact geometric computation [Yap06]. Interval arithmetic [AH83, Moo66, MB79, Sch99, EL00, DS88] treats a rounded real number as an interval and the calculations are performed on this interval — but the shortcoming of interval

arithmetic is that it gives overly pessimistic results. Symbolic manipulation is a possible way to avoid rounding and truncation errors. Thus using software such as Mathematica or Maple may be appropriate, but in many application cases, this might not be the best choice for efficiency reasons. Another approach proposed by Yap [Yap01] is exact geometric computation, which again uses approximate arithmetic, but with the level of precision guided by geometric exactness. A fifth possibility [HS05, ASZ07] is to use ordinary floating-point arithmetic, and to try to associate the error with the input data. This is appropriate if there is uncertainty in the input. It is the last mentioned approach that is studied in this work.

2.1.1 Floating-point number system

Floating-point numbers (IEEE 754/854) are the standard substitute for real numbers in scientific computation [Ove01]. Current state-of-the-art CAD (Computer Aided Design) systems used to create and interrogate curved objects are based on geometric solid modeling methods that typically operate using floating-point arithmetic [PM02, PH98, g-L].

A floating-point number system $\mathbf{F} \subset \mathbb{R}$ is a subset of the real numbers whose elements have the form [Hig96, p.40]:

$$y = \pm m \times \beta^{e-t}.$$

The system $\mathbf{F}$ is characterized by four integer parameters

- the *base* β (sometimes called the *radix*),
- the precision t, and
- the exponent range $e_{min} \leq e \leq e_{max}$.

The *mantissa* m is an integer satisfying $0 \leq m \leq \beta^t - 1$. To ensure a unique representation for each $y \in \mathbf{F}$ it is assumed that $m \geq \beta^{t-1}$ if $y \neq 0$, so that the system is normalized. The *range* of the nonzero floating-point numbers in $\mathbf{F}$ is given by $\beta^{e_{min}-1} \leq |y| \leq \beta^{e_{max}}(1 - \beta^{-t})$.

The IEEE standard 754/854 for floating-point arithmetic requires that the results of $+$, $-$, $\cdot$, $/$, and $\sqrt{}$ are exactly rounded, *i.e.* the result is the exact result according to the chosen rounding mode. It also specifies floating-point computation in single, single-extended, double, and double-extended precisions. Single precision is specified for a 32 bit word, double precision for two consecutive 32 bit words. In single precision the mantissa length is 24 (including a hidden leading 1 bit) and the exponent range is $[-126, 127]$. Double precision has mantissa length 53 and exponent range $[-1022, 1023]$ [Sch99, FGG03].

Floating-point arithmetic has numerous engineering advantages: it is well-supported by programming languages, it is portable, it has useful features such as automatic scaling, and it has been extensively optimized in current computer hardware [For95].

Since an infinite set of numbers is represented by only finitely many floating-point numbers, truncation/rounding techniques have to be used for real numerical values to fit the representation format. Consequently, floating-point computation is, by nature, inexact, and concepts such as representation range, precision and round-off error then arise. Naively applied floating-point arithmetic can invalidate axioms of geometry [Sch99]. The paper [Gol91] and the book by Overton [Ove01] are excellent references for this subject.

2.1.2 Sources of errors

There are three main sources of errors in numerical computation: rounding and truncation due to the finite-precision representation in computer, and data uncertainty [Hig96]. In practice, the input data is often not exact to start with for many applications [Hof89]. Uncertainty may arise in several ways: from error in measuring physical quantities, from errors in storing the data on the computer (truncation errors), or, if the data is itself the solution to another problem, it may be the result of errors in an earlier computation [Hig96]. Another source, additional to the three mentioned, is approximation error, which occurs often in the domain of geometric modeling for practical reasons. One example for this kind of error is the use of low-degree curves to approximate high-degree curves.

2.1.3 Error analysis

The unpredictability of floating-point code across architectural platforms in the 1970's and 1980's was resolved through a general adoption of the IEEE standard 754-1985, later enlarged as IEEE standard 854-1987 [Ove01]. But these standards only make program behavior predictable and consistent across platforms; the errors are still present. Ad hoc methods for fixing these errors (such as treating numbers smaller than some positive ϵ as zero) cannot guarantee their elimination [Yap04]. And since geometric operations usually require extensive numerical calculations, the propagation of the errors is of great concern and profoundly influences the accuracy and validity of the geometric operations [Hof89, MP07]. Therefore, error analysis became very necessary for reliable computation.

Backward error analysis was first proposed by Wilkinson [Wil60] to bound the errors re-

sulting from the fundamental floating-point arithmetic operations [PM02], especially addition of quantities of opposite sign and approximately equal magnitude: the computed result can be completely wrong due to a simple cancellation (see examples in the paper that follows in Ch. 3). It is often possible to associate the error in a calculation with either the problem or the solution, and there may be some choice about how much error is associated with each of these. Thus, in Fig. 2.1, all of the error could be viewed as forward error, with $\Delta x = 0$, or (as illustrated in the figure), part of the error can be associated with the problem. The process of bounding this backward error of a computed solution is called *backward error analysis*, and its motivation is twofold. First, it interprets rounding errors as being equivalent to perturbations in the data. The input data frequently contains uncertainties due to previous computations or errors committed in storing numbers on the computer, as previously mentioned. If the backward error is no larger than these uncertainties then the computed solution can hardly be criticized — it is as good as we can hope for. The second attraction is this. Rather than viewing all of the error as forward error, as mentioned just above, the backward error analysis permits to bound or estimate the influence of the total error by means of perturbation theory [DB08].

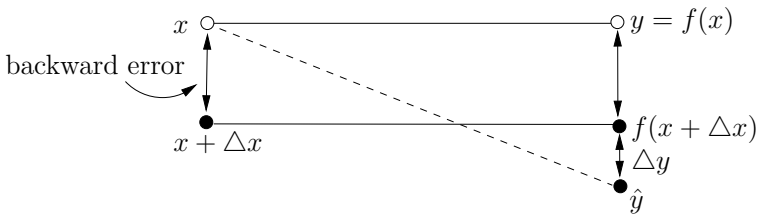


Figure 2.1: Backward/forward error analysis, solid line = exact, dashed line = computed.

2.2 Geometric modeling

Geometric modeling has rapidly become a central area of research and development that involves diverse applications. It is of critical importance in the traditional fields of engineering, general product design, and computer-aided manufacturing. It has also proved to be indispensable in a variety of modern industries, including computer vision, robotics, medical imaging, visualization, etc. [Sar03].

2.2.1 Historical summary

Geometric modeling traditionally identifies a body of techniques that can model certain classes of piecewise parametric surfaces, subject to particular conditions of shape and smoothness [GO97]. Its beginnings can be traced to the 1950s, and from these initial activities emerged four main streams of work that evolved largely independently for some two or three decades. The *computer graphics* stream focused on rendering and interaction. The *wireframe* stream lead to the commercial CAD systems of the 1970s and 80s. The *free-form curve and surface* stream found important applications in computer-aided design and the manufacture of car bodies, aircraft fuselages and in other tasks in the automotive and aerospace industries. *Solid modeling* is distinguished by the use of hopefully unambiguous representations for complete solids. A related fifth stream focuses on the theoretical aspects of design and analysis of geometric algorithms, and has become known as *computational geometry* [Req99]. Since the late 1990s, however, a tendency of convergence of all these different aspects of geometric computation has become evident, and new systems use ideas from all of these fields [Req99].

2.2.2 Geometric representations

The development of complex surface representation schemes has been one of the core fields of computer graphics and geometric modeling. The different representations currently available have succeeded in modeling certain properties of surfaces well, but none of them is general enough to satisfy all the requirements that could be demanded of a representation [HG00]. Two major representation schema are often used: *constructive solid geometry* (CSG) and *boundary representation* (B-rep). In CSG a solid is represented as a set-theoretic Boolean representation of primitive solid objects, so that both the surface and the interior of an object are defined implicitly. In B-rep the solid surface is represented explicitly as a quilt of vertices, edges, and faces [Hof89, GO97].

Most geometric modeling systems use B-rep. The different B-rep schemes appearing in the literature can be divided into two major families. One family restricts the solid surfaces to oriented manifolds. The second allows oriented nonmanifolds. Conversion from CSG to B-rep is usually available [GO97]. Throughout this work, we focus on the B-rep: three such representations will be presented in detail.

Parametric representations

Non-Uniform Rational B-Splines (NURBS) have become a *de facto* industry standard for the representation, design, and data exchange of geometric information processed by computer. Also, many international standards, e.g. STEP Part 42 [Ind97], recognize NURBS as powerful tools for geometric design [PT97]. Their excellent mathematical and algorithmic properties, combined with successful industrial applications, have contributed to the enormous popularity of NURBS. NURBS also play an important role in the CAD/CAM (Computer-Aided Manufacturing)/CAE (Computer-Aided Engineering) world .

A NURBS surface of degree p in the u direction and degree q in the v direction is a bivariate vector-valued piecewise rational function of the form [PT97]

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}(u,v)\mathbf{P}_{i,j} \quad 0 \le u,v \le 1, \tag{2.1}$$

where the $R_{i,j}(u,v)$ are the piecewise rational basis functions and $n = p+1, m = q+1$,

$$R_{i,j}(u,v) = \frac{N_{i,p}(u)N_{j,q}(v)w_{i,j}}{\sum_{k=0}^{n}\sum_{l=0}^{m} N_{k,p}(u)N_{l,q}(v)w_{k,l}}. \tag{2.2}$$

The $\{\mathbf{P}_{i,j}\}$ form a bidirectional control net, the $\{w_{i,j}\}$ are the weights, and the $\{N_{i,p}(u)\}$ and $\{N_{j,q}(v)\}$ are the usual nonrational B-spline basis functions.

NURBS provide a convenient way to describe surfaces of almost any shape. However, the most useful NURBS paradigm is constrained by the requirement that the surfaces are defined over rectangular regions and this leads to topologically rectangular patches. A generalization for an arbitrary topology can be obtained by collapsing some of the control mesh edges, but this creates surfaces with ambiguous surface normals and degenerate parametrization [CM00].

Trimming operations are essential for modeling non-regular B-rep objects. A trimmed-surface data type in the description of free-form objects was therefore introduced to provide greater power and flexibility to the NURBS representation. A trimmed surface is an ordinary tensor product surface that has a restricted parameter domain, thus overcoming the limitation of tensor product surfaces defined over rectangular regions, and allowing for arbitrary domains [CM00]. They can give a complete representation of the boundary of a geometric model by means of union of surfaces restricted to suitable domains.

A trimmed NURBS surface is defined by a tensor product NURBS surface and a set of trimming curves in the parametric space of the surface [CM00]. The additional trimming pro-

cess, using trimming curves, permits the removal of unneeded areas of the traditional NURBS surface. Combining thousands or even tens of thousands of trimmed surfaces makes it possible to design very sophisticated objects [KBK02].

Figure 2.2 gives an example of two trimmed patches joining together to form a single surface. The parametric domain D is delimited by a collection of trimming curves $\boldsymbol{p}$, and the restriction of the mapping $\boldsymbol{F}$ to D defines the trimmed patch in $\mathbb{R}^3$. In addition, explicit boundary information, provided by a function $\boldsymbol{b}(t)$ taking values in $\mathbb{R}^3$, may also be present [SWC00, Spa98, Ind97, KBK02].

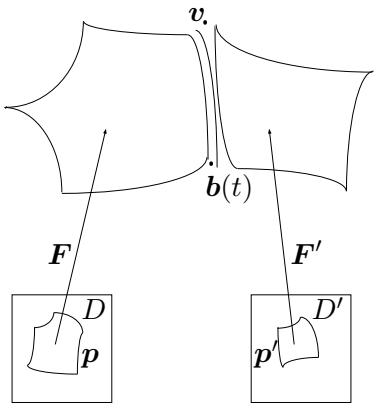


Figure 2.2: Two adjoining trimmed patches in a surface model [ASZ07].

Trimmed NURBS surfaces have been adopted widely by the CAD/CAM industry, and included in graphics standards. They are provided as primitives in several geometric modeling software systems, and the rendering of trimmed NURBS surfaces is supported by international standards, such as STEP Part 42 [Ind97] and PHIGS+ (Programmer's Hierarchical Interactive Graphics System), as well as graphics programming interfaces, such as OpenGL and Direct3D [CM00].

Mesh models

NURBS have the advantage of being able to describe almost any shape conveniently. But even today's advanced graphics hardware is unable to directly render trimmed NURBS models: they need to be transformed into a renderable (e.g. polygonal) representation [BGK04, KBK02]. Similarly, for many applications, piecewise linear approximations of smooth surfaces within a given tolerance are generated. Examples of such applications include finite-element analysis, stereolithography, and visualization of geometric models [SB00]. Many methods have been

proposed in the literature for this triangulation (approximation) procedure [SB00, Sug02].

A *mesh* is a discretization of a geometric domain into small simple shapes, such as triangles or quadrilaterals in two dimensions and tetrahedra or hexahedra in three dimensions [BP00]. Depending on the point of view, meshes can be classified in different ways. Based on topological properties, meshes can be divided into *structured* meshes[1], *unstructured* meshes[2] and *hybrid* meshes[3] [GKSS02, BP00]. Based on the mesh element type, meshes can be categorized into *tri/tetrahedral* meshes, *quad/hexahedra* meshes, and others[4] [Owe98].

For this Ph.D. work, we focused on triangular-surface meshes, based on the fact that we mainly work on B-rep models, and that triangles are the primitive representation elements for rendering. One of the most popular triangle and tetrahedral meshing techniques is based on the use of the Delaunay criterion, namely the Delaunay triangulation method.

Definition

Let S be a set of points in the plane. A triangulation T is a *Delaunay triangulation* of S if for each edge e of T there exists a circle C with the following properties [Che89a]:

- the endpoints of edge e are on the boundary of C, and
- no other vertex of S is in the interior of C.

A circle circumscribing a Delaunay triangle is called a *Delaunay circle*. If S contains four points that are cocircular then the Delaunay triangulation is not unique [Che89b, EL00]. In such a circumstance, any of the possible triangulations will do [Che89a]. The Delaunay triangulation is the straight line dual of the *Voronoi diagram* of S [Che89a].

The Delaunay triangulation has the following properties. Among all triangulations of a vertex set, the Delaunay triangulation maximizes the minimum angle in the triangulation, minimizes the largest circumcircle, and minimizes the largest min-containment circle, where the min-containment of a triangle is the smallest circle that contains it (and is not necessarily its circumcircle) [She99, DS89, BP00].

[1] All interior vertices of the mesh are topologically alike.

[2] Mesh vertices may have arbitrarily varying local topological neighborhoods.

[3] The mesh is formed by a number of small structured meshes combined in an overall unstructured pattern.

[4] This includes mixed tri-quad meshes, mixed tet-hex meshes and other less frequently used element-shape meshes.

Subdivision-surface models

Currently, the most common way to model complex smooth surfaces in the domain of geometric modeling is by using a patchwork of trimmed NURBS. Trimmed NURBS are used primarily because they are readily available in existing commercial systems such as Autodesk. They do, however, suffer from at least two difficulties [DKT98], which are discussed further in Ch. 4:

- Trimming is expensive and prone to numerical error.
- It is difficult to maintain smoothness, or even approximate smoothness, at the seams of the patchwork when the model is animated.

Subdivision surfaces have the potential to overcome both of these problems: they do not require trimming, and smoothness of the model is automatically guaranteed. Also, subdivision surfaces free the designer from worrying about the topological restrictions that haunt NURBS modelers [DKT98]. Further, compared to the regular mesh models presented in the previous section, subdivision-surface models offer more control over the objects, since they contain more topological and geometrical information about the mesh. But, on the other hand, subdivision-surface models also prevent the use of special tools that have been developed over the years to add features to NURBS models, which is one of the hindrances for the extensive use of subdivision-surface models, especially in the domain of CAD.

Subdivision is a method for generating smooth surfaces, which first appeared as an extension of splines to arbitrary-topology control nets, and was introduced as a generalization of knot insertion algorithms for splines. But it is much more general and offers considerable freedom in the choice of subdivision rules [Zor97]. Subdivision surfaces were first introduced to the domain of geometric modeling 1978, with the papers by Catmull and Clark [CC78], and by Doo and Sabin [DS78]. Subdivision-surface models are now widely used in many application areas, including computer graphics, solid modeling, computer-game software, film animation and others, as an alternative to B-splines and NURBS [AS09].

The basic idea of subdivision is to define a smooth curve or surface as the limit of a sequence of successive refinements [ZSD$^+$00]. Most oftenly the subdivision procedure contains two main steps: *refinement* and *smoothing*. Refinement (splitting rule) means splitting the edges and faces by inserting new vertices to obtain a finer version of the mesh, and smoothing (averaging rule) means shifting the vertices in order to increase the overall smoothness of the surface [AS09, ZSD$^+$00].

Classification – Many different subdivision schemes have been proposed in the last two decades. Based on different criteria, these schemes can be classified differently. For example, as proposed in [AS09], they can be classified according to the type of spline that is generated by the method: B-spline methods, Box-spline methods, general-subdivision-polynomial methods and affine-invariant subdivision methods (Fig. 2.3). Similarly, based on the presence or absence of an interpolating property of the produced surface, subdivision schemes can be categorized as: *interpolating* methods (e.g. Modified Butterfly [ZSS96], Kobbelt [Kob96]) and *approximating* methods (Doo-Sabin [DS78], Catmull-Clark [CC78], Loop [Loo87], 4-8 [VZ01], $\sqrt{3}$ [Kob00]).

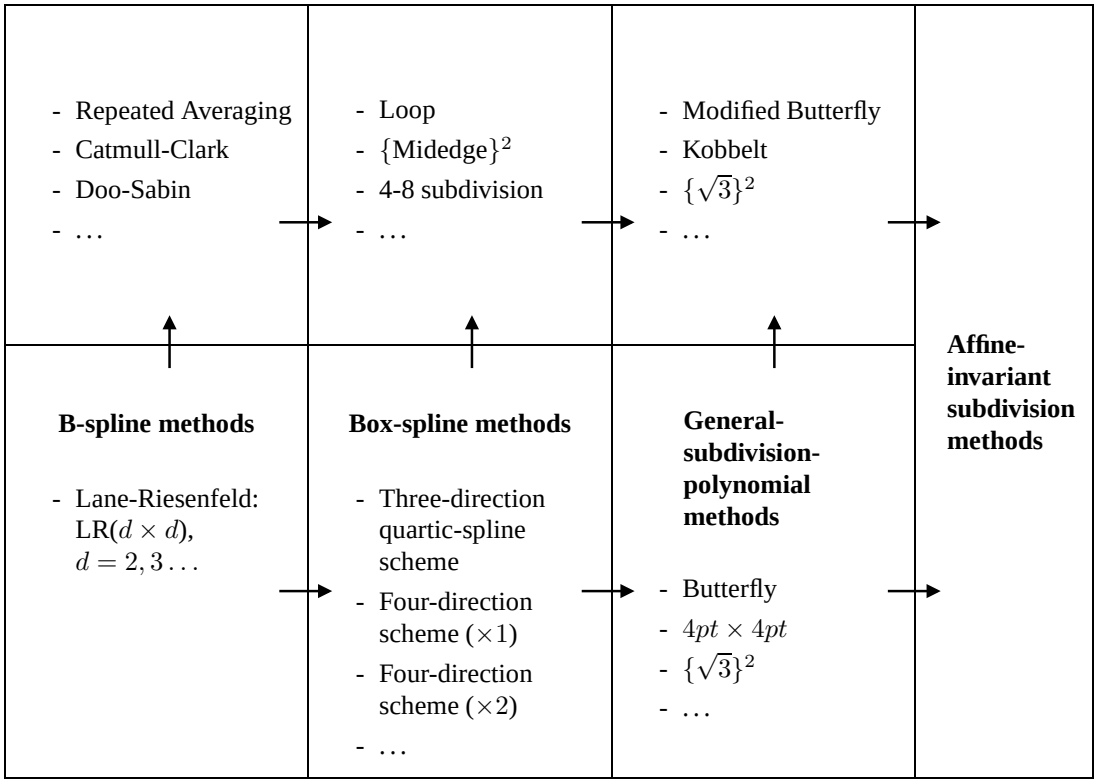


Figure 2.3: Subdivision-scheme classifications [AS09].

Surface evaluation – Another important issue concerning subdivision-surface models is surface evaluation. The first evaluation method (other than subdivision refinement itself) was proposed by Stam [Sta98a, Sta98b]: this method parameterizes the control mesh and the limit surface over a unit-mesh element (triangle or quadrilateral) to evaluate the surface at an arbitrary parameter value. Another method was presented in [WP04, WP05, BS02] . It uses the linearity of the subdivision process, the parameterization of the control mesh and the limit surface is set to be centered at each vertex (Fig. 2.4), such that the limit surface is evaluated as the linear combination of the basis functions, weighted by the original control points. One advantage of

this technique is that the parameterization near the extraordinary vertex has n-gon symmetry. It is the second method that we have used in the paper that follows in Ch. 5.

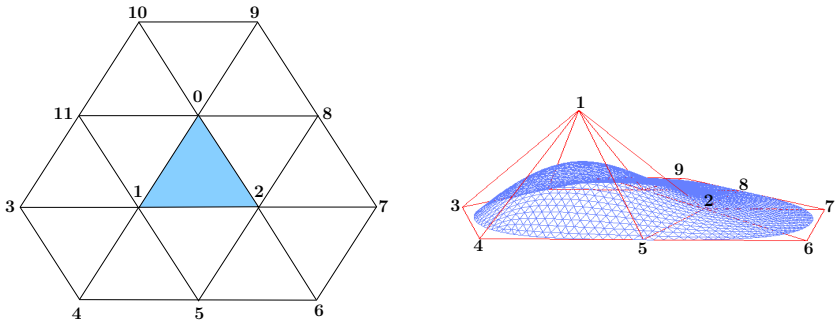


Figure 2.4: Wu-Peters [WP04] evaluation method: left: a base mesh used to generate the basis functions for the triangle 0-1-2 (regular case: vertex with valence 6); right: the resulting basis function at node 1 evaluated at subdivision level four.

Multiresolution – Multiresolution is a natural extension of subdivision surfaces. It extends subdivision by including detail offsets at every level of subdivision, unifying patch-based editing with the flexibility of high-resolution polyhedral meshes [ZSD+00, ZSS97].

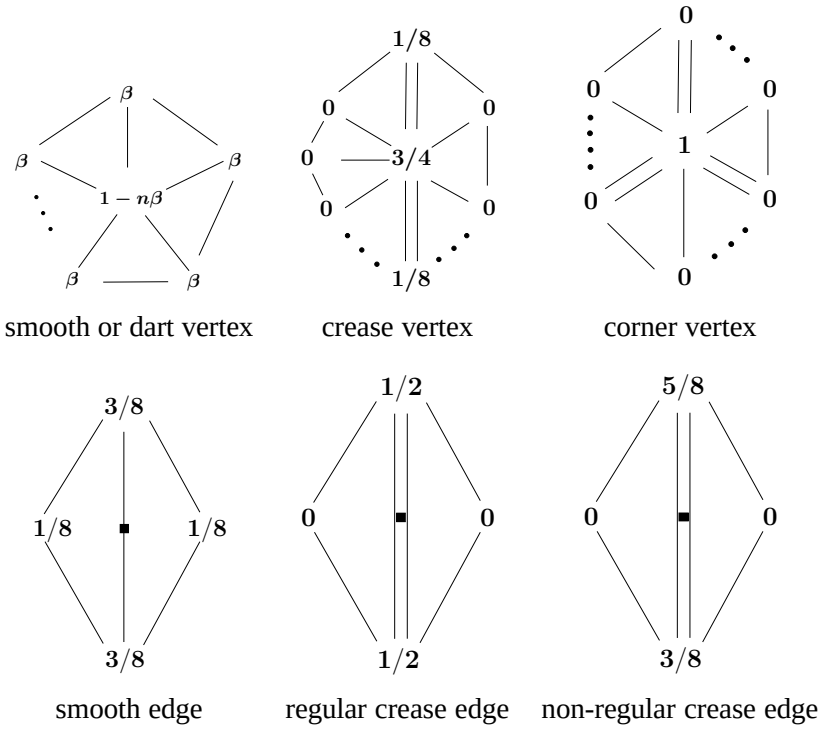
Lounsbery et al. were the first to propose algorithms to extend classical multiresolution analysis to arbitrary topology surfaces [Lou94, LDW97]. There are now many different techniques available for converting subdivision surfaces into a multiresolution hierarchy [LSS+98]. Two main schools exist. One approach extends classical multiresolution analysis and subdivision techniques to arbitrary topology surfaces [Lou94, LDW97, EDD+95, CPD+96]. The alternative is more general and is based on sequential mesh simplification, e.g. progressive meshes [Hop96, HG97]. In either case, the objective is to represent triangulated 2-manifolds in an efficient and flexible way [LSS+98].

For this work we are mostly interested in the triangular B-rep, so we will give more details on the now-classical Loop subdivision scheme. The Loop scheme is a simple approximating face-split scheme for triangular meshes first proposed by Loop [Loo87]. It is based on the *three-directional quartic box spline* [Bar07], which produces C^2-continuous surfaces over regular meshes. The Loop scheme produces surfaces that are C^2-continuous everywhere except at extraordinary vertices, where they are C^1-continuous. Later Hoppe et al. [HDD+94] proposed an extension to the Loop scheme with special rules defined for edges to include features such as creases and corners. In [BLZ00], the boundary rules are further improved, and new rules for concave corners and normal modification are proposed. The Loop scheme can be applied to

arbitrary polygonal meshes, and the resulting mesh is a triangular mesh [ZSD+00]. The proof of continuity of this scheme for all valences can be found in [Sch96, Zor97]. Below are the three important masks for the Loop subdivision scheme.

1. Subdivision mask

   A subdivision mask defines where new vertices will be inserted and how already existing vertices should be shifted at each subdivision step. Fig. 2.5 shows the subdivision mask for Loop subdivision scheme [HDD+94].



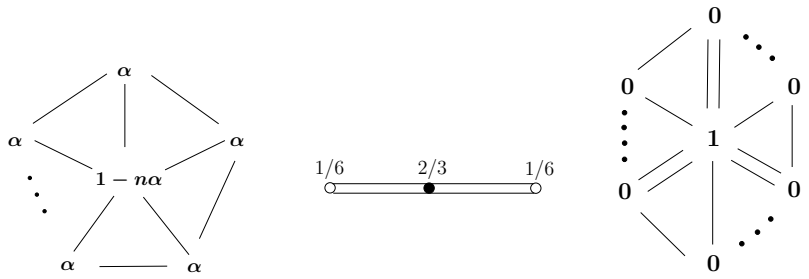
smooth or dart vertex | crease vertex | corner vertex

smooth edge | regular crease edge | non-regular crease edge

Figure 2.5: The subdivision mask for Loop subdivision scheme, where $\beta = \beta(n) = \frac{a(n)}{n}$, and $a(n) = \frac{5}{8} - \frac{(3+2\cos(2\pi/n))^2}{64}$. This equivalent form can be obtained from the substitution of $1 - n\beta = \frac{\alpha(n)}{n+\alpha(n)}$.

2. Limit mask

   A limit mask calculates the limit position of each vertex in the control mesh. The limit position can be expressed as an affine combination of the initial vertex position and its immediate neighboring vertices. For Loop subdivision scheme, this combination is expressed by the following mask (Fig. 2.6) [HDD+94, MMTP04].



smooth or dart vertex crease or boundary vertex corner vertex

Figure 2.6: The limit-position mask for Loop subdivision scheme, where α is defined as $\alpha = \alpha(n) = (\frac{3}{8\gamma(n)} + n)^{-1}$, with $\gamma(3) = \frac{3}{16}$, and $\gamma(n) = \frac{1}{n}(\frac{5}{6} - (\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n})^2)$ for $n \geq 4$.

3. Tangent mask

   Tangent vectors of the limit surface can be computed using the two left eigenvectors of the local subdivision matrix corresponding to the second largest eigenvalues. Then their cross product gives an exact normal vector to the limit surface. For a Loop surface, it can be expressed by the tangent mask (Fig. 2.7) [HDD+94, Kob98].

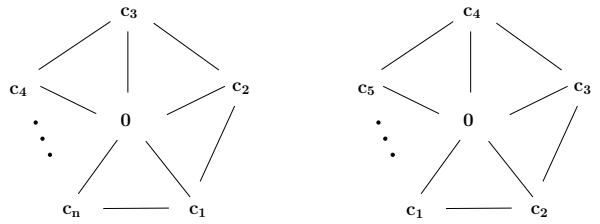


Figure 2.7: The tangent mask for Loop subdivision scheme, where $c_i = \cos(2\pi i/n)$.

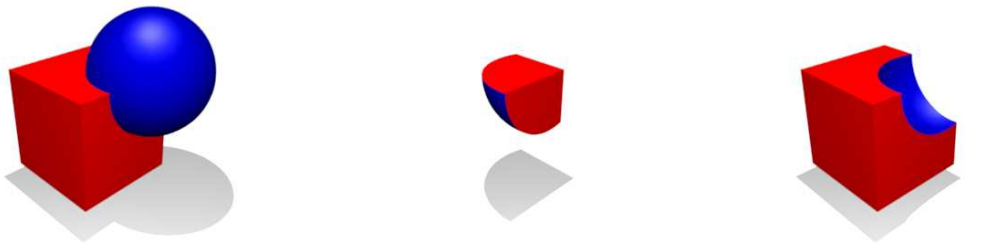
2.2.3 Geometric operations for geometric models

In most geometric modeling systems, geometric operations can be used to generate free-form models based on some primitive models, e.g. the geometric sweep operation [SG05]. Here we give two main groups of these operations.

- Boolean operations

  One of the most important facilities of solid modelers is the Boolean operations between solids [TTSC91, BKZ01]. Regularized Boolean operations include: *regularized union* $\bigcup^*$, *regularized intersection* $\bigcap^*$, and *regularized difference* $-^*$ (Fig. 2.8). They differ from the corresponding set-theoretic operations in that the result is the closure of the op-

eration on the interior of the two solids, and they are used to eliminate "dangling" lower-dimensional structures [Hof89]. These operations can be applied to both CSG models and the B-reps[5] and include some low-level operators as *classification*, *orientation*, *merging* and *intersection*.



union ($\bigcup^*$) intersection ($\bigcap^*$) difference ($-^*$)

Figure 2.8: Regularized Boolean operations [g-b].

- Signal processing

  Signal processing contains another important group of operations that has been widely used in the domain of geometry processing. It includes *downsampling*, *upsampling*, *smoothing* [JDD03], *filtering* [Ale02], etc., which have been used for geometry editing, simplification, denoising, compression and simulation [GSS99]. The paper [BPK+07] gives a nice overview on this subject.

Throughout this Ph.D. work, we put our focus on the Boolean operations on B-reps, although other related geometric operations are also studied.

2.2.4 Robustness issues

Boolean operations have been used in most modeling systems, but most often, care still has to be taken to handle special and degenerate cases for these operations [BMS94, TTSC91, BKZ01, Far99]. The inconsistencies arising from numerical error can lead to connectivity faults, such as breaks in the supposed boundary. And the inaccuracies in the calculations can also create geometric errors, often in the form of boundary self-intersections [SD07, Hof89]. In addition, implementation of Boolean operations is especially difficult for higher-order B-reps as it requires intersecting parametric surfaces, separating them into pieces and constructing new surfaces out of these pieces. Existing systems typically treat a B-rep as a collection of trimmed

[5] Algorithms for Boolean operations on B-reps are called also *boundary-evaluation and merging* algorithms.

spline patches, sharing boundaries. The boundaries of each individual patch are often matched only approximately, since it is difficult to ensure that two trimming curves in different parametric domains are identical in space. Thus each intersection operation leads to increasingly complex and difficult-to-handle trimming curves. Applying smooth deformation to the resulting models is also a very difficult task: special care must be taken to avoid cracks, etc. [Man88]. As a result, Boolean operations usually are neither fast nor robust, although excellent results have been achieved by some commercial solid modeling engines [LC07, BKZ01, BK97].

The framework necessary to prove that algorithms work rigorously is available [ASZ07], but, so far at least, the required analyses appear to be intractable. Much research has been devoted to seeking robust geometric-operation algorithms. Two groups of methods have been proposed to repair dirty CAD models: *surface-based* techniques and *volumetric* techniques. Surface-based techniques work directly on the input surface, using different methods to detect and resolve artifacts. These techniques include snapping boundaries to each other, projecting and inserting one boundary into the other, computing intersections of extended surface patches, and propagating the normal field from patch to patch [BK97, BW92, BS95, BDK98, GTLH01]. The *volumetric* technique converts the input into a volumetric representation, effects the repair in the volumetric model and extracts a surface as the final result. It contains different techniques for the B-reps to volumetric representation conversion [NT03, Ju04, FPRJ00], and for the surface extractions [KBSS01, Gib98, JLSW02]. Also, different hole-filling methods have been proposed [BK05, ABA02, DMGL02, NT03] for this volumetric technique. It is the surface-based technique that we will use in the paper in Ch. 5.

Robust operations on subdivision-surface models have recently attracted a lot of attention. Lai and Cheng [LC07, LC06] presented an algorithm that performs error-controllable Boolean operations on Catmull-Clark subdivision-surface models, using a volumetric approach. Lanquetin et al. [LFKN03] proposed an intersection calculation method for subdivision-surface models based on triangle-grouping technique. Biermann et al. [BKZ01] used a perturbation technique to avoid degenerate cases for Boolean operations on Loop subdivision-surface models. Further Smith and Dodgson [SD07] used symbolic-perturbation methods to guarantee topological correctness of the computed result of Boolean operations. In one of the following papers (Ch. 5), we proposed an algorithm performing Boolean operations on Loop subdivision-surface models using limit-mesh representation, with a verification method designed to guarantee the well-formedness of the computed result.

Chapter 3

Floating-point arithmetic for computational-geometry problems with uncertain data

This chapter presents our work on the application of backward error analysis in the area of computational geometry. The analysis is relevant in the context of uncertain data, which may well be the practical context for computational-geometry algorithms.

It has been suggested in the literature that ordinary finite-precision floating-point arithmetic is inadequate for geometric computation, and that researchers in numerical analysis may believe that the difficulties of error in geometric computation can be overcome by simple approaches. It is our purpose of this work to show that these suggestions, based on an example showing failure of a certain algorithm for computing planar convex hulls, are misleading, and why this is so.

Our exposition illustrates the fact that the backward error analysis does not pretend to overcome the problem of finite precision: it merely provides a tool to distinguish, in a fairly routine way, those algorithms that overcome the problem *to whatever extent it is possible to do so.* We also show that the situation in computational geometry, as mentioned in our principal reference [2], is exactly parallel to other areas. For example, algorithms for the planar convex-hull problem were discussed in [2], along with examples of failure of certain of the algorithms. But, although those failures are spectacular, the situation is exactly analogous to many areas of numerical analysis: there are certain algorithms that are stable, and certain algorithms that are unstable. If an unstable algorithm is used to solve a problem, then it may produce completely

wrong results, and this without warning. On the other hand, if a stable algorithm is applied, then, in the case of problems defined in terms of uncertain data, the algorithm produces an answer that is essentially as good as we can hope for. This means, in particular, that one cannot do better by using exact arithmetic.

Three examples (solving linear equations, the planar convex-hull problem and a three-dimensional extruded-objects problem) are then presented to illustrate our method of performing backward error analysis: how to measure the adequacy, how to perform the perturbation analysis and how to seek stable solution methods.

Part of the work was first presented at the Sixth Annual International Workshop on Computational Geometry and Applications, Glasgow, UK, May 8-11, 2006, and it appeared in *Lecture Notes in Computer Science* LNCS3980, pages 50-59, 2006. We also invited the authors of our main reference [2] to reply to our paper; the reply is published together with our initial paper in the LNCS volume [KMP$^+$06]. It is an interesting discussion that shows different points of view concerning the same problem in different research domains. The extended version of the paper presented here, which shows how the results apply in a simple three-dimensional case, will appear in the International Journal of Computational Geometry and Applications.

The main contributions of this work are:

- we show that the numerical difficulties described in the principal reference [2] are unexceptional.
- we show how to perform perturbation analysis in geometry modeling with three examples.

Floating-point arithmetic for computational-geometry problems with uncertain data

D. Jiang N. F. Stewart [1]

Département d'informatique et de recherche opérationnelle

Université de Montréal

to appear in

International Journal of Computational Geometry and Applications

[1]The research of the second author was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

Abstract

It has been suggested in the literature that ordinary finite-precision floating-point arithmetic is inadequate for geometric computation, and that researchers in numerical analysis may believe that the difficulties of error in geometric computation can be overcome by simple approaches. It is the purpose of this paper to show that these suggestions, based on an example showing failure of a certain algorithm for computing planar convex hulls, are misleading, and why this is so.

It is first shown how the now-classical backward error analysis can be applied in the area of computational geometry. This analysis is relevant in the context of uncertain data, which may well be the practical context for computational-geometry algorithms such as, say, those for computing convex hulls. The exposition will illustrate the fact that the backward error analysis does not pretend to overcome the problem of finite precision: it merely provides a way to distinguish those algorithms that overcome the problem *to whatever extent it is possible to do so.*

It is then shown that often the situation in computational geometry is exactly parallel to other areas, such as the numerical solution of linear equations, or the algebraic eigenvalue problem. Indeed, the example mentioned can be viewed simply as an example of the use of an unstable algorithm, for a problem for which computational geometry has already discovered provably stable algorithms.

Finally, the paper discusses the implications of these analyses for applications in three-dimensional solid modeling. This is done by considering a problem defined in terms of a simple extension of the planar convex-hull algorithm, namely, the verification of the well-formedness of extruded objects. A brief discussion concerning more difficult problems in solid modeling is also included.

Keywords:

floating-point arithmetic, robustness in geometric computation, stability, planar convex hull, backward error analysis.

3.1 Introduction

This paper is an extended version of a previous paper [1]. It discusses the use of floating-point arithmetic for the solution of problems in computational geometry that are defined in terms of uncertain data.

It has been suggested in the literature [2] that ordinary finite-precision floating-point arithmetic [3] is inadequate for geometric computation, and that researchers in numerical analysis may believe that the difficulties of error in geometric computation can be overcome by simple approaches. As pointed out in the previous paper [1], these suggestions are misleading[2], and it is the purpose of this paper to show why this is so.

3.1.1 Paper outline

We begin with a slightly modified version of the exposition in the previous paper [1], which illustrates how the backward/forward error analysis, from numerical analysis, relates to the study of robustness in computational geometry. This exposition is focused on the problem of planar convex hulls.

Algorithms for the planar convex-hull problem were discussed in a recent paper [2], along with examples of failure of certain of the algorithms. But, although those failures are spectacular, the situation is exactly analogous to many areas of numerical analysis: there are certain algorithms that are stable, and certain algorithms that are unstable. If an unstable algorithm is used to solve a problem, then it may produce completely wrong results, and this without warning. On the other hand, if a stable algorithm is applied, then, in the case of problems defined in terms of uncertain data, the algorithm produces an answer that is essentially as good as we can hope for. This means, in particular, that one cannot do better by using exact arithmetic.

Having established these basic facts, we go on to illustrate the implications of this discussion for applications in three-dimensional solid modeling. We make this link by the simple device of considering *extruded objects*, defined in terms of a two-dimensional contour and a direction $\boldsymbol{d}$ that defines the path along which the contour should be swept[3]. Objects of this kind have formed part of solid modeling systems from the very beginning, since such objects are widely used in design, and correspond to widely used manufacturing processes [4, 5]. The illustration given here will show how the question of well-formedness of such an object, in the (usual)

[2]The previous paper [1] includes an invited reply from the authors of the original paper [2].

[3]Throughout the paper, boldface characters are used to denote vectors in $\mathbb{R}^n$, and, in particular, in $\mathbb{R}^2$.

context of uncertain data, can be reliably guaranteed using already established results [6], along with Fortune's stable implementation of the Graham scan [1, 2, 7, 8] implemented in ordinary floating-point arithmetic. We then conclude with some remarks about the use of such arithmetic for more general problems in solid modeling.

3.1.2 Comments concerning the failure of an algorithm

As stated in the principal reference [2], "... the algorithms of computational geometry are designed for a machine model with exact arithmetic. Substituting floating-point arithmetic for the assumed real arithmetic may cause implementations to fail." The paper [2] goes on to say that "due to ... [a] ... lack of examples, instructors of computational geometry have little material for demonstrating the inadequacy of floating-point arithmetic for geometric computations, students of computational geometry and implementers of geometric algorithms still underestimate the seriousness of the problem, and researchers in our and neighboring disciplines, e.g., numerical analysis, still believe, that simple approaches are able to overcome the problem." An incremental scan algorithm (which is related to Graham's scan [8] and which we will refer to as *Graham_incremental*), for planar convex hulls, is then studied in some detail. In particular, examples are given which show the algorithm can fail, and an explanation is given for why it fails, when executed with floating-point arithmetic.

The examples given in the principal reference [2] should indeed be useful to students and teachers of computational geometry, in order to illustrate what can go wrong, and why, when finite-precision arithmetic is used to solve geometric problems. Furthermore, the paper [2] presents the results of experiments that are repeatable in every detail. In fact, we have implemented the *Graham_incremental* algorithm for example A1 of the principal reference [2], and we confirm that the algorithm behaves exactly as described there when applied to the data given. Briefly, for example A1, *Graham_incremental* produces a completely spurious result.

There are, however, three misleading suggestions in the final sentence quoted above, and it would be unfortunate if they were communicated to students of computational geometry. One of these is the suggestion that the approaches of computational geometry and numerical analysis are somehow adversarial, since in fact they are complementary. Another is the suggestion that numerical analysts believe that they can "overcome" the problem of finite precision. This is not true. What *is* true, however, is that in the case where input data is uncertain and a stability result is available, a backward/forward error analysis, and often a pure backward error analysis, can

deal with the problem in a fairly routine way, by showing that a stable algorithm overcomes the problem of finite precision *to whatever extent it is possible to do so*. Indeed, a stable algorithm provides us with a solution that is as good as the data warrants [9]. (Stability will be defined below in the context of a combined backward/forward analysis, but we will usually just refer to a backward error analysis, since this is usually sufficient.)

A third misleading remark in the passage, quoted above, is the reference to the "inadequacy" of floating-point arithmetic for geometric computations, which is incorrect as a general statement. In fact, some algorithms using floating-point will provide adequate solutions, while others will not, and a backward error analysis will permit us to recognize which algorithms are satisfactory. On the other hand, it *is* true that we must begin by defining precisely what constitutes an *adequate*, or *inadequate*, solution to a geometric problem.

We will show below that numerical robustness for the convex-hull problem is analogous to the case of linear equations, or the algebraic eigenvalue problem, and that when input data is uncertain, the difficulties documented in our principal reference [2] fit exactly into the paradigm of the backward error analysis. We emphasize that this does not imply that research into other paradigms, including exact arithmetic and others, should not be vigorously pursued. Our only claim is that in the proper context (uncertain input data), the backward-error analysis is a useful approach, and it should not be neglected.

We also present a brief summary of how the backward error analysis is used in numerical linear algebra, and a simple example is given to show that breakdowns of methods, of the sort described for the convex-hull problem, are quite typical in other fields. Then, a description of the combined backward/forward error analysis is given, and applied to the planar convex-hull problem. These ideas were developed several decades ago, but that work [9, 10] is very much relevant today. As already mentioned, the first task is to define exactly what is meant by the "inadequacy" of a solution to the convex-hull problem. We are then in a position to do a *perturbation analysis* [10] to examine the effects of perturbations of the input data (whether they are caused by original uncertainty or by subsequent application of a stable numerical algorithm). Finally, we discuss Fortune's implementation of the Graham scan, which we will call *Graham_Fortune*. This implementation is numerically stable for the planar convex-hull problem, as proved by Fortune [7]. Indeed, a slight modification of the algorithm will produce a sequence of points that lie on the topological boundary [11] of their convex hull, and this convex set is the correct convex hull for points that have been relatively perturbed by a small amount. Thus,

we can use a pure backward error analysis to affirm that *Graham_Fortune* provides a solution that is as good as we can hope for, given that the data is uncertain.

The situation for the geometric problem of finding planar convex hulls is, therefore, closely analogous to the case of solving linear equations. In both cases there exist unstable algorithms (*Graham_incremental*, and Gaussian elimination without pivoting, respectively), and in both cases there exist stable algorithms (*Graham_Fortune*, and Gaussian elimination with total pivoting, respectively). Also, in both cases there exist examples for which unstable algorithms produce complete nonsense, and this with no warning that anything is amiss. In fact, the only breakdown in the analogy is that in the case of the geometric problem, with the error criterion used below as an illustration, the situation is much *better* than for solving linear equations. This is because the perturbation analysis, mentioned above, shows that the problem is *well-conditioned*, which is not always true for linear equations. Thus, whereas even a stable algorithm may produce an unsatisfactory answer for the problem $A\boldsymbol{x} = \boldsymbol{b}$ (if A is the Hilbert matrix, for example), a stable algorithm such as *Graham_Fortune* always produces a satisfactory answer for the convex-hull problem.

3.2 Backward error analysis for linear-equation solvers

For linear equations, the problem is defined by the pair $[A, \boldsymbol{b}]$, and the solution is defined by $\boldsymbol{x}$ such that $A\boldsymbol{x} = \boldsymbol{b}$. We proceed as follows:

a. *Measuring error in the solution space.* A measure of the inadequacy of an approximate solution $\boldsymbol{y}$, for the problem $[A, \boldsymbol{b}]$, is the relative error $\frac{\|\boldsymbol{x}-\boldsymbol{y}\|}{\|\boldsymbol{x}\|}$, where $\|\cdot\|$ denotes any convenient vector norm [10].

b. *Perturbation Analysis.* A simple argument shows that if δA is a matrix representing perturbation of the elements of A, and if $\delta\boldsymbol{b}$ is a vector representing perturbations of the elements of $\boldsymbol{b}$, then the solution $\boldsymbol{y}$ of the perturbed problem $[A + \delta A, \boldsymbol{b} + \delta\boldsymbol{b}]$ satisfies (neglecting second-order terms):

$$\frac{\|\boldsymbol{x} - \boldsymbol{y}\|}{\|\boldsymbol{x}\|} \leq \|A\| \cdot \|A^{-1}\| \left\{ \frac{\|\delta A\|}{\|A\|} + \frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|} \right\}, \tag{3.1}$$

where $\|\cdot\|$ is now used also to denote a matrix norm subordinate [10] to the vector norm introduced above. The quantity $\|A\| \cdot \|A^{-1}\|$ is usually referred to as the *condition*

number of the problem: for a trivial matrix like the identity it will be equal to 1, while for a Hilbert matrix of even moderate dimension it will be very large. The condition number represents the amount by which a given perturbation of the input data for $A\boldsymbol{x} = \boldsymbol{b}$ will be magnified in the solution. A problem with a low condition number is said to be *well-conditioned*, and a problem with a large condition number is said to be *ill-conditioned*. The two cases are illustrated by the lines linking problems to solutions in Figures 3.1 and 3.2, where P denotes the class of problems, and S denotes the class of solutions [12]. In Figure 3.1, a small perturbation in the problem produces a small perturbation in the solution, while in Figure 3.2, a small perturbation in the problem produces a large perturbation in the solution. (The meaning of the unfilled circles in the figures will be explained immediately below.)

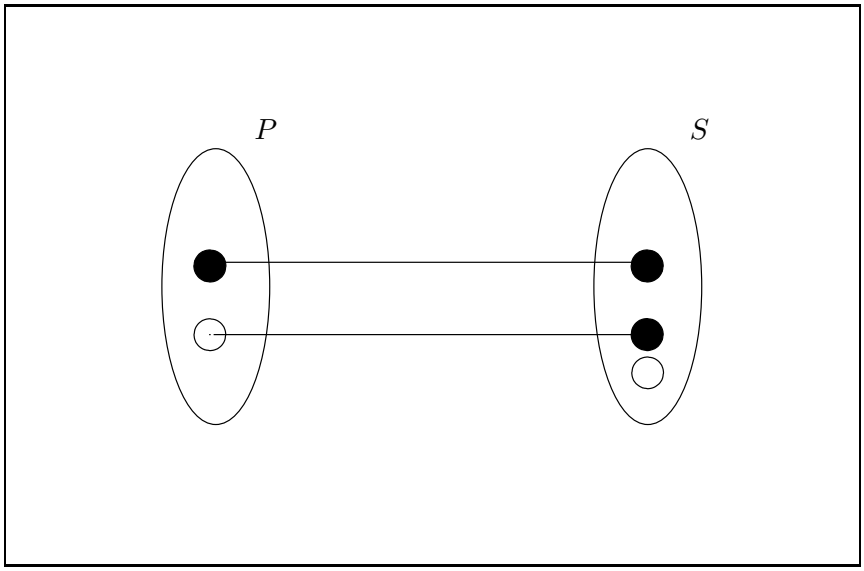


Figure 3.1: Well-conditioned problem.

c. *Stability proof.* The third step is to seek *stable* algorithms, that is, algorithms that produce a slightly incorrect solution to a slightly perturbed problem [9], as illustrated by the unfilled circles in Figures 3.1 and 3.2. (This describes a combined backward/forward error analysis; if the words "a slightly incorrect solution" can be replaced by "the exact solution", so that there is no need for the unfilled circle in S, then we have a pure backward error analysis.) Gaussian elimination with total pivoting is stable for the problem $A\boldsymbol{x} = \boldsymbol{b}$. Such algorithms produce answers that are, for practical purposes, as good as the best answers we can hope for *(even using infinite precision)*, if the "slight perturbation" is

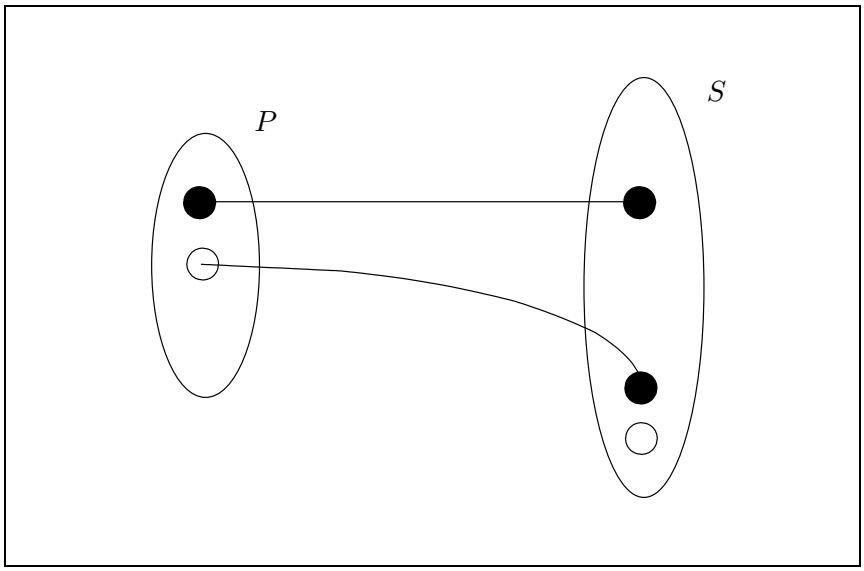


Figure 3.2: Ill-conditioned problem.

smaller than the uncertainty already in the data. Furthermore, by the perturbation analysis of step b, above, the size of the error in the solution can be estimated.

It should be observed that the concept of problem condition, and the corresponding perturbation analysis, are considered prior to any discussion of numerical methods [10]. This reflects the central idea of the backward error analysis: if the elements of A contain uncertainty that may be as large as $\frac{\|\delta A\|}{\|A\|}$, and the elements of $\boldsymbol{b}$ contain uncertainty that may be as large as $\frac{\|\delta \boldsymbol{b}\|}{\|\boldsymbol{b}\|}$, then the relative error $\frac{\|\boldsymbol{x}-\boldsymbol{y}\|}{\|\boldsymbol{x}\|}$ may be as large as is indicated in (3.1). This means that even an exact, infinite-precision algorithm cannot help us avoid a large error in the solution, in the case of an ill-conditioned problem, because of the effects of the inherent uncertainty in the data (see Figure 3.2). It also means, however, that if we can find an algorithm that produces a solution that is the exact solution, of a problem that differs from the given problem by an amount smaller than the inherent uncertainty in the data, then the algorithm has produced an answer that is as good as the data warrants [9].

We emphasize again that a stable algorithm does not necessarily produce an answer with small error: it only produces an answer with error on the order of that which we must accept in any event, due to data uncertainty (see Figure 3.2, where the unfilled circle in S indicates that the method has done a good job of solving the perturbed problem, but has nonetheless produced an answer with large error). The backward error analysis does not "overcome" the problem of numerical error: it merely allows us to identify algorithms that produce errors of the same order

as those that we must accept anyway.

We conclude this section with the remark that computational geometry is by no means unusual in the fact that there are theoretically exact algorithms that produce nonsense when implemented in floating-point arithmetic. For example, in the case of $A\boldsymbol{x} = \boldsymbol{b}$, suppose we attempt to solve the sequence of problems

$$\begin{bmatrix} \phi(\rho) & 1.0 \\ 1.0 & 0.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix}, \quad \rho = 0.1, 0.2, \ldots$$

where $\phi(\rho) = \rho^2 - 0.01$. For $\rho = 0.1$, the correct answer is $x_1 = 1.0$, $x_2 = 1.0$, but Gaussian elimination without pivoting, as implemented in the following program, returns the answer $x_1 = 0.0$, $x_2 = 1.0$. There is no division by zero, and no overflow or underflow occurs during the execution of the implemented algorithm. In the evaluation of $b[1]$, however, the first term on the righthand side of the assignment statement is shifted off the end of a register and ignored.

```
  double rho = 0.1, phi = rho * rho - 0.01 ;
  double A[2][2] = {{phi, 1.0} , {1.0, 0.0}} , b[2] = {1.0, 1.0} ;
  double x[2] ;
/********* Triangulate A **********/
  double mult = 1.0 / A[0][0] ;
  A[1][1] = A[1][1] - mult * A[0][1] ;
  b[1] = b[1] - mult * b[0] ;

/********* Back-substitute *********/
  x[1] = b[1] / A[1][1] ;
  x[0] = (b[0] - A[0][1] * x[1]) / A[0][0] ;
  cout<<" The result is: "<<x[0]<<" "<<x[1]<<endl ;
/*******************************/
  The result is: 0 1
```

3.3 Backward error analysis for planar convex hulls

We will now provide a parallel development for the problem of computing convex hulls of points in the plane. In this case the problem is defined [2] by a finite set of vectors $S = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$, where each $\boldsymbol{a}_i$ lies in the plane[4]. An algorithm to compute the extreme points of S will normally select a subset $\{i_1, i_2, \ldots, i_m\}$ of the indices $\{1, \ldots, n\}$ and declare $\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m}\}$ to be the

[4]We denote the points by $\boldsymbol{a}$ in order to increase the parallelism with Section 3.2.

solution, but since we are envisaging the possibility of uncertainty in the problem data, we will permit any non-empty finite set of vectors $Y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m\}$ as a solution, where each $\boldsymbol{y}_i$ lies in the plane.

3.3.1 Step a: measuring inadequacy

If $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ is a finite set of vectors in the plane, define $conv(\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}) \subseteq \mathbb{R}^2$ to be the convex hull of the set of vectors. We will define the distance d between two distinct solutions Y^1 and Y^2 of the convex-hull problem to be infinite if for $i = 1$ or 2 the vectors in Y^i do not actually lie on the topological boundary of $conv(Y^i)$; otherwise, d is defined to be the Hausdorff distance between $conv(Y^1)$ and $conv(Y^2)$. Defining $d(Y^1, Y^1) = 0$, the distance d is a metric. Let $\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m}\}$ be a set of points lying on the topological boundary of $conv(\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\})$, and such that $conv(\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m}\}) = conv(\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\})$. The *error* E in a solution Y is defined to be

$$E = d(\{\boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_m}\}, Y)/M, \tag{3.2}$$

where M is a fixed upper bound for the absolute value of any coordinate of any point [7]. (Thus, for a solution to be considered accurate, its points are required to actually lie on a convex polygon [13]). In Figure 3.3, the solution to the problem defined by $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \boldsymbol{a}_4, \boldsymbol{a}_5\}$ is $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_4, \boldsymbol{a}_5\}$. The solution $Y = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \boldsymbol{a}_4, \boldsymbol{a}_5\}$ has infinite error, since $\boldsymbol{a}_3$ is not in the

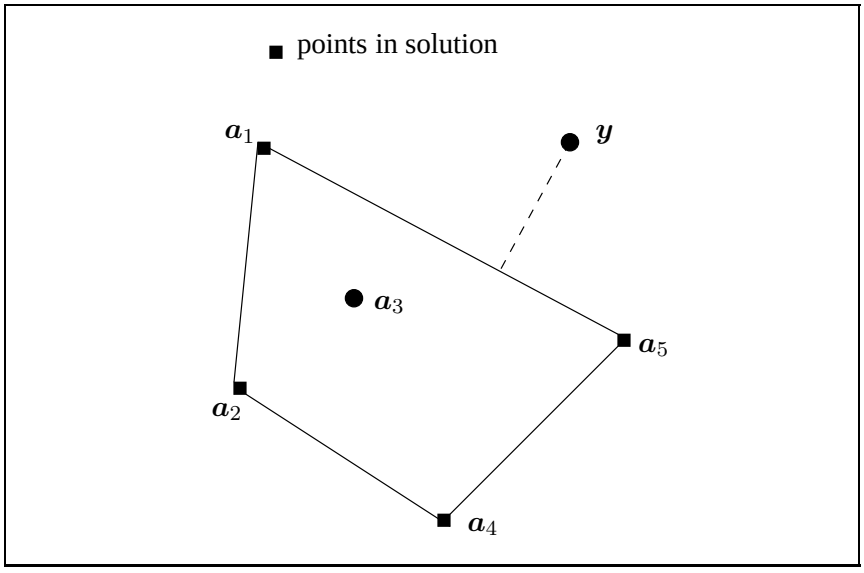


Figure 3.3: Example convex-hull problem.

boundary of $conv(\{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \boldsymbol{a}_4, \boldsymbol{a}_5\})$, while $Y = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_4, \boldsymbol{a}_5, \boldsymbol{y}\}$ has error as indicated by the dashed line.

It is possible to define other measures of distance between solutions of this problem, *e.g.*, we might penalize solutions with redundant points on the boundary of the convex hull.

We will use the simple criterion (3.2) to illustrate our point, which is that if we wish to prove rigorous theorems about the inadequacy of computed solutions, we must give a careful definition of inadequacy.

3.3.2 Step b: perturbation analysis

If the input data $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$ is uncertain, then the true problem that we wish to solve is defined by $\{\boldsymbol{a}_1+\delta\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n+\delta\boldsymbol{a}_n\}$, where each $\delta\boldsymbol{a}_i$ is a vector in the plane. Suppose that $\frac{\|\delta\boldsymbol{a}_i\|_2}{\|\boldsymbol{a}_i\|_2} \leq \Delta$, $i = 1, \ldots, n$, where $\|\cdot\|_2$ denotes the Euclidean norm. This means that the relative error in the computed solution could be as large as $\Delta\sqrt{2}$, due to the uncertainty in the input data alone, since the Hausdorff distance between $conv(\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\})$ and $conv(\{\boldsymbol{a}_1 + \delta\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n + \delta\boldsymbol{a}_n\})$ has the achievable upper bound of $\Delta\sqrt{2}M$. Thus, if criterion (3.2) is used, $\sqrt{2}$ can be taken as a condition number for the problem of planar convex hulls.

In comparison with the linear-equations case, this is a very satisfying result: the problem of computing planar convex hulls is always well conditioned. (In this respect, the convex-hull problem, with the metric we have used, is closer to the problem of computing the eigenvalues of a real symmetric matrix than to the problem of solving $A\boldsymbol{x} = \boldsymbol{b}$: the symmetric eigenvalue problem is also always well-conditioned [10]. It should be observed, however, that if a different metric is used to measure distance, then a different perturbation analysis will result. For example, if only the distance between points in the convex hull is included in the metric, then the convex-hull problem would be ill-conditioned.)

3.3.3 Step c: stability of algorithms

Just as in the case of linear equations, there exist both unstable and stable algorithms for the planar convex-hull problem, when criterion (3.2) is used. In particular, it has been shown [2] that *Graham_incremental* is unstable. This algorithm should therefore not be used (just as we should not use Gaussian elimination without pivoting to solve $A\boldsymbol{x} = \boldsymbol{b}$). On the other hand, a slight modification of the *Graham_Fortune* algorithm [7] is numerically stable, that is, the computed answer is such that it is the exact solution for a perturbed problem for which the

relative perturbation bound in problem space is at most $O(n\epsilon)$, where ϵ is the relative error of floating-point arithmetic. The algorithm uses a function called *TriangleTest* [7], first to establish lists of candidates for upper and lower chains, and secondly to decide whether or not to retain the *middle* point of possible triplets in these chains. The proof of stability depends on *both* uses of *TriangleTest* to show, for example, that slightly perturbed versions of the candidates for an upper convex chain satisfy the following condition: either they were retained and form part of an actual upper convex chain, or they were not retained but nonetheless lie above the line determined by the two points with minimum and maximum x-coordinate. The slight modification, referred to above, is to use the *a priori* bounds for finite-precision floating-point arithmetic to implement the test of a "left turn" in *TriangleTest* in a fail-safe way, so that an ambiguous point is considered to be part of a "left turn", and dropped from the computed convex hull. (This modified test is described in detail elsewhere [13], and a similar test has also been used for another purpose [14].)

3.3.4 Consequence

The consequence of these well-conditioning and stability is this: not only is it true that a stable algorithm such as *Graham_Fortune* will always produce an answer that is scarcely more in error than we should expect because of data uncertainty (this conclusion follows from stability), it is true in addition that the actual error in the computed solution is small (this follows from well-conditioning). We are in the situation illustrated in Figure 3.1. The overall situation is illustrated in Figure 3.4, where p_1 is the true problem to be solved, p_2 is the problem presented

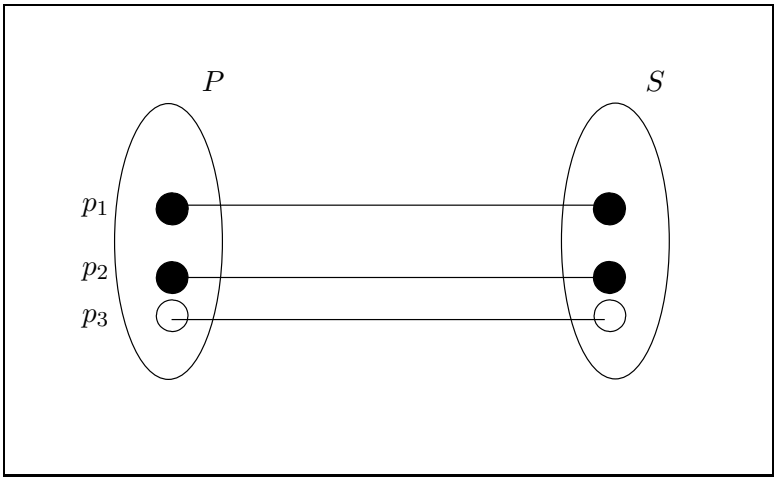


Figure 3.4: Overall situation.

to the method, and p_3 is the problem for which the method actually finds an exact solution. This is a pure backward error analysis, with a well-conditioned problem. Even if (3.2) were replaced by a criterion that rendered the problem ill-conditioned, however, it would remain true that the algorithm always produces an answer that is scarcely more in error than we should expect because of data uncertainty.

3.4 Practical implications for three-dimensional applications

The convex-hull problem discussed in the principal reference [2], and analyzed in our previous paper [1] and in Section 3.3, above, is only two-dimensional, but it has direct application to a practical three-dimensional problem. Extruded objects, discussed in Section 3.4.1, are widely used in solid-modeling systems. (Much more general extrusions than those discussed in Section 3.4.1 have been used [16]). We use the example of guaranteeing the well-formedness of extruded objects to illustrate the rigorous use of floating-point arithmetic in a geometric application. Then, in Section 3.4.2, we give a brief commentary on the use of such arithmetic for more difficult geometric problems in $\mathbb{R}^3$.

3.4.1 A simple application in $\mathbb{R}^3$: extruded objects

For $m = 0, \ldots, M-1$, let $\boldsymbol{R}^m(t)$ be a planar Bézier curve of degree ν defined by the control points $\boldsymbol{Q}^m = \{\boldsymbol{R}_0^m, \boldsymbol{R}_1^m, \ldots, \boldsymbol{R}_\nu^m\}$:

$$\boldsymbol{R}^m(t) = \sum_{i=0}^{\nu} \binom{\nu}{i} (1-t)^{\nu-i} t^i \boldsymbol{R}_i^m, \;\; 0 \le t \le 1.$$

The control points lie in $\mathbb{R}^2$, and they are assumed to satisfy the conditions $\boldsymbol{R}_\nu^m = \boldsymbol{R}_0^{m+1}$, $m = 0, \ldots, M-1$, where indices are calculated *modulo* M. Since a Bézier curve interpolates its first and last control points [15], the sequence of curves $\boldsymbol{R}^m(t), m = 0, \ldots, M-1$, defines a simple closed curve Γ in the plane, provided that no curve self-intersects, no adjacent pair of curves mutually intersect other than at prescribed endpoints, and no two distinct curves intersect (see Figure 3.5).

If we are given also a direction vector $\boldsymbol{d}$ with $\|\boldsymbol{d}\|_2 = 1$, and two scalars λ_l and λ_u, $\lambda_l \le \lambda_u$, then these data define the extruded object

$$\mathcal{E} = \{\boldsymbol{x} \in \mathbb{R}^3 : \lambda_l \le \boldsymbol{d} \cdot \boldsymbol{x} \le \lambda_u, \; \mathcal{O}[\boldsymbol{x} - (\boldsymbol{d} \cdot \boldsymbol{x})\boldsymbol{d}] \in Int(\Gamma)\},$$

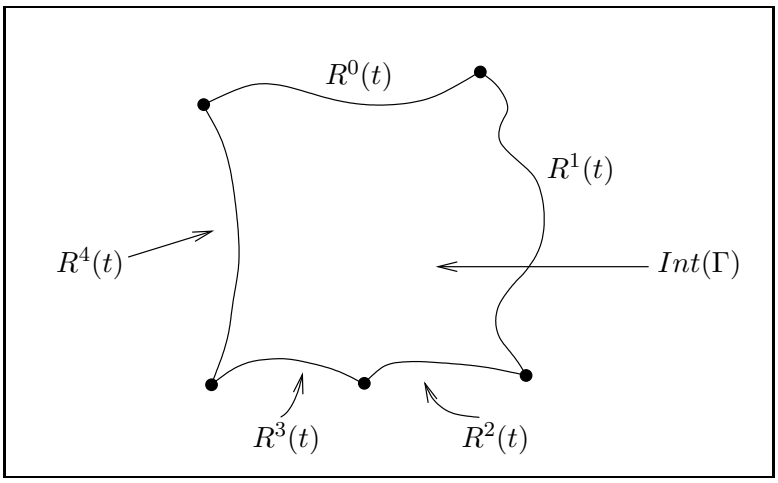


Figure 3.5: Simple closed curve formed of Bézier segments ($M = 5$).

where $\mathcal{O}$ is the rotation matrix that carries $\boldsymbol{d}$ into $[0, 0, 1]^T$, and Int denotes the interior of the simple closed curve. An extruded object is illustrated in Figure 3.6.

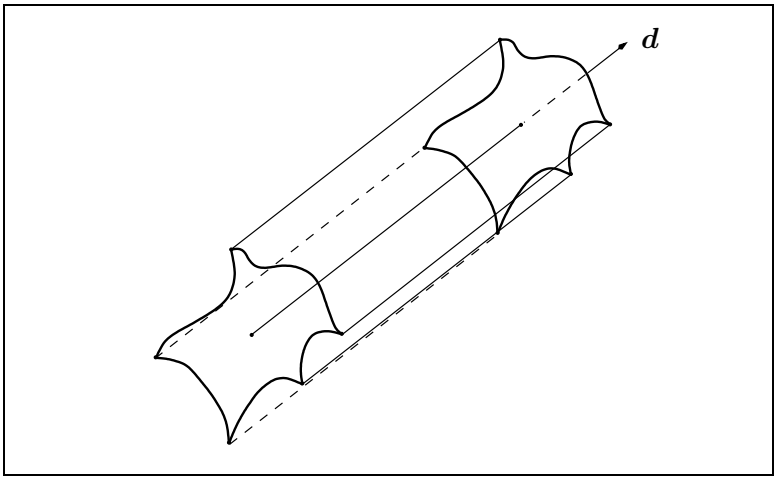


Figure 3.6: Extruded object.

To decide the question of well-formedness of $\mathcal{E}$, an algorithm must check the three conditions mentioned above:

1. no curve $\boldsymbol{R}^m(t)$ may self-intersect;
2. there may be no intersections, other than at prescribed endpoints, between adjacent pairs of curves $\boldsymbol{R}^m(w)$, $0 \leq w \leq 1$, and $\boldsymbol{R}^{m+1}(v)$, $0 \leq v \leq 1$, where $\boldsymbol{R}^m_\nu = \boldsymbol{R}^{m+1}_0$, $m = 0, \ldots, M-1$;

3. no two distinct curves $\boldsymbol{R}^{m_1}(w)$ and $\boldsymbol{R}^{m_2}(v)$, $m_1 \neq m_2$, may intersect.

Necessary and sufficient conditions for these three conditions to be satisfied have been given previously [6], along with less sharp but more tractable sufficient conditions. These are, respectively for each case:

1. Let $\boldsymbol{q} = \{\boldsymbol{R}_{i+1}^m - \boldsymbol{R}_i^m : i = 0, \ldots, \nu - 1\}$. Then a sufficient condition for non-selfintersection of $\boldsymbol{R}^m(t)$ is that $\mathbf{0} \notin conv(\boldsymbol{q})$ [6].

2. We first make the change of variables $u = 1 - w$ and rewrite $\boldsymbol{R}^m(w)$ as $\boldsymbol{R}^m(u) = \sum_{i=0}^{\nu} \binom{\nu}{i} u^i (1-u)^{\nu-i} \boldsymbol{R}_{\nu-i}^m$, so that $\boldsymbol{R}^m(u)|_{u=0} = \boldsymbol{R}^{m+1}(v)|_{v=0}$. Let $\boldsymbol{Q}'_a = \{\boldsymbol{R}_{\nu-i-j}^{m+1} - \boldsymbol{R}_i^m : 0 \leq i \leq \nu - 1, 0 \leq j \leq \nu\}$ and $\boldsymbol{Q}'_b = \{-\boldsymbol{R}_{i+j}^m + \boldsymbol{R}_{\nu-i}^{m+1} : 0 \leq i \leq \nu - 1, 0 \leq j \leq \nu\}$. Then a sufficient condition precluding intersection of the two adjacent curves is that $\mathbf{0} \notin conv(\boldsymbol{Q}'_a)$ and $\mathbf{0} \notin conv(\boldsymbol{Q}'_b)$.

3. The classical sufficient condition ensuring that distinct curves $\boldsymbol{R}^{m_1}(w)$ and $\boldsymbol{R}^{m_2}(v)$ do not intersect is that the convex hulls $conv(\{\boldsymbol{R}_0^{m_1}, \ldots, \boldsymbol{R}_\nu^{m_1}\})$ and $conv(\{\boldsymbol{R}_0^{m_2}, \ldots, \boldsymbol{R}_\nu^{m_2}\})$ of their control points should not intersect [15].

Thus, in each case, guaranteeing the sufficient condition involves solving a planar convex-hull problem. Application of Criterion 2.1* and Criterion 2.2* is simplified by using the corresponding theorems [6] Theorem 2.1* and Theorem 2.2*, which transform the two criteria into statements about the maximal perturbation of the data that will not cause unwanted intersections. These maximal perturbations are, respectively, $dist(\mathbf{0}, conv(\boldsymbol{q}))$ and $\max\{dist(\mathbf{0}, conv(\boldsymbol{Q}'_a)), dist(\mathbf{0}, conv(\boldsymbol{Q}'_b))\}$, where $dist$ denotes the separation between $\mathbf{0}$ and the convex set.

The elements defining the sets $\boldsymbol{q}$, $\boldsymbol{Q}'_a$, $\boldsymbol{Q}'_b$, $conv(\{\boldsymbol{R}_0^{m_1}, \ldots, \boldsymbol{R}_\nu^{m_1}\})$ and $conv(\{\boldsymbol{R}_0^{m_2}, \ldots, \boldsymbol{R}_\nu^{m_2}\})$ might be entered by a user indicating a pixel on a screen. Thus, the user is uncertain about the exact value of the points presented to the planar convex-hull algorithm. If $\boldsymbol{R}_i^m$ is the value stored by the system, denote by $\boldsymbol{R}_i^m + \boldsymbol{\delta R}_i^m$ a value envisaged by the user, where the double symbol $\boldsymbol{\delta R}_i^m$ denotes a vector in $\mathbb{R}^2$. The user may be ignorant of (and perhaps indifferent to) the exact value of $\boldsymbol{R}_i^m + \boldsymbol{\delta R}_i^m$, and capable only of specifying a bound on $\|\boldsymbol{\delta R}_i^m\|_2$. In the present context, it is reasonable to suppose that the best bound available for $\|\boldsymbol{\delta R}_i^m\|_2$ is, say, $10^{-3}\|\boldsymbol{R}_i^m\|_2$.

Additional uncertainty in the input data (*i.e.*, the input data for the convex-hull algorithm) is added by the numerical calculations necessary to compute the elements of $\boldsymbol{q}$, $\boldsymbol{Q}'_a$, $\boldsymbol{Q}'_b$,

$conv(\{\boldsymbol{R}_0^{m_1}, \ldots, \boldsymbol{R}_\nu^{m_1}\})$ and $conv(\{\boldsymbol{R}_0^{m_2}, \ldots, \boldsymbol{R}_\nu^{m_2}\})$, and to perform the rotation contained in the definition of $\mathcal{E}$. Bounding this additional uncertainty can be done using standard *a priori* bounds on floating-point arithmetic [10], and in our case, might add relative uncertainty on the order of 10^{-14}, assuming that double-precision floating-point arithmetic (relative error $\epsilon \cong 10^{-16}$) has been used. Note that the uncertainty associated with the input data is different for each of the convex-hull problems to be solved. Similarly, the uncertainty implicitly associated with the input data, by Fortune's stability proof, will also be different for each convex-hull problem. But if each sufficient condition is satisfied independently, then $\mathcal{E}$ will be well-formed.

Independently of the exact additional uncertainty, the total will overwhelm the $O(n\epsilon)$ uncertainty introduced by *Graham_Fortune* (see Section 3.3.3). In the largest of our convex-hull problems, we have $n = \nu(\nu + 1)$. Thus, for example, if cubic splines are used, $\nu = 3$ and $n = 12$. Use of exact arithmetic would permit us to eliminate the $O(n\epsilon)$ uncertainty, but not the input uncertainty, which is larger by a factor of many billions. And the user must live with the effects of the input uncertainty in any event.

3.4.2 Other problems

The topic of providing an analysis of the sort described in Section 3.3 for floating-point-arithmetic implementations for more complicated problems such as Boolean operations on trimmed-NURBS representations, has been much studied over the last two decades; whether this will prove tractable, however, remains an open question [17, 18]. It is quite likely that certain parts of the necessary algorithms will require implementation using more expensive arithmetics. There is no claim in this paper that ordinary floating-point arithmetic will always be sufficient—as stated in Section 3.1.2, we only claim that it *may* be sufficient, in spite of the existence of unstable algorithms such as those discussed above.

A framework for a backward error analysis, suitable for the case of Boolean operations on objects represented by internally inconsistent trimmed-NURBS representations, was given elsewhere [18]. The fundamental difficulty in providing theorems in this case comes from the problem of topology resolution [19]. There are many good algorithms for computing intersections between NURBS surfaces [19, 20], but to rigorously account for short intersection edges between surfaces, and inconsistent decisions based on the use of small numerical tolerances, is difficult, especially in the case where several surfaces are involved. On the other hand, it has been shown that certain computed intersections of surfaces can be viewed as the exact inter-

section of slightly perturbed surfaces [21]. This is an essential ingredient for a backward error analysis for Boolean operations on trimmed-NURBS representations. Furthermore, rigorous backward-error analyses are more easily obtained in the simpler case of objects represented by locally-planar subdivision surfaces [22].

3.5 Conclusion

In order to prove theorems about the adequacy of numerical algorithms in computational geometry, we must define how to measure adequacy. Furthermore, in the case where data is uncertain, it is worthwhile to do a perturbation analysis, and seek stable solution methods, in order to perform a backward error analysis. Carrying out these steps in the context of the planar convex-hull problem shows that the numerical difficulties described in the principal reference [2] are unexceptional. Furthermore, these results carry over to simple applications in three-dimensional solid modeling. On the other hand, whether it is possible to carry out a backward error analysis for floating-point arithmetic, for problems such as Boolean operations for trimmed-NURBS solids, remains an open question.

References

[1] Jiang, D. and Stewart, N. F. Backward error analysis in computational geometry, *Proc., Part I, Computational Science and Its Applications ICCSA* Glasgow, UK, May 8-11 2006. *Lecture Notes in Computer Science* LNCS 3980, (2006) pp. 50–61.

[2] Kettner, L., Mehlhorn, K., Pion, S., Schirra S. and Yap, G. Classroom examples of robustness problems in geometric computations. *ESA '04: Proceedings of the 12th Annual European Symposium on Algorithms*, Bergen, Norway, (Springer, Sept. 2004) pp. 702–713.

[3] IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std 754–1985, IEEE, New York, NY, (1985).

[4] Requicha, A. A. G. Representations for rigid solids: theory, methods, and systems. *Computing Surveys* (12), No. 4, (1980) 437–464.

[5] Johnson, R. H. Solid Modeling, Second Edition, CAD/CIM Alert and North-Holland, Amsterdam, (1986).

[6] Andersson, L.-E., Peters, T. J. and Stewart, N. F. Selfintersection of composite curves and surfaces, *Computer Aided Geometric Design* 15, No. 5, (1998) 507–527.

[7] Fortune, S. Stable maintenance of point set triangulations in two dimensions, *Proc. of the 30th annual IEEE Symp. Foundations of Computer Science* Vol. 30, (1989) 494–499.

[8] Graham, R. L. An efficient algorithm for determining the convex hull of a finite planar set, *Information Processing Letters*, 1:132–133 (1972).

[9] Kahan, W. M .A survey of error analysis. *IFIP '71*, North Holland, (1971) 1214–1239.

[10] Wilkinson, J. H. The Algebraic Eigenvalue Problem. Clarendon Press, Oxford (1965).

[11] Jänich, K. Topology. Springer Verlag, New York (1984).

[12] Hoffmann, C. M. and Stewart, N. F. Accuracy and semantics in shape-interrogation applications, *Graphical Models* 67, No. 5, (Sept. 2005) 373–389.

[13] Jaromczyk, J. W. and Wasilkowski, G. W. Computing convex hulls in a floating point arithmetic, *Computational Geometry* 4, (1994) 283–292.

[14] Fortune, S. Numerical stability of algorithms for 2D Delaunay triangulations, *International Journal of Computational Geometry and Applications* (5), No. 1, (1995) 193–213.

[15] Farin, G. Curves and Surfaces for CAGD, Third Edition. Academic Press (1993).

[16] Snyder, J. M. Generative Modeling for Computer Graphics and CAD. Academic Press (1992).

[17] Farouki, R. Closing the gap between CAD model and downstream applications, *SIAM News* (32), No. 5 (June 1999).

[18] Andersson, L.-E., Stewart, N. F. and Zidani, M. Error analysis for operations in solid modeling in the presence of uncertainty. *SIAM Journal on Scientific Computing* (29), No. 2, (2007) 811–826.

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

[20] Patrikalakis, N. and Maekawa, T. Shape Interrogation for Computer Aided Design and Manufacturing, (Springer 2002).

[21] Song, X., Sederberg, T. W., Zheng, J., Farouki, R. T. and Hass, J. Linear perturbation methods for topologically consistent representations of free-form surface intersections, *Computer Aided Geometric Design* (21) (2004) 303–319.

[22] Jiang, D. and Stewart, N. F. Robustness of Boolean operations on subdivision-surface models, *Dagstuhl seminar 08021*, January 6-11, 2008. *Dagstuhl Research Online Proceedings (DROPS):* http://drops.dagstuhl.de, (2008).

Chapter 4

Reliable joining of surfaces for combined mesh-surface models

The joining (or merging) operator is a very important primitive operator for Boolean operations. It can be applied to different geometric representations, including subdivision-surface models and trimmed-surface models. In this work, we study the latter representation, which is a composite model containing both a NURBS surface patch and a triangular mesh patch. A naively designed joining operator can produce very poor results, e.g. triangles along the target joining curve in the final result (triangular mesh) can be turned upside down by the joining process, even in the case when maximum auxiliary information is available. Our motivation for this work is to seek a reliable joining algorithm taking into account a normal error criterion.

To evaluate the result produced by our joining algorithm, and also to guide the joining process, we first define two error measures, the *absolute error* and the *normal-vector error*. The Whitney extension theorem is then used as a theoretic base to perform the joining. Its use guarantees that in the joined mesh patch, the absolute error will be no greater than that already present along the boundary of the input mesh patches, and its slope will be smaller than or equal to the maximum slope along the boundary of the two input mesh patches. Two different cases can be treated with our algorithm, based on the availability of an explicit common edge curve which represents the boundary between the two patches to be joined. Implemented results are also presented.

The preliminary work that deals with a single joining segment was presented in the IMCS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numer-

ics (SCAN), Duisburg, Germany, September, 2006. The complete work included here was presented at the 21st European Conference on Modelling and Simulation (ECMS), Prague, Czech Republic, June 2007, and appeared in the conference proceedings.

The main contributions of this work are:

1. we propose to use the Whitney extension theorem as the theoretical base for our joining algorithm.

2. a joining algorithm is proposed to merge combined mesh-surface patches, which can deal with two different cases based on the availability of certain auxiliary information.

3. two error measures (*i.e.* absolute error and normal-vector error) are proposed to guide the joining process, and evaluate the quality of the joint result surface.

Small corrections:

1. In page 53 line -3: should read “approximately 30 floating-point operations for each piecewise linear segment”.

2. Two footnotes have been added (pages 49, 52).

Reliable joining of surfaces for combined mesh-surface models

D. Jiang N. F. Stewart

Département d'informatique et de recherche opérationnelle

Université de Montréal

appeared in

Proceedings of 21st European Conference on Modelling and Simulation (ECMS),

pages 297-303, 2007.

Abstract

Algorithms to join two mesh patches along an edge are of immediate practical interest in the context of higher-level operations on models of objects formed by such mesh patches. Such models are widely used in graphical visualization and simulation, shape interrogation, and other areas. Thus, there are now available methods to join two subdivision surfaces along a common edge curve, as well as methods to join mesh patches that approximate given trimmed-surface patches. The latter problem is studied in this paper.

The auxiliary information available to the algorithm, in the context of surface joining, varies, depending upon circumstances. In particular, it may or may not be true that an explicit common edge curve, representing the boundary between the two patches to be joined, is available as part of the data. Even in the case, however, when maximal auxiliary information is available algorithms are not necessarily reliable. For example, methods that do not use normal-vector error criteria, to measure the discrepancy between the surface patch and the associated mesh patch, can produce poor results, due to large changes in the normal direction of a triangle near the mesh boundary. It is even possible to give examples where the triangles near the joined boundary are turned upside down by the joining process, so that computed meshes self-intersect. In this paper an algorithm is presented that uses a proxy for a normal-vector error criterion, and the Whitney extension theorem, to produce reliable algorithms. Examples are given, and an implementation is described.

Keywords:

surface mesh, joining, graphical simulation, shape-interrogation models, normal-vector error

4.1 Introduction

This paper is concerned with the problem of the reliable joining of surface meshes used in combined mesh-surface models. Such models are of interest for graphical visualization of solid objects, shape interrogation, computer-aided design, and vision [1, 2, 3, 4, 5, 6, 7]. The joining process is sometimes referred to as *sewing* [1]. The main novel aspect of the work is the use of normal-vector criteria, described below, to prevent folding of edges during the joining process.

A *mesh patch* is a surface made up of non-degenerate triangles lying in $\mathbb{R}^3$. Algorithms to join two mesh patches along a common edge are of immediate practical interest in the context of higher-level operations on objects formed by such mesh patches. For example, methods have been given to join two subdivision patches along a common edge curve, specified in $\mathbb{R}^3$. In particular, *combined subdivision surfaces* [8] were designed for this purpose, and *dynamic subdivision surfaces* [9] may be used to produce subdivision surfaces with hard edges along a given curve in space. Similarly, methods are available [1], [10, Sec.3.4] to join together mesh patches that approximate given trimmed-surface patches lying in $\mathbb{R}^3$. It is the latter problem (surface-mesh joining) that is studied in this paper.

The auxiliary information available to the algorithm, in the context of surface-mesh joining, may vary. Mesh solids formed by a trimmed-surface model coupled with a triangular mesh are used in solid modeling [1, 3, 5] and in graphical simulation [1, 2, 4]. In the latter case, the mesh model may be carried along with the surface model, or computed adaptively during rendering, given the current camera position. The trimmed-surface model is illustrated in Figure 4.1.

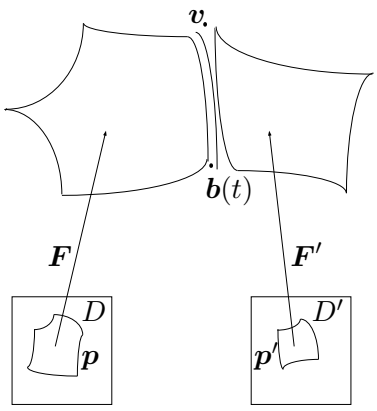


Figure 4.1: Two adjoining trimmed patches in surface model, with boundary curve $\boldsymbol{b}(t)$, $t \in [0, 1]$.

The parametric domain D is delimited by a collection of p-curves (a typical p-curve is denoted here by $\boldsymbol{p}$), and the restriction of the mapping $\boldsymbol{F}$ to D defines the trimmed patch in R^3. In addition, explicit boundary information may also be present. Sometimes [3, 11, 12] this may take the form of explicit curves $\boldsymbol{b}(t)$ taking values in R^3, due to the convenience of having such explicit representations available. This curve is analogous to the common edge curve specified for combined subdivision surfaces. Alternatively, explicit boundary information in R^3 may be represented in other ways; for example, it may be represented approximately by scan conversion [1].

Even with an explicit boundary curve provided, joining algorithms are not necessarily reliable, and it is this fact that led to the development of the algorithms described below.

We present joining algorithms for both cases: when an explicit curve $\boldsymbol{b}(t)$ is provided, and when it is not. The algorithms described are based on the use (as a supplement to absolute error criteria) of normal-vector error criteria [13, 14, 15] for the discrepancy between the surface patch and the mesh-patch. A difficulty, with algorithms that do not use such criteria, is that they may cause large changes in the normal direction of a triangle near the joined boundary, which may in turn introduce undesired visual effects. In fact, it could even happen that triangles near the boundary are turned upside down, so that computed meshes self-intersect. The nature of the difficulty is illustrated in Figure 4.2, in the case where joining moves mesh vertices on the basis of interpolation along a polygonal path that is not a straight line. In both illustrations, vertex l_1 is paired with vertex r_1, and vertex l_4 is paired with vertex r_2. The intervening joining vertices are obtained by joining the midpoints of pairs of points obtained by linear interpolation along the polylines l_1-l_2-l_3-l_4 and r_1-r_2. In the first illustration, this leads to a well-behaved triangulation, but in the second illustration, the position of the vertex l_4 is different: it is further towards the interior of the segment r_1-r_2, but still within the joining tolerance, relative to r_1-r_2.

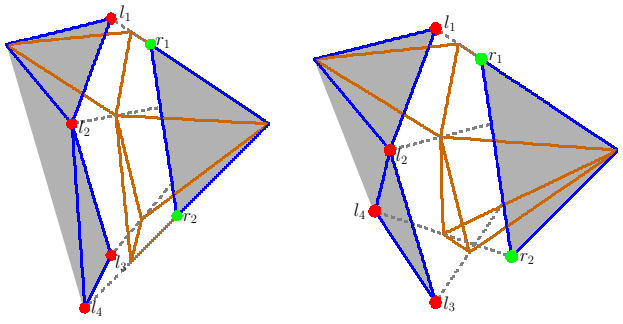


Figure 4.2: Sewing based on midpoints of pairs of points interpolated along mesh edges.

This phenomenon is called "folding" [1], and can result in a mesh triangle that has flipped, as in the second illustration of Figure 4.2. Such phenomena can be avoided by using normal-vector criteria, and in fact, if the normals of the triangles in the mesh-patch can be bounded, they can be used to rigorously exclude the possibility of extraneous intersections between neighboring mesh-patches [16, 17, 18]. In the context of graphical simulation, it is clearly of interest to do so.

The algorithms presented here use the Whitney extension theorem [19] to ensure that a proxy for the normal-vector error (defined below), and the absolute error, should not be any larger than the corresponding errors already present along the edges of the patch. Thus, in addition to avoiding the difficulty described in the previous paragraph, the procedure smooths the input mesh patches, in the sense just described of error minimization. The algorithms apply to the case of general trimmed patches, and we describe an implementation.

Whitney extension can be viewed as a way to perform transfinite interpolation between boundary curves. Amongst many other applications, it has been suggested for use as a meshing method in [5]. The algorithms below will adjust the vertices of the input mesh patch in a way that constrains them to lie in a transfinite interpolant defined by Whitney extension.

Numerical properties of one of our algorithms were discussed, in the special case of planar patches with straight-line boundaries, in [20].

Related areas of work include mesh simplification (finding a "... concise, yet geometrically faithful ... representation of a surface ..." [14, Sec. 1]), remeshing [14, Sec. 1.1] [15], [5, 21] and mesh fairing [22]. A good overview is given in [14]. Yet other work deals with computation of meshes over imperfect geometry [3, 23], and methods for mesh repair [2, 4, 24, 25].

Other work on meshing can be related to ours in another way, namely, by examining the metrics used to compare surfaces. The general concept of the absolute error in a mesh, relative to a given surface, is ubiquitous (see for example [26]). Again, the reference [14] gives a good overview. As already mentioned, other authors [13, 15] have introduced normal-vector criteria similar to ours. For example, in [15], although priority is given to other mesh-smoothness criteria, it is verified from time to time that a criterion, similar to the mean-square criterion discussed in Section 4.2, is not above a certain threshold. Somewhat different criteria are used in other applications. For example, in the context of snakes on triangular meshes, [27] refers to bending-energy and curvature-distribution criteria that are different from but nonetheless similar to the height-field-slope criterion introduced below.

4.2 Error criteria to measure mesh-patch quality

One measure of the quality of a mesh patch M is the *absolute error*. Let $\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_n \in R^3$ be the vertices of M, and $T_1, \ldots, T_r$ its triangles, where $T_j =< \boldsymbol{\nu}_{i_1}, \boldsymbol{\nu}_{i_2}, \boldsymbol{\nu}_{i_3} >, 1 \leq i_1, i_2, i_3 \leq n$. We assume that the Jacobian of the mapping $\boldsymbol{F}$ is of full rank, i.e., the rank is equal to 2. Let

$$\boldsymbol{n}(u,v) = (\boldsymbol{F}_u(u,v) \times \boldsymbol{F}_v(u,v))/\|\boldsymbol{F}_u(u,v) \times \boldsymbol{F}_v(u,v)\|$$

be the unit normal of the surface $\boldsymbol{F}$ at (u,v), and let the height $\eta(u,v) \in R$ be the scalar such that

$$\boldsymbol{M}(u,v) = \boldsymbol{F}(u,v) + \eta(u,v)\boldsymbol{n}(u,v) \in |M|,$$

where $|M|$ denotes the mesh viewed as a subset of R^3, if a unique such η exists. We suppose in fact that for all mesh patches considered, the mapping

$$\boldsymbol{M}^{-1} : |M| \mapsto [0,1]^2$$

is well-defined and injective. Thus, it is assumed that for any $\boldsymbol{m} \in |M|$,

$$|\eta| = dist(\boldsymbol{m}, \boldsymbol{F}) \doteq \min_{\boldsymbol{y} \in \boldsymbol{F}} \|\boldsymbol{m} - \boldsymbol{y}\|$$

is uniquely defined, and furthermore, that the corresponding point (u,v) is well defined and lies in $[0,1]^2$. (It follows that the mapping $\boldsymbol{F}$ itself must be injective, at least on the part of the domain of interest. Note also that the symbol $\boldsymbol{F}$ has been used to denote both the mapping and the image of the mapping, which is a pointset.)

A possible definition of the absolute error in M is the supremum of $|\eta|$ over $I \subseteq [0,1]^2$, where I is the inverse image of $|M|$. Meshes are in practice close enough to $\boldsymbol{F}[D]$ that the assumption above, that $|\eta|$ is well defined, does not present a problem, provided $I \subseteq [0,1]^2$. (The mesh must be close relative to the local minimum normal curvature of $\boldsymbol{F}$.) On the other hand, there is a theoretical difficulty in simply defining the absolute error to be

$$\sup_{(u,v) \in I} |\eta| \tag{4.1}$$

because there is nothing in this criterion to force full coverage of the surface patch by the mesh. For example, a degenerate mesh M consisting of a single vertex lying in $\boldsymbol{F}[D]$ would produce

an error of zero. As observed in [14, Sec. 2.1], use of (4.1) amounts to using a one-sided version of the Hausdorff metric. In spite of the difficulty we have just described, this approach is often used in practice [14], and we will do so here. The coverage of practical meshes is usually quite good.

We also assume that D lies strictly inside $[0,1]^2$, *i.e.*, that the patch is trimmed on all sides. There is no theoretical problem in the opposite case, since normally [26] the mapping $\boldsymbol{F}$ is defined outside $[0,1]^2$. If, however, the inverse image of a point in $|M|$ lies outside $[0,1]^2$, there may be numerical difficulties in the calculation of η.

A second measure of the quality of M is the *normal-vector error*, defined here as the largest, over all triangles T_j, of the maximum slope (in absolute value) of the height field. Let I_j be the inverse image of T_j under $\boldsymbol{M}$, and let L_j be the smallest value of L for which η satisfies the Lipschitz condition

$$|\eta(\boldsymbol{p}_1) - \eta(\boldsymbol{p}_2)| \leq L \cdot \|\boldsymbol{p}_1 - \boldsymbol{p}_2\|$$

for all points $\boldsymbol{p}_1 = (u_1, v_1)$ and $\boldsymbol{p}_2 = (u_2, v_2)$ in I_j. Our second criterion is then $\max_j L_j$.

To relate this criterion to similar normal-vector measures introduced elsewhere [13, 14, 15], we note that

$$\sup_{(u,v)} \|\boldsymbol{n}(u,v) - \boldsymbol{n}_j\|$$

(where $\boldsymbol{n}_j$ is the unit normal of the triangle T_j, and the supremum is taken over I_j) is analogous to the mean-square norm [14, Sec. 2.3.1] [15] of $\boldsymbol{n}(u,v) - \boldsymbol{n}_j$, normalized to allow for the area of the region I_j:

$$\|\boldsymbol{n} - \boldsymbol{n}_j\|_2 \doteq \left[\frac{1}{Area(I_j)} \int_{I_j} \|\boldsymbol{n}(u,v) - \boldsymbol{n}_j\|^2 du dv\right]^{1/2}.$$

It is, however, a more strict criterion, since

$$\|\boldsymbol{n} - \boldsymbol{n}_j\|_2 \leq \sup_{(u,v)} \|\boldsymbol{n}(u,v) - \boldsymbol{n}_j\|.$$

On the other hand, $\sup_{(u,v)} \|\boldsymbol{n}(u,v) - \boldsymbol{n}_j\|$ and the criterion L_j, defined above, are equivalent metrics[1], a fact which follows from our assumptions about the Jacobian of $\boldsymbol{F}$, and the Implicit Function theorem. This justifies the terminology "normal-vector error" for the maxi-

[1]Two metrics are equivalent if the unit sphere of each can be contained in the other by multiplying a positive constant.

mum slope of the height field.

It was stated in Section 4.1 that our algorithms control only a proxy for the normal-vector error. This proxy is obtained as follows. First of all, the slope of η on I_j is replaced by the slope measured only between the three corner points of I_j. This process can increase the error in the case of long thin triangles, but the difficulty can be avoided by mesh-edge splitting. (The error estimates given below, in Section 4.3.4, take account of the potential error introduced in this way, *i.e.*, it is not assumed that mesh-edge splitting has been used to reduce the error.) Secondly, in order to reduce computational cost, we estimate $\max_j L_j$ by using the Whitney theorem with the ordinary Euclidean norm of $\boldsymbol{p}_1 - \boldsymbol{p}_2$, over all of $I = \bigcup_{j=1}^{r} I_j$, which could in principle (see Section 4.3.1) lead to the minimization not of $\max_j L_j$ but, rather, the minimization of a certain upper bound for $\max_j L_j$.

4.3 Joining algorithms

As mentioned in the introduction, joining algorithms that do not use normal-vector criteria may cause large changes in the normal direction of a triangle near the boundary. The nature of the difficulty was shown in the second illustration in Figure 4.2. Thus, even though the input mesh patches satisfy the assumptions of Section 4.2, and have small height η along the edges of the two patches, folding may occur within (or approximately within) the curvilinear surface $\boldsymbol{F}$. In this section we present algorithms that avoid this problem, and which, at the same time, smooth the mesh. Both of these are of obvious importance in graphical simulation. An example will be given below, in Section 4.4, which shows the possible ill effects of folding.

We begin by giving a brief summary of Whitney extension, which is used in both of the algorithms presented. We then give an algorithm in the case when the boundary curve $\boldsymbol{b}(t)$ is provided as part of the input, and in a subsequent subsection, we deal with the opposite case, by constructing ourselves a boundary curve $\boldsymbol{b}(t)$ based on the input mesh patches. The algorithms adjust the mesh vertices to ensure that the proxy, mentioned above, for the normal-vector error, and the absolute error, should not be any larger than the errors already present along the edges of the input patch. In fact, they will not be any larger than those associated with the boundary curves $\boldsymbol{b}(t)$ bordering the mesh patch. This of course represents only *part* of the error present in the input data: the error in the edge of the input mesh patch itself could in principle be even larger (and this fact makes our bound even more attractive).

4.3.1 Whitney extension

As mentioned at the end of Section 4.1, our algorithms adjust the vertices of mesh patches in a way that constrains them to lie in a transfinite interpolant defined by Whitney extension. This process is referred to as *reprojection* in the algorithm outlines given below. The reprojected mesh interpolates the curves $\boldsymbol{b}(t)$, and the assumption of injectivity of $\boldsymbol{M}^{-1}$, at the beginning of Section 4.2, includes in particular the assumption that we can compute the height $\eta(u_0, v_0)$ corresponding to a given $\boldsymbol{b}(t_0) \in R^3$, where $(u_0, v_0) = \boldsymbol{M}^{-1}(\boldsymbol{b}(t_0))$. This is done, as for vertices in a given mesh patch, by computing $dist(\boldsymbol{b}(t_0), \boldsymbol{F})$. (As in Section 4.2, the assumption requires that $\boldsymbol{b}(t_0)$ be close to $\boldsymbol{F}[D]$, relative to the local minimum normal curvature of $\boldsymbol{F}$.)

Now, suppose given a mesh patch M with m edges, and corresponding boundary curves $\boldsymbol{b}^k(t), k = 0, \ldots, m-1, t \in [0,1]$. Let $\epsilon(\boldsymbol{p})$ be the height $\eta(\boldsymbol{M}^{-1}(\boldsymbol{b}^k(t))$ defined for a point $\boldsymbol{p} \in \partial R$, the inverse image of $\{\boldsymbol{b}^k(t) : k = 0, \ldots, m-1, t \in [0,1]\}$. We suppose that ∂R is the boundary of a well-defined region $R \subseteq [0,1]^2$.

The optimality of the reprojection obtained by Whitney extension can be described as follows. We view the height associated with the curves $\boldsymbol{b}^k(t)$ as a discrepancy between the surface data $\boldsymbol{F}$ and the boundary data. Let $\epsilon(\boldsymbol{p})$ be the discrepancy $\eta(\boldsymbol{p})$ defined by $\boldsymbol{M}^{-1}(\boldsymbol{b}^k(t)) = \boldsymbol{p}$, *i.e.*, the discrepancy defined by the boundary curves $\boldsymbol{b}^k(t)$ for $k = 0, \ldots, m-1$ and $t \in [0,1]$. Then, if the reprojected mesh (denoted $\bar{M}$) is to interpolate the boundary curves, the maximum absolute discrepancy $|\epsilon(\boldsymbol{p})|$ of $\bar{M}$, measured over *all* of R, cannot be less than $\max_{p \in \partial R} |\epsilon(\boldsymbol{p})|$, and the maximum slope of the reprojected mesh over all of R cannot be less than the slope on ∂R, defined by

$$L = \sup_{\boldsymbol{p}_1, \boldsymbol{p}_2 \in \partial R, \boldsymbol{p}_1 \neq \boldsymbol{p}_2} \frac{|\epsilon(\boldsymbol{p}_1) - \epsilon(\boldsymbol{p}_2)|}{\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|}. \tag{4.2}$$

This follows from the fact that $\partial R \subseteq R$.

Now, a continuous extension of $\epsilon(\boldsymbol{p})$ from ∂R to R will be called *Whitney* if it satisfies the Lipschitz condition

$$|\epsilon(\boldsymbol{p}_1) - \epsilon(\boldsymbol{p}_2)| \leq L \cdot \|\boldsymbol{p}_1 - \boldsymbol{p}_2\|$$

everywhere on R (and not just on the boundary ∂R). There exist [29] a bracketing pair of extensions $l(\boldsymbol{p})$ and $u(\boldsymbol{p})$ that are Whitney, and such that for any extension $\epsilon(\boldsymbol{p})$ that is Whitney, we have

$$l(\boldsymbol{p}) \leq \epsilon(\boldsymbol{p}) \leq u(\boldsymbol{p}), \quad \boldsymbol{p} \in R.$$

(The explicit definitions of $l(\boldsymbol{p})$ and $u(\boldsymbol{p})$ are given below, in (4.3) and (4.4).) Furthermore, if

we take the average

$$a(\boldsymbol{p}) = \frac{1}{2}[l(\boldsymbol{p}) + u(\boldsymbol{p})],$$

then $a(\boldsymbol{p})$ is Whitney, and

$$|a(\boldsymbol{p})| \leq \sup_{q \in \partial R} |\epsilon(\boldsymbol{q})|, \quad \boldsymbol{p} \in R.$$

Thus, using $a(\boldsymbol{p})$ to reproject the mesh, as we do below, provides an extension that has absolute error no greater than that already present along the boundary ∂R, and which has slope[2] no greater than that already imposed by the slope of $\eta(\boldsymbol{p})$ on ∂R. It is therefore optimal (and the errors minimal) in the sense that we cannot do better.

In [28, Sec. 3.5] an alternate but computationally more expensive version of the Whitney theorem is given, appropriate for severely non-convex domains. There is a possibility in such cases, if the ordinary Whitney theorem is used, of over-estimation of $\max_j L_j$. The practical risk is small. Also, there exist [19] extensions that are smoother than the C^0-continuous extension described above, when the data along the boundary is smooth. These might be used to permit specification of joining with a given level of continuity. We have not explored this possibility.

4.3.2 Case 1: The $\boldsymbol{b}^k(t)$ are provided as input

The outline of the joining algorithm, in the case when the boundary curves $\boldsymbol{b}^k(t)$ are provided as part of the input, is as follows:

1. Project the vertices $\boldsymbol{\nu}_i$ of the input mesh M into $[0,1]^2$ in the u-v domain, to produce a *projected mesh*. (There is of course an approximation involved here, since the inverse images of triangles T_j are typically curvilinear sets in the u-v domain.)

2. Project a piecewise-linear approximation of each $\boldsymbol{b}^k(t)$ into $[0,1]^2$ in the u-v domain.

3. Remove a sufficient number of peripheral triangles from the projected mesh (in the u-v domain) to guarantee that the projected mesh does not extend beyond the projection of the boundary curves $\boldsymbol{b}^k(t)$, but with at least one layer of triangles removed from the periphery of the projected mesh. The remaining part of the projected mesh will be referred to as the *central mesh*. See Figure 4.3.

[2] Here the slope is not the slope along the boundary: there could be variation across the interior of R.

4. Triangulate the region between the projection of the boundary curves and the central mesh. (This will be referred to as the triangulation of the *external region*. See Figure 4.3.)

5. Reproject the vertices of the combined mesh (the central mesh and the triangulation of the external region) to R^3 using Whitney extension, as described in Sec. 4.3.1.

6. Merge the reprojected combined mesh, along the joint boundary (in R^3) between the two parts of the combined mesh, to obtain $\bar{M}$.

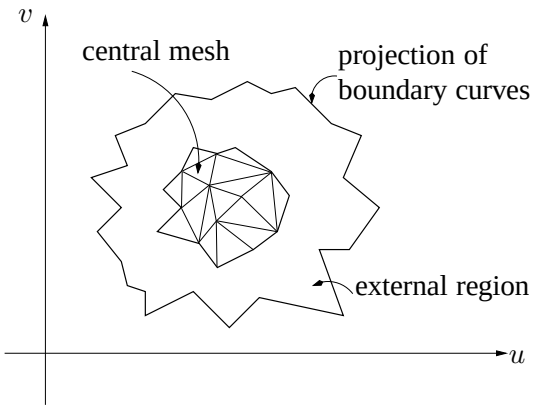


Figure 4.3: Meshing domain.

The projection of the input mesh (step 1), and of the curves $\boldsymbol{b}^k(t)$ (step 2), can be dealt with in several ways [6, 30, 31, 32]; here we simply used the Fletcher-Reeves gradient algorithm provided in the GNU Scientific Library [33].

The reprojection (step 5) requires calculation of the functions $l(\boldsymbol{p})$ and $u(\boldsymbol{p})$, mentioned in Sec. 4.3.1. The functions $l(\boldsymbol{p})$ and $u(\boldsymbol{p})$ are defined by

$$l(\boldsymbol{p}) = \sup_{\boldsymbol{q}\in\partial R} \{\epsilon(\boldsymbol{q}) - L \cdot \|\boldsymbol{p} - \boldsymbol{q}\|\}, \;\; \boldsymbol{p} \in R, \tag{4.3}$$

and

$$u(\boldsymbol{p}) = \inf_{\boldsymbol{q}\in\partial R} \{\epsilon(\boldsymbol{q}) + L \cdot \|\boldsymbol{p} - \boldsymbol{q}\|\}, \;\; \boldsymbol{p} \in R, \tag{4.4}$$

[29]. Due to the use of the piecewise-linear approximation (step 2), the calculation of the supremum in the definition of $l(\boldsymbol{p})$, and the infimum in the definition of $u(\boldsymbol{p})$, together require only 8 floating-point operations for each piecewise-linear segment.

The triangulation of the external region (step 4) is done using a slightly modified version of Ruppert's Delaunay refinement algorithm [34], namely the variant [35]. Suppose that the

triangulation producing the projection mesh is done using the same algorithm. Then, because we remove at least one layer of triangles in step 3, it follows that the minimum angle in the boundary of the external region is at least $\theta = 26.45$ degrees, provided that this condition is also satisfied by the projections of the $\boldsymbol{b}^k(t)$. Consequently, it follows [35] under these hypotheses that the minimum angle in the triangulated external region is no smaller than $\arctan[(\sin\theta)/(2-\cos(\theta)]$, which is approximately 21.96 degrees.

The merging required in step 6 refers to triangle splitting when there are extra vertices along the boundary, between the two parts of the combined mesh, arising from the triangulation of the external region.

4.3.3 Case 2: Certain of the $\boldsymbol{b}^k(t)$ are not provided as input

The procedure in the case when certain of the $\boldsymbol{b}^k(t)$ are not provided is exactly the same as in Sec. 4.3.2, except that before projecting a piecewise-linear approximation of the curves $\boldsymbol{b}^k(t)$, it may be necessary to calculate surrogates for the missing boundary curves. Note that we need $\boldsymbol{b}^k(t)$ (or a surrogate) for all k, even if no mesh patch is to be joined along certain edges.

If a curve $\boldsymbol{b}^k(t)$ is present, for a given k, it is used as in Sec. 4.3.2.

If $\boldsymbol{b}^k(t)$ is not present, for a given k, then there are two possibilities. If there is not an adjoining mesh along edge k, then we simply use the boundary of the input mesh to compute ∂R along that edge. If there is an adjoining mesh along edge k, then we compute a piecewise-linear median polyline, deleting loops if necessary. Folding causes no problem here: there is no requirement that the external region be convex in order to triangulate it.

4.3.4 Error estimates

Use of the Whitney theorem (step 5) in Sec. 4.3.2 guarantees that the slope of the reprojected mesh points, between corners of the combined-mesh triangles, will be less than or equal to the value of L along the boundary of the mesh. It does not, however, guarantee that the minimum slope of the actual triangles in the combined mesh will be less than or equal to L, as can be seen by consideration of a long thin triangle. On the other hand, if the triangulation in the u-v domain has minimum angle equal to 21.96 degrees, then it can be shown that the cosine of the angle of inclination, of a triangle in the reprojected mesh, is greater than or equal to $\{(1+L^2)[1+\left(\frac{2L}{\sin 21.96}\right)^2]\}^{-1/2}$. This follows from a straightforward trigonometric argument using spherical coordinates. The value of $\sin 21.96$ is approximately 0.384.

The problem just mentioned, related to long thin triangles, can be avoided if a long edge of such a triangle is split, and the Whitney reprojection calculated at the inserted vertex. Note however that the worst-case risk of neglecting to do the mesh-edge split is that the slope of the triangle could be unnecessarily large. There is no danger of a flipped triangle (Figure 4.2).

4.4 Computational examples

4.4.1 Examples illustrating the two algorithms

In the accompanying figures, examples of the use of the joining algorithms are given. The examples involve joining of trimmed patches: the trimmed patch illustrated in Figure 4.4 is exactly the input patch shown in the upper right corner of each of Figure 4.5 and Figure 4.6. The second input patch, in the upper left corner of Figure 4.5 and Figure 4.6, is, similarly, a trimmed patch obtained from a larger untrimmed surface (not shown). The joined patches are shown in the lower part of Figure 4.5 and Figure 4.6, respectively.

Figure 4.7 shows two input patches with folding present. The result of joining by means of linear interpolation along polylines, as described in Section 4.1, is shown in Figure 4.8. The result of using the algorithm of this paper is shown in Figure 4.9.

The triangulations of the input trimmed patches were obtained using Maya [4]. The triangulations of the exterior regions were obtained, as explained in Section 4.3.2, using a variant of the Ruppert algorithm.

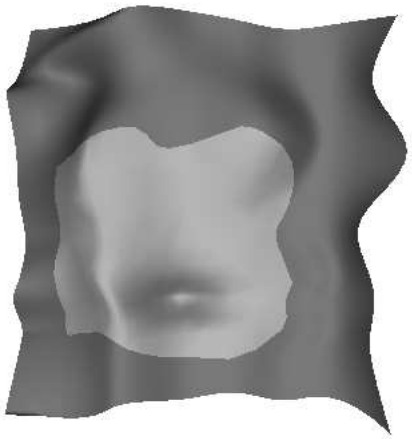


Figure 4.4: Trimmed patch together with its original surface.

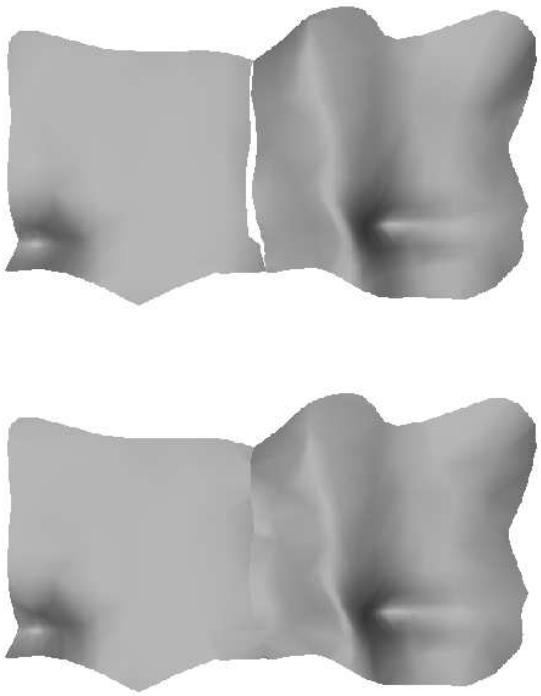


Figure 4.5: Example with $\boldsymbol{b}(t)$ not provided. Top: the input trimmed patches; bottom: the result of joining.

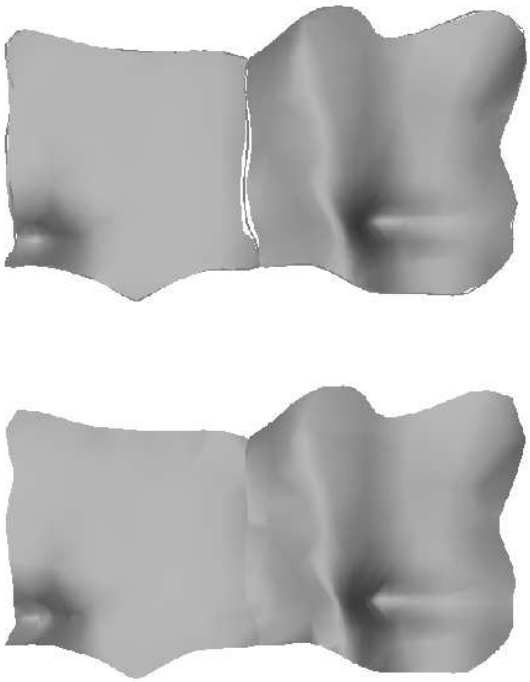


Figure 4.6: Example with $\boldsymbol{b}(t)$ provided. Top: the input trimmed patches; bottom: the result of joining.

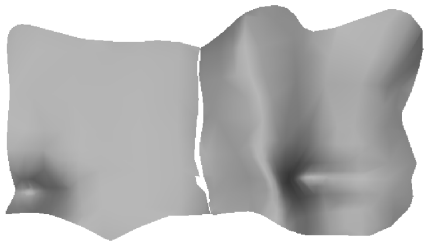


Figure 4.7: Input patches with folding present.

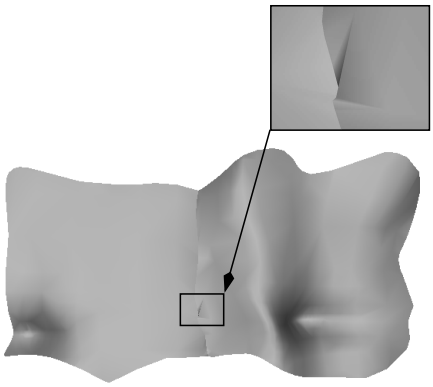


Figure 4.8: Result with flipped triangles.

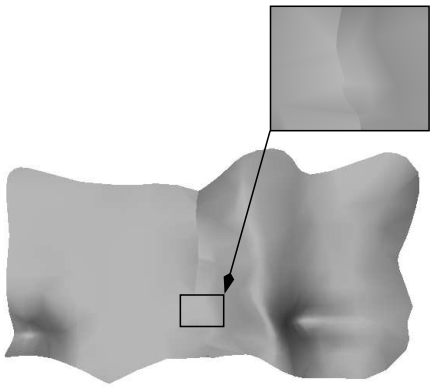


Figure 4.9: Sewing result with Whitney extension.

4.4.2 Computational cost

Let σ be the number of segments in the piecewise linear approximation of the boundary curves $\boldsymbol{b}^k(t)$ (step 2 in Section 4.3.2). The time required to do the joining, including the projection and reprojection, varies directly with $\sigma \cdot n$, where n is defined (Section 4.2) to be the number of vertices in M. The constant of proportionality in our experiments (run on a 2.2 GHz AMD Athlon 64 3500+ processor), was approximately $0.5 \cdot 10^{-4}$. Thus, for a pair of meshes comprising $2.1K$ nodes, with $\sigma = 80$, the total time required was 8.16 seconds. (The examples shown in Figures 4.5 - 4.9 had fewer nodes, and required less time.) Whitney reprojection accounts for 65-85$\%$ of the total time cost.

4.5 Conclusion

Our first conclusion, as suggested in Section 4.1, is that normal-vector criteria will be necessary if we wish to devise reliable algorithms. Note that the purpose of presenting examples like those of Figure 4.2 and Figure 4.8 is not to suggest that such examples will occur frequently when using any particular algorithm but, rather, to illustrate possibilities that must be excluded if we want provably reliable methods. One of the two main contributions of the paper is to set out the minimal requirements for an eventual proof of reliability.

Our second conclusion is that it is possible to devise algorithms, operating at reasonable cost, that will join given mesh patches together while maintaining a proxy for the normal-vector error, as well as the absolute error, at a level below that already present in the given mesh. Furthermore, the mesh in the u-v domain is not disturbed by the reprojection process, and the triangulations of the central mesh and the external region in the u-v domain can be done using the best available method. In this paper the central mesh was triangulated using Maya, while the external region was triangulated using a variant of Ruppert's algorithm, but if better methods become available, they can be used directly. Similarly, the u-v coordinates of any previously-applied mesh-fairing or smoothing algorithm will not be disturbed—only the height field is modified in order to ensure that its slope over the whole patch will not be larger than the slope along the edge of the patch.

The advantage of using normal-vector criteria for graphical simulation is clearly evident from the example of Figure 4.8. Further research should focus on the estimation of normal-vector error by using the mesh itself.

4.6 Acknowledgments

The authors are grateful to F. Duranleau, P. Poulin, I. Stewart and T. Tautges, who provided several useful comments. Responsibility for any errors or omissions lies solely with the authors.

The research of the second author was supported in part by the Natural Sciences and Engineering Research Council of Canada.

References

[1] Kahlesz, F., Balázs, A. and Klein, R. *Multiresolution rendering by sewing trimmed NURBS surfaces*, 281-288, Symposium on Solid Modeling and Applications, June 17-21, 2002, Saarbrücken, Germany.

[2] Borodin, P., Novotni, M. and Klein, R. Progressive gap closing for mesh repairing. Manuscript, Computer Graphics Group, University of Bonn, 2002.

[3] Steinbrenner, J. P., Wyman, J. and Chawner, J. R. Fast surface meshing on imperfect CAD models. Proceedings of the 9th International Meshing Roundtable, 33-41, 2000.

[4] Maya Version 7, Help Manual, Autodesk, 2006.

[5] Haimes, R. and Aftosmis, M. J. Watertight anisotropic surface meshing using quadrilateral patches. Proceedings of the 13th International Meshing Roundtable, 311-322, 2004.

[6] Patrikalakis, N. M. and Maekawa, T. Shape Interrogation for Computer Aided Design and Manufacturing, Springer, 2002.

[7] Campbell, R. J. and Flynn, P. J. A survey of free-form object representation and recognition techniques. *Computer Vision and Image Understanding* 81, 166-210, 2001.

[8] Litke, N., Levin, A. and Schröder, P. Trimming for subdivision surfaces. *Computer Aided Geometric Design* (18), 463-481, 2001.

[9] Mandal, C., Qin, H. and Vermuri, B. C. A novel FEM-based dynamic framework for subdivision surfaces. *Computer-Aided Design* (32), 479-497, 2000.

[10] Kumar, G. V. V. R., Srinivasan, P., Shastry, K. G. and Prakash, B. G. Geometry based triangulation of multiple trimmed NURBS surfaces. *Computer-Aided Design*, (33) 439-454, 2001.

[11] ACIS 3D Toolkit, 1998. Spatial Technology, Boulder, CO.

[12] STEP International Standard, ISO 10303-42, 1997. Industrial automation systems and integration—Product data representation and exchange—Part 41. International Organization for Standardization (ISO), Reference Number ISO 10303-42 1994(E). First Edition 1994-12-15.

[13] Frey, P. J. and Borouchaki, H. Surface mesh evaluation. Proceedings of the 5th *International Meshing Roundtable*, 363-374, 1996.

[14] Cohen-Steiner, D. Alliez, P. and Desbrun, M. Variational shape approximation. *ACM Trans. Graph.* (23), No. 3, 905-914, 2004.

[15] Guskov, I. Manifold-based approach to semi-regular meshing. *Graphical Models*, (69):1, 1-18, 2007.

[16] Grinspun, E. and Schröder, P. Normal bounds for subdivision-surface interference detection. Proceedings of the IEEE Conference on Visualization, 333-340, October 2001.

[17] Volino, P. and Thalmann, N. M. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. In: M. Daehlen and M. Kjelldahl K. (editors), Eurographics '94, vol. 13, No. 3. Blackwell Publishers, pp. C-155–C-164.

[18] Andersson, L.-E., Stewart, N. F. and Zidani, M. Conditions for use of a non-selfintersection conjecture. *Computer Aided Geometric Design* (23), 599-611, 2006.

[19] Whitney, H. Analytic extensions of differentiable functions defined in closed sets. *Trans. Am. Math. Soc.* (36), 63-89, 1934.

[20] Jiang, D. and Stewart, N. F. Maintaining validity of mesh-solids using floating-point arithmetic. SCAN2006: 12th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics. Duisburg, Germany, 26-29 September, 2006. Abstracts Book pp. 119-120.

[21] Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B. and Desbrun, M. Anisotropic polygonal remeshing. *ACM Trans. Graph.* (22), No. 3, 485-493, 2003.

[22] Desbrun, M., Meyer, M., Schröder, P. and Barr, A. H. Implicit fairing of irregular meshes using diffusion and curvature flow. SIGGRAPH Proceedings, 1999.

[23] Beall, M. W. Walsh, J. and Shephard, M. S. Accessing CAD geometry for mesh generation. Proceedings of the 12th International Meshing Roundtable, 2003.

[24] Liepa, P. Filling holes in meshes. Eurographics Symposium on Geometry Processing, Kobbelt, L. Schröder, P. and Hoppe, H. (editors), 2003.

[25] Barequet, G. and Kumar, S. Repairing CAD models. Proceedings of the IEEE Conference on Visualization, 363-370, 1997.

[26] Piegl, L. A. and Tiller, W. Geometry-based triangulation of trimmed NURBS surfaces. *Computer-Aided Design* (30), No. 1, 11-18, 1998.

[27] Bischoff, S., Weyand, T. and Kobbelt, L. Snakes on triangle meshes. Bildverarbeitung für die Medizin, 208-212, 2005.

[28] Aronsson, G. Extension of functions satisfying Lipschitz conditions. *Ark. Mat.* (6), No. 28, 551-561, 1967.

[29] Andersson, L.-E., Stewart, N. F. and Zidani, M. Error analysis for operations in solid modeling in the presence of uncertainty. *SIAM J. Scientific Computing* (29), No. 2, 811-826, 2007.

[30] Kim, M. -S. and Elber, G. Problem reduction to parameter space. Proceedings of the 9th IMA Conference on the Mathematics of Surfaces, 82-98, 2000, London.

[31] Seong, J. -K., Johnson, D. E. and Cohen, E. A higher dimensional formulation for robust and interactive distance queries. Proceedings of Solid and Physical Modeling, 197-205, 2006.

[32] Sud, A., Otaduy, M. A. and Manocha, D. DiFi: fast 3D distance field computation using graphics hardware. *Computer Graphics Forum* (33), No. 3, 2004.

[33] Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Booth, M. and Rossi, F. GNU Scientific Library Reference Manual. Revised second edition. August 2006.

[34] Ruppert, J. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. of Algorithms* (18), 548-585, 1995.

[35] Miller, G. L., Pav, S. E. and Walkington, N. J. When and why Ruppert's algorithm works. Proceedings of the 12th International Meshing Roundtable, 91-102, 2003.

Chapter 5

Robustness of Boolean operations on subdivision-surface models

Boolean operations on standard trimmed-NURBS geometric models are still notoriously difficult problems, and the associated difficulties manifest themselves in the appearance of artifacts such as cracks and gaps. On the other hand, subdivision-surface models as a representation are rapidly gaining popularity in the field of geometric modeling. More and more frequently they are used in place of trimmed-NURBS representations due to their simplicity, and efficiency for smooth surface construction. Also, based on our previous experience with the merging operation on combined mesh-surface models (Ch. 4), the availability of both NURBS information and the mesh data is not easily satisfied. In addition, pure mesh representation (polygon soup) usually does not contain enough topological and geometrical information about the model for the explicit shape control. As an alternative, we can extract subdivision topology from arbitrary meshes using some existing methods [LDW97, EDD+95], to convert arbitrary meshes into subdivision-surface models. But even though the fundamental theory underlying subdivision-surfaces has been widely discussed in the domain of mathematics, there does not exist any theoretical guarantee about the robustness of the implemented applications, *i.e.*, at which precision level we can safely use these models. Based on these observations, we move our focus from trimmed-NURBS representations to subdivision-surface representations, and the target operation is enlarged from a simple merging operation to complete Boolean operations.

An algorithm performing Boolean operations on subdivision-surface models is proposed first. It is based on the use of limit meshes, rather than a refined version of the control meshes.

Limit meshes have intrinsic advantages: they contain fewer triangles than refined control meshes of comparable accuracy, and they are closer to the limit surfaces than the control meshes of the same subdivision level. In this work we restrict our discussion to the Loop subdivision scheme, but the ideas are more generally applicable. We still put our focus on robustness: this includes error bounds and numerical methods for the *a posteriori* validation of topological form of the computed result. In this work, we also use some of our previously published results, for example, the reliable three-dimensional orientation test in Ch. 3 is used in the triangle-triangle intersection procedure.

The preliminary part of this work was presented at the Dagstuhl Seminar in January, 2008, and later appeared in the Dagstuhl seminar proceedings (Dagstuhl Research Online Publication Server). It contained some early-stage bounding results related to the use of the limit mesh, which turned out to be insufficient for our purposes. In the final submitted version of the paper presented here, a different bounding technique is used.

The main contributions of this work are:

- the use of limit mesh for Boolean operations on subdivision-surface models is proposed.
- an error bound is presented for the use of limit mesh.
- a checking method for the well-formedness of the computed result is presented to guarantee the quality of the models produced by our algorithm.

Robustness of Boolean operations on subdivision-surface models

D. Jiang N. F. Stewart

Département d'informatique et de recherche opérationnelle

Université de Montréal

submitted to

Lecture Notes in Computer Science (LNCS)

May 28, 2008

Abstract

This paper describes an algorithm to perform Boolean operations, based on the use of limit meshes, in the case when input objects are defined in terms of triangular meshes and Loop subdivision. The focus of the paper is on robustness, including error bounds and numerical methods for the *a posteriori* validation of topological form.

5.1 Introduction

Boolean operations on standard trimmed-NURBS geometric models [1] are still notoriously difficult problems, and the associated difficulties manifest themselves in the appearance of artifacts such as cracks and gaps [2]. The framework necessary to prove that algorithms work rigorously is available [3], but, so far at least, the required analyses appear to be intractable.

On the other hand, subdivision-surface models are more and more frequently being used in place of trimmed-NURBS representations due to their simplicity, generality, and efficiency for smooth surface construction [4]. In this paper we describe an algorithm for computing Boolean operations on objects defined by their boundaries, represented as subdivision surfaces. The algorithm is similar to the one described in [5], but uses what is called the limit mesh to perform the initial boundary intersection calculation rather than a refined version of the control mesh. The focus of the paper is on robustness: for example, we do not discuss fitting operations [5] in detail. We do, however, consider several robustness issues: integration of Fortune's α-predicate into the code for triangle-triangle intersection [6], new error bounds for the limit surface, and, at least in the regular case, simple and rigorous methods to verify *a posteriori* that the polyhedral computed solution has the same topological form as its corresponding boundary surface. Finding such bounds, and performing such *a posteriori* validations, are essential steps in providing an *a posteriori* backward error analysis [7] for a Boolean-operation algorithm.

Previous work on robustness for Boolean operations on subdivision surfaces includes [8] and [9]. In [8], voxelization representations were used to calculate the Boolean intersection of sets defined by Catmull-Clark subdivision surfaces. In [9], symbolic perturbation methods were used to guarantee topological correctness of the computed result of a Boolean operation.

The algorithm presented here has been implemented, and to some extent we have been concerned with questions of efficiency and triangle count, as described below. In this paper, however, we restrict our attention for the most part to the robustness issues mentioned above.

We suppose that the reader has a general familiarity with subdivision-surface methods for the representation of solids [10].

Boolean operations on solids defined using a subdivision-surface representation are usually carried out on a piecewise polygonal mesh (the *control mesh*), rather than the *limit surface* that defines the true geometry of an input operand [11]. Such an approximation might not be accurate (nor, in the context of collision detection, safe) [12]. The accuracy can be improved, however, by using the *limit mesh*, a polyhedral approximation formed by driving each of the

control points in the control mesh to its limit position [13, 14]. This representation better approximates the limit surface while maintaining the same topological form as the control mesh.

The algorithm discussed in this paper is based on the use of the limit mesh. The discussion refers to the Loop subdivision scheme, but the ideas are more generally applicable. As already mentioned, we do not discuss fitting procedures, but we note here that the *a posteriori* validation is applicable both before and after such fitting procedures have been applied. Also, we often phrase the discussion in terms of regularized Boolean intersection [15] (there is no loss in generality in doing so: different Boolean operations merely change which segments of the original meshes should be retained). The input solids may be denoted S and S', and the operation studied is $S \cap^* S'$, where $\cap^*$ denotes regularized intersection. The input solids are represented by subdivision surfaces defining their boundaries.

The remainder of the paper is organized as follows. In Section 5.2 we discuss the representation of solids using subdivision surfaces. In Section 5.3 we describe the Boolean intersection algorithm. This is followed by the discussion of error bounds and validation of topological form in Section 5.4, and by a short concluding section.

5.2 Representations of solids

A typical solid will be denoted S. It is defined by its boundary surface ∂S, a two-manifold without boundary embedded in $\mathbb{R}^3$, and a directed normal vector specifying which side of ∂S corresponds to the inside of the object. The surface ∂S is defined by a polyhedral mesh (M, P), where M is a (logical) locally-planar triangular mesh, P^T is a $3 \times L$ matrix containing the control points $\mathbf{p}_i \in \mathbb{R}^3, i = 1, \ldots, L$, and the limit surface is defined implicitly by Loop subdivision. We call the polyhedral mesh a *control mesh*, and denote it $\breve{M}$.

Loop subdivision was proposed in [16] and extended in [18, 17, 19]. Triangles are subdivided by splitting each edge, and joining the new vertices created by this split with an edge. The weight for a newly introduced edge point is given by the mask in Figure 5.1 (lower left), and existing vertices are modified using the mask in Figure 5.1 (upper left), with $\beta = \beta(n) = a(n)/n$, and $a(n) = 5/8 - (3 + 2\cos(2\pi/n))^2/64$ [17]. Since $\beta(6) = 1/16$, for regular triangular meshes (*i.e.*, meshes for which the valence n of each vertex is equal to 6) we have $1 - n\beta = 5/8$. Figure 5.1 (right) is discussed below.

The limit surface defined by Loop subdivision is a box spline surface [20], and ∂S can be

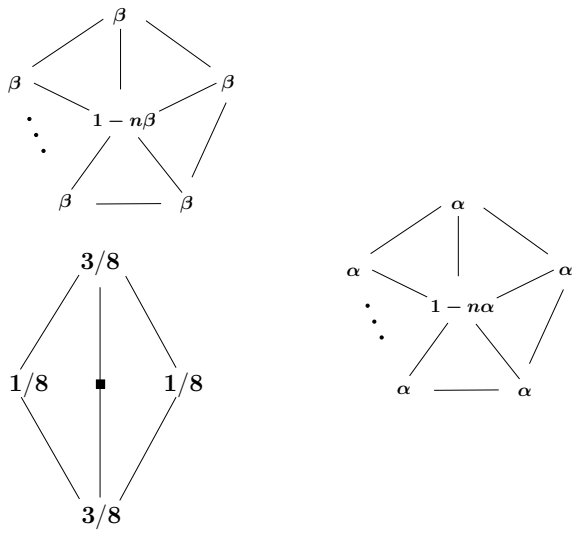


Figure 5.1: Subdivision masks (left) and limit mask (right).

expressed as

$$\partial S = \partial S(u,v) = \sum_i \mathbf{p}_i b_i(u,v) \tag{5.1}$$

where on regular parts of the mesh the basis functions[1] b_i are piecewise polynomials.

The range of the index i in (5.1) was left undefined. In the case of a box spline defined on all of $\mathbb{R}^2$, the range of i could be taken to be the entire grid $\mathbb{Z}^2$. Both in this case and in the case of a finite locally-planar mesh without boundary, however, it is sufficient to consider only vertices in a one-ring neighbour of a triangular patch, as illustrated in Figure 5.2 (right), provided that at least one step of subdivision has been carried out, so that there are no adjacent non-regular vertices.

This can be seen as follows. If we consider the domain of the $b_i(u,v)$ to be all of $\mathbb{R}^2$, the functions $b_i(u,v)$ can be found by substituting a scalar control point with $p_i = 1$ for i corresponding to a particular grid-point labelled i in $h\mathbb{Z}^2 \subset \mathbb{R}^2$, and $p_j = 0$ for $j \neq i$, and then applying the subdivision process until convergence. If we do this by using the masks given in Figure 5.1 (left), it can be shown that the support of $b_i(u,v)$ lies in the convex hull of the set of vertices at distance 2 from i, where distance is measured as an integer quantity in the graph formed by the triangulated grid embedded in $\mathbb{R}^2$ (see Figure 5.2, left). Figure 5.2 (right) is the consequence of looking at this fact from the opposite point of view: the value of the surface on the patch corresponding to a single triangle is determined by the control points that are 1-ring neighbours of the patch. Similarly, if the local parametric domain is supposed to be embedded

[1] In fact, in contrast to the tensor-product B-spline case, these functions do not form a basis for the spline space. A better name would be "nodal functions" [21].

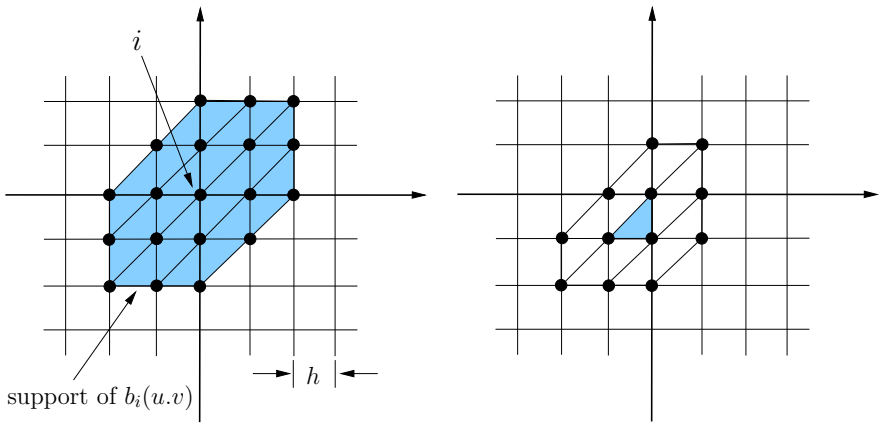


Figure 5.2: Loop subdivision

in $\mathbb{R}^2$ as shown in Figure 5.3 (left) [12], then the corresponding nodal function can be found in

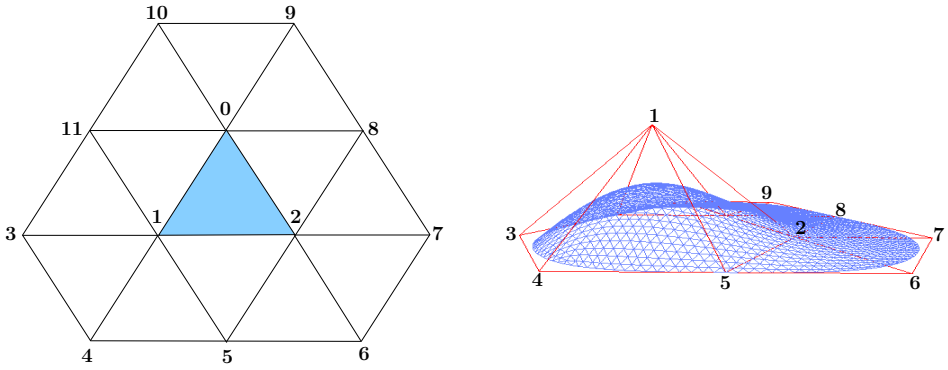


Figure 5.3: Left: a base mesh used to generate the basis functions for the triangle 0-1-2 (regular case: vertex with valence $n = 6$) [12]; right: the resulting basis function at node 1 evaluated at subdivision level four.

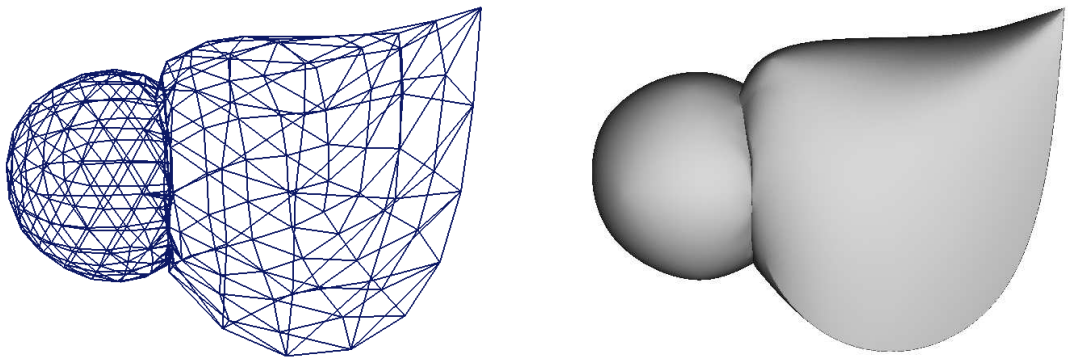
the same way. It is illustrated for the regular case in Figure 5.3 (right).

Finally, to deal with creases introduced due to design considerations, or due to Boolean operations, it is necessary to introduce additional subdivision rules for crease edges and corner vertices [18, 17, 19]. The implementation described below permits crease edges in the input objects, and produces crease edges along intersection curves.

By using the limit mask in Figure 5.1 (right) we can drive any control point to its position on the limit surface. If we take the set of such limit points, and link them together into a polyhedral mesh with the same connectivity as $\breve{M}$, we obtain the *limit mesh*, denoted $\bar{M}$. Both $\breve{M}$ and $\bar{M}$ depend on the level of subdivision ι, but since ι is the same for both meshes, and fixed, we do not show it explicitly.

5.3 The Boolean algorithm

The goal of the Boolean-operation algorithm is to apply the operation to two subdivision-surface models, and to form the result, made up of the desired boundary segments. The algorithm takes the boundaries ∂S and $\partial S'$ of two solids, as described in Section 5.2, and produces a single well-formed object boundary as output. The algorithm introduces modifications of ideas previously suggested by other authors, *e.g.*, the *triangle-triangle-intersection* procedure of [6] is modified by the α-predicate [22] to ensure robustness. The overall idea of the algorithm is similar to [5], but we use the limit meshes $\bar{M}$ and $\bar{M}'$, rather than refined control meshes (which have more triangles), for the intersection-curve calculation. The limit mesh $\bar{M}$ is generally closer to the limit surface than the control mesh $\breve{M}$, with fewer triangles than a refined control mesh of comparable accuracy, which makes the calculation less expensive. An example (in this case, a union operation) produced by the implemented algorithm is given in Figure 5.4.



(a) (b)

Figure 5.4: (a) control mesh (b) union.

Here is the overall description of the algorithm.

1. **Surface intersection.** This step computes the intersection curves of two limit meshes $\bar{M}$ and $\bar{M}'$ and maps them to the control meshes $\breve{M}$ and $\breve{M}'$. The computation uses a triangle-triangle-intersection test, and takes floating-point roundoff error into account.

2. **Cutting.** This step takes the mapped intersection curves as a reference to construct cutting curves, and separates the original control meshes into cut meshes.

3. **Merging.** This step combines the desired parts to form a well-formed object; the intersection curve is tagged as a crease.

4. **Fitting.** This is an optional procedure that aims to reduce the difference between the computed result and true solution [5].

The Boolean intersection algorithm involves two main procedures, *triangle-triangle intersection*, and *refinement*, which is used in the cutting and merging steps. A *snapping* procedure is also used in [5] (if a vertex in the mapped intersection curve is within a certain threshold of a vertex in the control mesh, the latter vertex is moved, and all segments of the intersection curve within a one-ring neighbourhood of the displaced control point are updated). Based on our observation in the context of an algorithm based on the limit mesh, such a procedure has little influence on the number of triangles in the computed result, but a large (negative) effect on the geometric form of the result. Consequently, we did not include it. This reduces both the amount of work and potential robustness problems.

Our first comments on robustness concern the *triangle-triangle-intersection* procedure. This procedure is largely based on the work of Guigue and Devillers [6]. For our implementation, we downloaded their source code (available online); modifications were made in order to introduce the equivalent of Fortune's α-predicate, for robustness reasons. The hypothesis [6] that there are no degenerate triangles in the input will always be satisfied in practice if the input objects have been provided by means of a coarse control mesh. Otherwise this condition must be checked.

Similarly to [33, 22], we define ϵ to be an upper bound $\epsilon > |\delta|$, for all x, y, where $x \hat{*} y = (x * y)(1 + \delta)$ and $\hat{*}$ is a set of operations $\hat{+}, \hat{-}, \hat{\times}, \hat{/}$ defined on the representable reals with relative error ϵ.

The intersection computation relies exclusively on the sign of certain 4×4 determinants, where *sign* is a three-valued function taking values in $\{-1, 0, 1\}$. Consider first the *above-predicate*, which determines whether the point $\boldsymbol{t}$ is above (positive), below (negative), or on (zero) the plane through $\boldsymbol{p}, \boldsymbol{q}$ and $\boldsymbol{r}$:

Definition 1. *Given four three-dimensional points* $\boldsymbol{p} = (p_x, p_y, p_z)$, $\boldsymbol{q} = (q_x, q_y, q_z)$, $\boldsymbol{r} = (r_x, r_y, r_z)$, *and* $\boldsymbol{t} = (t_x, t_y, t_z)$, *we define the above-predicate*

$$ap[\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{t}] := -\begin{vmatrix} p_x & q_x & r_x & t_x \\ p_y & q_y & r_y & t_y \\ p_z & q_z & r_z & t_z \\ 1 & 1 & 1 & 1 \end{vmatrix} = (\boldsymbol{t} - \boldsymbol{p}) \cdot ((\boldsymbol{q} - \boldsymbol{p}) \times (\boldsymbol{r} - \boldsymbol{p})). \tag{5.2}$$

The evaluation of this predicate is error-prone due to the use of finite precision arithmetic [5]. Consequently, a perturbation δ' is introduced similar to the α-*predicate* in [22], and the

classification of point positions is modified as follows:

$$\boldsymbol{t} \leftrightarrow \begin{cases} above\triangle & : & ap[\triangle, \boldsymbol{t}] \in (\delta', \infty) & (sign(ap[\triangle, \boldsymbol{t}]) = 1) \\ on\triangle & : & ap[\triangle, \boldsymbol{t}] \in [-\delta', \delta'] & (sign(ap[\triangle, \boldsymbol{t}]) \Leftarrow 0) \\ below\triangle & : & ap[\triangle, \boldsymbol{t}] \in (-\infty, -\delta') & (sign(ap[\triangle, \boldsymbol{t}]) = -1) \end{cases} \tag{5.3}$$

where $\Leftarrow$ means *considered* to be zero. With these modifications, the plane through $\triangle \boldsymbol{pqr}$ is thickened to contain an ambiguity zone with $\delta' = 160M^3\epsilon$, neglecting higher-order terms of ϵ, and M is a fixed upper bound for the absolute value of any coordinate of any point.

We assume that not all points are coplanar. If all the vertices of one triangle have sign equal to zero with respect to the other triangle, we are in the coplanar case, and we can ignore the potential intersection, since the edges of neighbouring triangles will produce the desired result. To eliminate ambiguities in the opposite case, the first step is to perturb the point having sign equal to 0 by an amount ρ, where $\rho > 2\tau$, in a direction away from the edge opposite the point [6]. The vertices of the two triangles T_1 and T_2 are then permuted to form the layout shown in Figure 5.5, where a simple comparison of intervals determines whether there is a non-empty intersection.

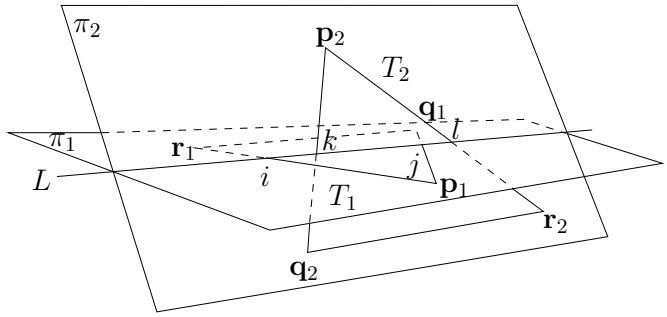


Figure 5.5: Triangle-triangle intersection.

Given two triangles $T_1 : (\boldsymbol{p}_1, \boldsymbol{q}_1, \boldsymbol{r}_1)$ and $T_2 : (\boldsymbol{p}_2, \boldsymbol{q}_2, \boldsymbol{r}_2)$, suppose that at least one of the vertices of T_1 has a non-zero sign for the *above-predicate*, say, $sign(ap[T_2, \boldsymbol{r}_1]) \neq 0$, and that at least one of the vertices of T_1 has different sign from vertex $\boldsymbol{r}_1$, *e.g.*, $sign(ap[T_2, \boldsymbol{p}_1]) \neq sign(ap[T_2, \boldsymbol{r}_1])$. Thus, we are in the case where there is a potential intersection.

Without loss of generality, let $sign(ap[T_2, \boldsymbol{r}_1]) = 1$. Then there are two possibilities for the position of point $\boldsymbol{p}_1$ in the case of intersection:

1. $sign(ap[T_2, \boldsymbol{p}_1]) = -1$; in this case there is definitely an intersection, and we apply the original Guigue-Deveillers algorithm [6].

2. $sign(ap[T_2, \boldsymbol{p}_1]) = 0$; this means that the point $\boldsymbol{p}_1$ falls in the ambiguity zone, and an α-*arithmetic* modification must be applied in order to remove this ambiguity. The ρ perturbation is applied: let the perturbed point be $\boldsymbol{p}_1' = \boldsymbol{p}_1 + \rho\boldsymbol{n}$, where $\boldsymbol{n}$ is the direction of perturbation, determined by the direction through $\boldsymbol{p}$ and orthogonal to the opposite edge of T_1. Here, $\|n\| = 1$.

Our version of the algorithm as described here fails safe, in the sense that if there is actually an intersection, it will be detected, but errors of the opposite type may occur. The maximum error in the case of errors of opposite type can be determined by applying the standard *a priori* bounds [33, p. 107] to the Guigue-Devillers algorithm [6].

The arguments presented here clearly do not constitute a proof of the correctness of the overall process: in particular, such a proof would have to involve consideration of multiple perturbations of a single vertex; the merging step, described below; and take into account the classical steps described in [24] to obtain a regularized result. Note also that, given the fail-safe nature of our algorithm, it might be decided to implement a postprocessing step to eliminate small thin sets (slivers) [9]. This, however, lies outside the domain of numerical analysis.

The goal of *refinement* is first to guarantee that the mesh remains valid (merging step), and secondly, that the cutting curves conform to the shape of the mapped intersection curves (cutting step). A triangle containing a part of the intersection curve is refined if it is detected as "bad", *i.e.* the curve intersects the triangle boundary more than twice, does not intersect at all (the curve is completely inside the triangle), or intersects the boundary twice but on the same side. The refinement is done using quadrisection (midpoint insertion on the triangle edges).

The steps just summarized make up a large part of the implemented Boolean operation algorithm, but since they are not directly concerned with the robustness questions we discuss, we omit the details (the main requirement, from the robustness point of view, is that the process should not modify the topological form of the meshes).

In order to improve the approximation to the true intersection result, an optional fitting step can be applied [5]. This step is applied after execution of the complete Boolean operation. We have used a modified fitting procedure which minimizes the functional formed by the sum, for the two objects, of the terms

$$\sum_j ||f(\tilde{\boldsymbol{p}}_j^\iota) - \mathcal{L}\boldsymbol{p}_j^\iota||^2, \tag{5.4}$$

where j indexes the vertices in the mesh at subdivision level ι, $\boldsymbol{p}_j^\iota$ is one vertex in the mesh at level ι, $\tilde{\boldsymbol{p}}_j^\iota$ is its corresponding position in the original coarse control mesh $\breve{M}$, $f(\cdot)$ is the

limit-surface evaluation function, and $\mathcal{L}$ is the limit matrix that determines the limit position of the vertex $\boldsymbol{p}_j^\iota$. Other constraints can be added to obtain better fitting.

5.4 Error estimation and verification of well-formedness

5.4.1 Error estimation

Using the limit mesh as an approximation to the limit surface for the intersection calculation implies potential errors in the final result. In this section, we will give a bound on the possible error, based on the work of [14], followed by some possible improvements.

Bounds of this type were discussed in a preliminary way in [25]. Other work on this topic includes [13, 12], as well as earlier work [26] on B-splines that used derivatives to bound the surface.

Each face $\bar{\boldsymbol{F}}$ in the limit mesh $\bar{M}$ is defined by the corners $\boldsymbol{q}_0$, $\boldsymbol{q}_1$, and $\boldsymbol{q}_2$, which can be obtained by limit-surface evaluation

$$\boldsymbol{q}_j = \partial S(u_j, v_j) = \sum_{i=0}^{n+5} \boldsymbol{p}_i \cdot b_i(u_j, v_j), \;\; j = 0, 1, 2, \tag{5.5}$$

where the $\boldsymbol{p}_i$ are the control points in the control mesh $\breve{M}$ that affect the position of $\boldsymbol{q}_j$, the b_i are the nodal functions, and (u_j, v_j) is the coordinate for $\boldsymbol{q}_j$ in the parametric domain illustrated in Figure 5.3 (left).

Let $\boldsymbol{n}$ denote the face normal of $\bar{\boldsymbol{F}}$. An upper and lower bound at each of these three vertices can be obtained:

$$\ell_j \leq \boldsymbol{n}^T \boldsymbol{q}_j \leq \mu_j \tag{5.6}$$

where

$$\begin{aligned} \ell_j &= \sum_{i=0}^{n+5} (\boldsymbol{n}^T(\boldsymbol{p}_i - \boldsymbol{q}_j))^+ \;\; b_i^- + \sum_{i=0}^{n+5} (\boldsymbol{n}^T(\boldsymbol{p}_i - \boldsymbol{q}_j))^- \;\; b_i^+ \\ \mu_j &= \sum_{i=0}^{n+5} (\boldsymbol{n}^T(\boldsymbol{p}_i - \boldsymbol{q}_j))^+ \;\; b_i^+ + \sum_{i=0}^{n+5} (\boldsymbol{n}^T(\boldsymbol{p}_i - \boldsymbol{q}_j))^- \;\; b_i^- \end{aligned} \tag{5.7}$$

as illustrated in Figure 5.6 for a two-dimensional case, and

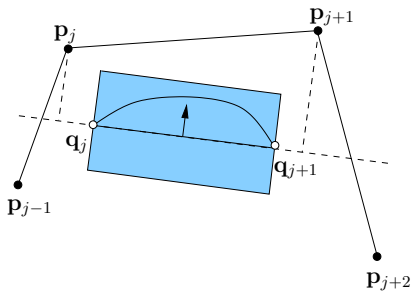


Figure 5.6: A 2D illustration for the upper and lower bound construction.

$$(\boldsymbol{n}^T(\boldsymbol{p}_i - \boldsymbol{q}_j))^+ = \max\{\boldsymbol{n}^T(\boldsymbol{p}_i - \boldsymbol{q}_j), 0\}$$
$$(\boldsymbol{n}^T(\boldsymbol{p}_i - \boldsymbol{q}_j))^- = \min\{\boldsymbol{n}^T(\boldsymbol{p}_i - \boldsymbol{q}_j), 0\}$$

(see [14]). It is necessary here to estimate the range $[b_i^-, b_i^+]$ of the basis function b_i, where

$$b_i^- = \min_{u,v} b_i(u,v), \qquad b_i^+ = \max_{u,v} b_i(u,v),$$

and the minimum and maximum are taken over the triangle 0-1-2 in Figure 5.3 (left). As suggested in [14], this can be done by estimating the basis function by applying the subdivision process to the Dirac polygon described above ($p_i = 1$, $p_j = 0$ if $j \neq i$). Since this only gives an estimate, however, it is necessary to iterate the process [14], beginning with the coarse estimate of the range $[-1, 1]$. In this way we get a bounding volume $\mathcal{V}$ defined by the offsets of limit-mesh vertices (see Figure 5.7):

$$\boldsymbol{q}_j + \frac{\ell_j}{\boldsymbol{n}^T\tilde{\boldsymbol{n}}_j}\tilde{\boldsymbol{n}}_j, \qquad \boldsymbol{q}_j + \frac{\mu_j}{\boldsymbol{n}^T\tilde{\boldsymbol{n}}_j}\tilde{\boldsymbol{n}}_j \tag{5.8}$$

where $\tilde{\boldsymbol{n}}_j$ is the normal vector at each vertex $\boldsymbol{q}_j$.

Possible improvements on the bounding volume can be obtained by using the fact that the limit mesh is a down-sampling of the limit surface, which means that all of its vertices lie on the limit surface (except for floating-point error). We will modify the bound above for a tighter enclosure of the limit mesh by exploring this idea.

Using the tangent mask, a tangent plane $\mathcal{P}_j, j = 0, 1, 2$, at the three vertices of each limit

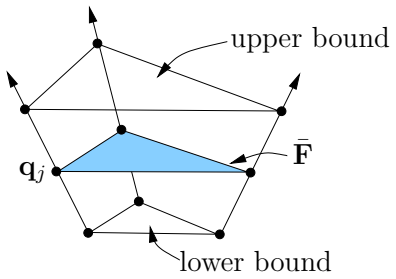


Figure 5.7: Upper and lower bounds for a single face in the limit mesh.

face can be obtained as:

$$\mathcal{P}_j = (\boldsymbol{q}_j, \tilde{\boldsymbol{n}}_j) \tag{5.9}$$

where $\boldsymbol{q}_j$ is vertex of the limit face that lies in the plane, and $\tilde{\boldsymbol{n}}_j$ is its vertex normal, given as

$$\tilde{\boldsymbol{n}}_{\boldsymbol{j}} = \boldsymbol{u}_{\boldsymbol{1}} \times \boldsymbol{u}_{\boldsymbol{2}} \tag{5.10}$$

$$\boldsymbol{u}_1 = c_1\boldsymbol{p}_1 + c_2\boldsymbol{p}_2 + \ldots + c_n\boldsymbol{p}_n$$
$$\boldsymbol{u}_2 = c_2\boldsymbol{p}_1 + c_3\boldsymbol{p}_2 + \ldots + c_1\boldsymbol{p}_n,$$

where $\boldsymbol{p}_1$, $\boldsymbol{p}_2$,...,$\boldsymbol{p}_n$ are the neighbours of vertex $\boldsymbol{q}_j$, and $c_i = \cos(2\pi i/n)$ are the limit-mask coefficients. Let

$$\theta_j = \frac{\boldsymbol{n}^T\,\tilde{\boldsymbol{n}}_j}{||\boldsymbol{n}||||\tilde{\boldsymbol{n}}_j||}, j = 0, 1, 2, \tag{5.11}$$

and

$$\theta = \min\{\theta_j, j = 0, 1, 2\}. \tag{5.12}$$

We can adjust each vertex normal $\tilde{\boldsymbol{n}}_j$ outward from the center of the limit face, by rotating the vector $\boldsymbol{c} - \boldsymbol{q}_j$ around the axis formed by $\tilde{\boldsymbol{n}}_j \times (\boldsymbol{c} - \boldsymbol{q}_j)$ where $\boldsymbol{c}$ is the center of the limit face, until the new vertex normal $\tilde{\boldsymbol{n}}'_j$ satisfies

$$\frac{\boldsymbol{n}^T\,\tilde{\boldsymbol{n}}'_j}{||\boldsymbol{n}||||\tilde{\boldsymbol{n}}'_j||} = \theta, \;\; j = 0, 1, 2. \tag{5.13}$$

Then for each vertex $\boldsymbol{q}_j$ we get a new plane

$$\mathcal{P}_j = (\boldsymbol{q}_j, \tilde{\boldsymbol{n}}'_j). \tag{5.14}$$

By reflecting each of these three planes with respect to the limit face $\bar{\boldsymbol{F}}$, we get three other planes $\mathcal{P}'$. Intersecting each of these planes with the bounding volume $\mathcal{V}$ previously calculated, $\mathcal{P}_j$ with the upper bound, and $\mathcal{P}'$ with the lower bounds (see Figure 5.8), we can get a tighter closure for each face in the limit mesh.

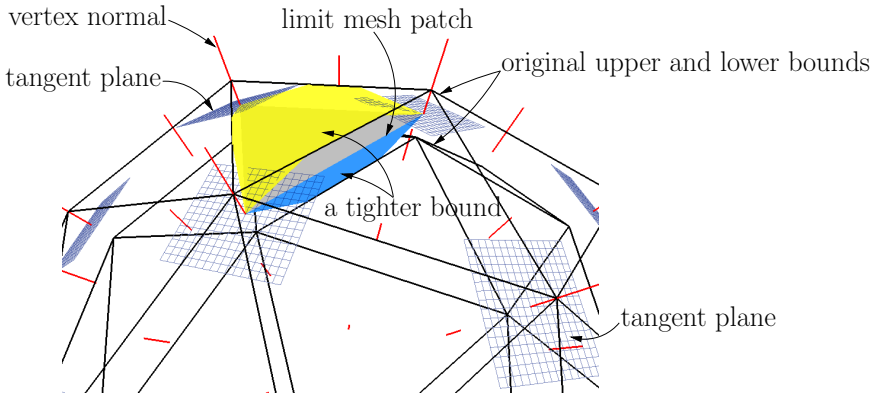


Figure 5.8: Illustration for the tighter bound construction.

For now, these modifications provide only approximate bounds, and more work is required to transform them into provable bounds that are guaranteed to enclose the limit surface.

5.4.2 A posteriori verification of well-formedness

It is of interest to be able to confirm that the limit mesh $\bar{M}$ (respectively $\bar{M}'$) has the same topological form as the corresponding input set, represented by its boundary ∂S (respectively $\partial S'$). Similarly, suppose that M^c is the mesh corresponding to the computed approximation of the result of the Boolean operation, *i.e.*, M^c is intended to approximate the boundary of $S_I = S \cap^* S'$. (The mesh M^c is obtained from refined control meshes corresponding to each input operand.) Again, it may be of interest to confirm that M^c has the same topological form as ∂S_I^c, the actual surface associated with the computed mesh. We will phrase the discussion of these questions in terms of the first of the examples just given.

Given the limit mesh $\bar{M}$, the fact that two of its faces are disjoint does not imply that the corresponding faces of ∂S are disjoint. Similarly, it may happen that $\bar{F}_1$ and $\bar{F}_2$ are adjacent faces sharing an edge or vertex, but that the corresponding faces F_1 and F_2 of ∂S have extraneous intersections, *i.e.*, intersections other than those along the designated edge or at the designated vertex. A completely robust algorithm should be able to perform *a posteriori* validations of computed results that exclude the possibility of inconsistencies of this kind. (Note that there

is no practical inconvenience in assuming that faces in a well-formed mesh do not share more than a single edge or vertex.)

Detection of intersection between patches F_1 and F_2 that are supposed to be disjoint can be detected on a fail-safe basis by comparison of convex hulls (*i.e.*, non-intersection of convex hulls is a sufficient condition for non-intersection of patches). Excluding the possibility of self-intersection of a patch F_1, and of extraneous intersections of adjacent patches F_1 and F_2, was discussed in [28], where the method of [27] was used. We extend that work as follows. First of all, we conclude that in the regular case, it is not necessary to compute the projection direction required in [27]. This means, in particular, that in the regular case there is no need to omit verification of the second condition in [27], which was suggested as a possible approach in [28]. Secondly, [28] detects extraneous intersections by applying the criterion of [27] to the union of adjacent patches. It was shown in [29], however, that there is a supplementary condition to be satisfied if this method is used, and we show how to verify this supplementary condition in the regular case.

The details for the following extensions can be found in [30]. The first extension follows from the fact that if the corners of $\bar{F}_1$ and $\bar{F}_2$ all have valence 6 (the regular case), then the corresponding patches F_1 and F_2 can be expressed as Bézier surfaces, and the Bézier coefficients are explicitly available [31, 32]. This means that extraneous intersections can be detected by the convex-hull criterion [30, Crit. 3.2.1*] (common edge) and [30, Crit. 3.2.2*] (common vertex). Furthermore, it is easy to extend this approach to work in a fail-safe manner, once the separation plane specified in these criteria has been found, by applying the standard *a priori* bounds for floating-point arithmetic to the calculation of the inner products defining the separation planes. Similar remarks apply to the case of self-intersection of a patch, say F_1, using [30, Crit. 3.1*].

The second extension, mentioned above, concerns the fact that application of the criterion of [27] to the union $F_1 \cup F_2$ of adjacent patches requires verification of a supplementary condition along the common boundary, namely that the mapping defining the combined patch must be locally one-to-one along the common boundary [29, Prop. 2.2]. This is true in both the regular and non-regular case. In the regular case the condition can be verified, using the fact that the common boundary is a Bézier curve, and using [30, Crit. 2.1*]. Again, this result can be made fail-safe when ordinary floating-point arithmetic is used.

5.5 Conclusion

We have given a summary description of an implemented algorithm that computes Boolean operations on objects represented by their subdivision-surface boundaries. The algorithm is based on the use of the limit mesh, rather than a refined control mesh, for the computation of the intersection between the surfaces defining the two operands. Most of the discussion in the paper was concerned with three robustness issues of interest in the context of this algorithm, namely the robustness of triangle-triangle intersection, approximation of the limit surface by the limit mesh, and *a posteriori* verification of well-formedness. While the nature of the mathematical arguments necessary to resolve these issues was described, the paper did not give proofs. Thus, future work should include integration of the analysis outlined above into a combined whole, to produce a unified robustness result for Boolean intersection, including validation results in the non-regular case. Such a result would include, in particular, procedures permitting the *a posteriori* validation of topological form.

References

[1] STEP International Standard. Industrial automation systems and integration—Product data representation and exchange—Part 42. ISO 10303-42, 1997.

[2] Farouki, R. Closing the gap between CAD model and downstream application. *SIAM News* (32), No. 5, 1999.

[3] Andersson, L.-E., Stewart, N. F. and Zidani, M. Error analysis for operations in solid modeling in the presence of uncertainty. *Sc. J. Sci. Comput.* 29(2), 811-826, 2007.

[4] Bischoff, S. and Kobbelt, L. Teaching meshes, subdivision and multiresolution techniques. *Computer-Aided Design*, 36(14) 1483-1500, 2004.

[5] Biermann, H., Kristjansson D. and Zorin, D. Approximate Boolean operation on free-form solids. *Proc. ACM SIGGRAPH*, 2001.

[6] Guigue, P. and Devillers, O. Fast and robust triangle-triangle overlap test using orientation predicates. *J. Graphics Tools* 8(1), 25-32, 2003.

[7] Jiang D. and Stewart, N. F. Floating-point arithmetic for computational-geometry problems with uncertain data. *IJCGA*, 2008, to appear.

[8] Lai, S. and Cheng, F. Robust and error controllable Boolean operations on free-form solids represented by Catmull-Clark subdivision surfaces. *Computer Aided Design and Applications*, 4(1-4), 487-496, 2007.

[9] Smith, J. M. and Dodgson, N. A. A topologically robust algorithm for Boolean operations on polyhedral shapes using approximate arithmetic. *Computer-Aided Design* (39) 149-163, 2007.

[10] DeRose, T., Kobbelt, L., Levin, A. and Sweldens, W., Subdivision for modeling and animation. *SIGGRAPH course notes* 2000.

[11] Linensen, L. Netbased Modelling. *Proc. SCCG*, 259-266, 2000.

[12] Wu, X. and Peters, J. Interference detection for subdivision surfaces. *Eurographics*, 2004.

[13] Huang Z. and Wang, G. Distance between a Catmull-Clark subdivision surface and its limit surface. *Proc. ACM Symp. Solid and Physical Modeling*, 233-240, 2007.

[14] Kobbelt, L. Tight bounding volumes for subdivision surfaces. *Pacific Graphics*, 17-26, ed. B. Werner, 1998.

[15] Requicha, A. A. G. Representations for rigid solids: theory, methods and systems. *Computing Surveys*, 12(4), 437-464, 1980.

[16] Loop, C. T. Smooth subdivision surfaces based on triangles. MSc thesis, Department of Mathematics, University of Utah, August, 1987.

[17] Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J. and Stuetzle, W., Piecewise smooth surface reconstruction, *J. Computer Graphics*, 295-302, 1994.

[18] Biermann, H., Levin, A. and Zorin, D. Piecewise smooth subdivision surfaces with normal control. *Proc. ACM SIGGRAPH*, 113-120, 2000.

[19] Ma, W., Ma, X., Tso, S.-K. and Pan, Z. A direct approach for subdivision surface fitting from a dense triangle mesh. *Computer-Aided Design*, 36(6) 525-536, 2004.

[20] de Boor, C., Köllig, K. and Riemenschneider, S., Box Splines. Springer-Verlag, 1993.

[21] Peters. J. and Reif, U. Subdivision Surfaces. Springer, 2008.

[22] Fortune, S. Stable maintenance of point set triangulations in two dimensions. *Proc. 30th annual IEEE Symp. Foundations of Computer Science*, (30) 494-499, 1989.

[23] Wilkinson, J. H. The Algebraic Eigenvalue Problem. Oxford, 1965.

[24] Tilove, R. B. Set membership classification: a unified approach to geometric intersection problems. *IEEE trans. Computers*, 29(10), 874-883, 1980.

[25] Jiang, D. and Stewart, N. F. Robustness of Boolean operations on subdivision-surface models. *Dagstuhl seminar proceedings*, Dagstuhl Research Online Publication Server (DROPS) http:://drops.dagstuhl.de/opus/volltexte/2008/1443

[26] Filip, D. Magedson, R. and Markot, R. Surface algorithms using bounds on derivatives. *Computer Aided Geometric Design*, (3) 295-311, 1986.

[27] Volino, P. and Thalmann, N. M. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Eurographics* (13), C155-C164, 1994.

[28] Grinspun, E. and Schröder, P. Normal bounds for subdivision-surface interference detection. *IEEE Visualization*, 2001.

[29] Andersson, L.-E., Stewart, N. F. and Zidani, M. Conditions for use of a non-selfintersection conjecture. *CAGD* (23), 599-611, 2006.

[30] Andersson, L.-E., Peters, T. J. and Stewart, N. F. Self-intersection of composite curves and surfaces. *CAGD* 15(5), 507-527, 1998.

[31] Boehm, W. Triangular spline algorithms. *CAGD* (1), 61-67, 1985.

[32] Kim, M. and Peters, J. Fast and stable evaluation of box-splines via the Bézier form. Technical Report, University of Florida, REP-2007-422, 2007.

[33] Dahlquist, G. and Björk, A. Numerical Methods in Scientific Computing, Volume I. Society for Industrial and Applied Mathematics, Philadelphia, 2008.

Chapter 6

Conclusion

"*During the 1991 Gulf War, the United States used a missile defense system called Patriot to defend its troops. The system was largely effective but on one occasion, it failed badly. An analysis after the event explained what happened. The internal clock of the computer that controlled the defense system stored the time as an integer value in units of tenths of a second, and the computer program converted this to a floating point value in units of seconds, rounding the expansion accordingly. Because the program was an old one that had been updated to account for new technology, the conversion to floating point was done more accurately in some places in the program than in others. To calculate a time interval, the program took two snapshots of the clock and subtracted them. Because of the round inconsistencies, the system failed to work when it had been running for more than 100 hours.*" [Ove01]

The above example may help explain the importance of reliability, as it is said "*There is one thing that is even more important than lightning speed, and that is reliability*" [Ove01]. This is especially true because many critical matters today are dependent on complex computer programs, and much of this code depends, in one way or another, on floating-point computing. They can be greatly affected by its reliability.

In this thesis, we presented our work on the problem of reliable computation for geometric models. It covered three individual but related problems: floating-point arithmetic for computational-geometry problems, especially with the application of backward error analysis in different geometric problems; the combined mesh-surface-model repair problem, with focus on the joining procedure; and the robustness of Boolean operations on subdivision-surface models.

6.1 Summary

Floating-point arithmetic is very convenient for most practical work because of its numerous engineering advantages, but naively applied floating-point arithmetic can cause disasterous results. The now-standard backward error analysis provides us a tool to distinguish those algorithms that overcome the problem to *whatever extent it is possible to do so*. Three examples were presented to illustrate how to carry out error analysis in different geometric application contexts. We showed that floating-point arithmetic *may* be sufficient, provided that a stable algorithm is applied, in the case where uncertainties are present in the data.

Trimmed-NURBS surfaces have been widely adopted in most geometric modelers, and geometric operations on this representation are very important. We proposed an algorithm for the joining operation for combined mesh-surface patches, with guidance based on the use of two error measures. The joined result is guaranteed to satisfy both the absolute error criterion and the normal error criterion. The necessity of these two error criteria has also been proved, if we wish to devise a reliable algorithm. Two different cases are considered for the proposed algorithm, based on the availability or not of an explicit joining curve.

Trimmed-NURBS get their advantage from being able to model complex geometrical objects, but the trimming difficulties and the error-prone conversion procedure hinder their application. Subdivision-surface models, as an alternative to trimmed-NURBS, have rapidly gained popularity as a geometric representation due to their simplicity and efficiency for smooth surface construction. But even though the fundamental theory of these models has been well discussed and understood, few theoretical guarantees about the robustness of the implemented applications are available.

Amongst these applications are the Boolean operations. Boolean operations are one of the most important facilities of geometric modelers. Their application on trimmed-NURBS models are known to be difficult, and care has to be taken to handle special and degenerate cases. We have studied the problem of applying Boolean operations to subdivision-surface models. An implemented algorithm that computes Boolean operations on objects represented by their subdivision-surface boundaries was presented. The proposed algorithm is based on the use of the limit mesh, rather than a refined control mesh, for the computation of the intersection between the surfaces defining the two operands. Our focus has remained on the robustness issues of interest in the context of this algorithm.

6.2 Future work

A first possible extension to the current work is a theoretic justification for the use of the limit mesh of subdivision-surface models, as an operand for Boolean operations, in place of a finer control mesh. Some empirical results have been presented in [HW07] for Catmull-Clark subdivision-surface models, but no theoretical result is available on this subject.

A framework for a backward error analysis, suitable for the case of Boolean operations on objects represented by internally inconsistent trimmed-NURBS representations, was given in [ASZ07], but no such framework has been given for subdivision-surface models. Therefore, an immediate extension of our work would be to generalize the current validation results to the non-regular case, and to integrate all of this analysis into a combined whole, to produce a unified robustness result for Boolean intersection for subdivision-surface models. This result would include, in particular, procedures permitting the *a posteriori* validation of topological form.

The impact of nonrobustness in the domain of geometric modeling is well known, especially its effects on economics and productivity, e.g. it is the principal barrier to the full automation of the modeling system [Yap01]. Over the past twenty years much progress has been made on the precision and robustness problem. Methods to enhance the precision of intersection computation, to monitor numerical error contamination and to find alternate means of performing arithmetic, have been explored in some detail [Muk05]. Further, more attention has been paid to improving robustness, e.g. the birth of Computational Geometry Algorithms Library (CGAL) project [g-c], which is a joint effort by a number of research groups in Europe and Israel to produce a robust software library of geometric algorithms and data structures [Hal02]. The goal of CGAL is to make available a carefully designed and implemented library with an emphasis on robustness and generality.

From a long-term view, unfortunately, no satisfactory general-purpose solution has been found for the robustness problem, especially in geometric modeling [Sch99]. Robustness issues are still critical in the passage from theory to practice in geometric algorithms. Ignoring these issues can result in unreliable or incorrect programs. Transforming a geometric algorithm into an effective computer program is particularly difficult because of the basic assumptions made on most theoretical geometric algorithms, concerning complexity measures and the handling of robustness, namely issues related to arithmetic precision and degenerate input [Hal02]. For the CAD community, one of the biggest challenges today is still robustness related issues [KBF05].

Translation of geometries from one CAD system into another is far from stable: holes, translation errors, and other problems often arise. The major sources are: floating-point arithmetic and tolerances. Floating-point arithmetic can be dealt with theoretically but not yet practically. Digital arithmetic and current mathematical theory are insufficient to perform reliably for complex geometric operations and to interoperate well with CAD downstream analysis software [Far99, KBF05, BAA+99]. Other problems include mesh-based techniques: major problems are reliability and difficulty in preserving small features whose size is of the same order of error due to some user-specified global distance threshold [PM08].

As "*the availability of greatly improved computational techniques and immensely faster computers allows the routine solution of complicated problems that would have seemed impossible just a generation ago*" [Ove01], we hope, one day, nonrobustness will be resolved as well.

Appendix

Permission has been obtained from the publishers for the following two articles (one to appear, and one published) in this thesis:

1. Permission from *International Journal of Computational Geometry and Applications* (IJCGA) for the paper “Floating-point arithmetic for computational-geometry problems with uncertain data”.

2. Permission from *European Council for Modelling and Simulation* (ECMS) for the paper “Reliable joining of surfaces for combined mesh-surface models”.

Notes on the implementation

The implementation of all of this Ph.D. work was carried out in C++. OpenGL and qt were used for visualization and interface design.

For the work on reliable joining of surfaces for combined mesh-surface models (Ch. 4):

- Software Maya [g-ma] was used to generate 3D trimmed-NURBS models and the corresponding triangular meshes.
- GNU Scientific Library (GSL) was used for the construction of correspondences between trimmed-NURBS and triangular meshes.

For the work on robustness of Boolean operations on subdivision-surface models (Ch. 5):

- The halfedge data structure was used for subdivision-surface models (OpenMesh [g-o]), both for the data storage and the mesh manipulation.
- The Axis-Aligned Bounding Box (AABB) hierarchy was used in the subdivision surfaces intersection calculation procedure for optimization purposes.
- The GNU Scientific Library (GSL) was used for the minimization problem in the fitting procedure.

Software such as *matlab* [g-mb] and *mathematica* [g-mc] were used for prototype and verification purposes. Xfig [g-x] was used to draw the illustration figures in the thesis.

Bibliography

[ABA02] Andujar, C., Brunet, P., and Ayala, D. Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graphics*, 21(2):88–105, 2002.

[AH83] Alefeld, G. and Herzberger, J. *Introduction to Interval Computation*. Academic press, New York, 1983.

[Ale02] Alexa, M. Wiener filtering of meshes. In *Shape Modeling International*, pages 51–60, 2002.

[AS09] Andersson, L.-E. and Stewart, N. F. *An Introduction to the Mathematics of Subdivision Surfaces*. SIAM, 2009. To appear.

[ASZ07] Andersson, L.-E., Stewart, N. F., and Zidani, M. Error analysis for operations in solid modeling in the presence of uncertainty. *SIAM Journal of Scientific Computing*, 29(2):811–826, 2007.

[BAA+99] Bern, M., Agarwal, P. K., Amenta, N., Chew, P., Dey, T., Dobkin, D. P., Edelsbrunner, H., Grimm, C., Guibas, J., Harer, J., Hass, J., Hicks, A., Johnson, C. K., Letscher, D., Plassmann, P., Sedgwick, E., Snoeyink, J., Yap, C., and Zorin, D. Emerging challenges in computational topology, 1999. NSF-funded workshop on Computational Topology.

[Bar07] Barthe, L. *Informatique Graphique, Modélisation Géométrique et Animation*, chapter: Courbes et surfaces de subdivision, pages 135–162. Hermes, 2007.

[BDK98] Barequet, G., Duncan, C., and Kumar, S. Rsvp: A geometric toolkit for controlled repair of solid models. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):162–177, 1998.

[BGK04] Balazs, A., Guthe, M., and Klein, R. Efficient trimmed NURBS tessellation. *Journal of WSCG*, 12(1), 2004.

[BK97] Barequet, G. and Kumar, S. Repairing cad models. In *Proceedings of IEEE Visualization*, pages 363–370, 1997.

[BK04] Bischoff, S. and Kobbelt, L. Teaching meshes, subdivision and multiresolution techniques. *Computer-Aided Design*, 36(14):1483–1500, 2004.

[BK05] Bischoff, S. and Kobbelt, L. Structure preserving CAD model repair. *Computer Graphics Forum*, 24(3):527–536, 2005.

[BKZ01] Biermann, H., Kristjansson, D., and Zorin., D. Approximate Boolean operations on free-form solids. In *Proceedings of ACM SIGGRAPH*, pages 185–194, 2001.

[BLZ00] Biermann, H., Levin, A., and Zorin, D. Piecewise smooth subdivision surfaces with normal control. In *Proceedings of ACM SIGGRAPH*, 2000.

[BMS94] Burnikel, C., Mehlhorn, K., and Schirra, S. On degeneracy in geometric computations. In *5th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 16–23, 1994.

[BP00] Bern, M. and Plassmann, P. Mesh generation. In Sack, J. and Urrutia, J., editors, *Handbook of Computational Geometry*. Elsevier science, 2000.

[BPK+07] Botsch, M., Pauly, M., Kobbelt, L., Alliez, P., Lévy, B., Bischoff, S., and Rössl, C. Geometric modeling based on polygonal meshes. ACM SIGGRAPH course notes, 2007.

[BS95] Barequet, G. and Sharir, M. Filling gaps in the boundary of a polyhedron. *Computer-Aided Geometric Design*, 12(2):207–229, 1995.

[BS02] Bolz, J. and Schröder, P. Rapid evaluation of Catmull-Clark subdivision surfaces. In *Proceedings of the 7th International Conference on 3D Web Technology*, pages 11–17. ACM press, 2002.

[BW92] Bohn, J. H. and Wozny, M. J. Automatic CAD model repair: Shell-closure. In *Proceedings of Symposium on Solid Freeform Fabrication*, pages 86–94, 1992.

[CC78] Catmull, E. and Clark, J. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 1978.

[Che89a] Chew, L. P. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[Che89b] Chew, L. P. Guaranteed-quality triangular meshes. Technical report, Cornell university, April 1989.

[CM00] Casciola, C. and Morigi, S. *Advances in the theory of computational mathematics, recent trends in numerical analysis*, chapter: The trimmed NURBS age. Nova Science Pub., 2000.

[CPD+96] Certain, A., Popovic, J., DeRose, T., Duchamp, T., Salesin, D., and Suetzle, W. Interactive multiresolution surface viewing. In *Proceedings of ACM SIGGRAPH*, pages 91–98, 1996.

[DB08] Dahlquist, G. and Björck, A. *Numerical methods in scientific computing*, volume 1. SIAM, 2008.

[DKT98] DeRose, T., Kass, M., and Truong, T. Subdivision surfaces in character animation. In *Proceedings of ACM SIGGRAPH*, pages 85–94, 1998.

[DMGL02] Davis, J., Marschner, S., Garr, M., and Levoy, M. Filling holes in complex surfaces using volumetric diffusion. In *Proceedings of International Symposium on 3D Data Processing, Visualization, Transmission*, pages 428–438, 2002.

[DS78] Doo, D. and Sabin, M. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(2):356–360, 1978.

[DS88] Dobkin, D. P. and Silver, D. Recipes for geometry and numerical analysis - part I: an empirical study. In *Proceedings the 4th Annual Symposium on Computational Geometry (SCG)*, pages 93–105, New York, NY, USA, 1988. ACM Press.

[DS89] D'Azevedo, E. F. and Simpson, R. B. On optimal interpolation triangle incidences. *SIAM Journal on Scientific and Statistical Computing*, 10(6):1063–1075, 1989.

[EDD+95] Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., and Stuetzle, W. Multiresolution analysis of arbitrary meshes. In *Proceedings of ACM SIGGRAPH*, pages 173–182, 1995.

[EL00] Ely, J. S. and Leclerc, A. P. Correct Delaunay triangulation. *Journal of Reliable Computing*, 6:23–38, 2000.

[Far99] Farouki, R. Closing the gap between CAD model and downstream application. *SIAM News*, 32(5), 1999.

[FGG03] Fernández, J.-J., García, I., and Garzón, E. M. Floating point arithmetic teaching for computational science. *Future Generation Computer Systems*, 19:1321–1334, 2003.

[For93] Fortune, S. *Directions in computational geometry*, chapter Computational geometry. Information geometers, 1993.

[For95] Fortune, S. Numerical stability of algorithms for 2D Delaunay triangulations. *International Journal of Computational Geometry and Applications*, 5(1):193–213, 1995.

[FPRJ00] Frisken, S. F., Perry, R. N., Rockwood, A. P., and Jones, T. R. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH*, pages 249–254, 2000.

[g-b] http://en.wikipedia.org/wiki/Constructive_solid_geometry.

[g-c] CGAL website, http://www.cgal.org/.

[g-L] http://gala.univ-perp.fr/ langlois/dea/seances04/index.html.

[g-ma] http://usa.autodesk.com.

[g-mb] http://www.mathworks.com/.

[g-mc] http://www.wolfram.com/.

[g-o] http://www.openmesh.org/.

[g-x] http://www.xfig.org/.

[Gib98] Gibson, S. F. F. Using distance maps for accurate surface representation in sample volumes. In *Proceedings of IEEE Symposium on Volume Visualizations*, pages 23–30, 1998.

[GKSS02] Guskov, I., Khodakovsky, A., Schröder, P., and Sweldens, W. Hybrid meshes: multiresolution using regular and irregular refinement. In *Proceedings of the 18th Annual Symposium on Computational Geometry SCG*, pages 264–272, 2002.

[GO97] Goodman, J. E. and O'Rourke, J., editors. *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.

[Gol91] Goldberg, D. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[GSS99] Guskov, I., Sweldens, W., and Schröder, P. Multiresolution signal processing for meshes. In *Proceedings of ACM SIGGRAPH*, pages 325–334, 1999.

[GTLH01] Guéziec, A., Taubin, G., Lazarus, F., and Horn, B. Cutting and stitching: converting sets of polygons to manifold surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 7(2), 2001.

[Hal02] Halperin, D. Robust geometric computing in motion. *International Journal of Robotics Research*, 21(3):219–232, 2002.

[HDD+94] Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., and Stuetzle, W. Piecewise smooth surface reconstruction. In *Proceedings of ACM SIGGRAPH*, pages 295–302, 1994.

[HG97] Heckbert, P. S. and Garland, M. Survey of polygonal surface simplification algorithms. Technical report, Carnegie Mellon University, 1997.

[HG00] Hubeli, A. and Gross, M. A survey of surface representations for geometric modeling. Technical report, ETH Zurich, Switzerland, 2000.

[Hig96] Higham, N. J. *Accuracy and stability of numerical algorithms*. SIAM, 1996.

[Hof89] Hoffmann, C. M. *Geometric and solid modeling: an introduction*. Morgan Kaufmann Publishers, Inc., 1989.

[Hof01] Hoffmann, C. M. Robustness in geometric computations. *Computing and information science in engineering*, 1:143–156, 2001.

[Hop96] Hoppe, H. Progressive meshes. In *Proceedings of ACM SIGGRAPH*, pages 99–108, 1996.

[HS05] Hoffmann, C. M. and Stewart, N. F. Accuracy and semantics in shape-interrogation applications. *Graphics Models*, 67(5):373–389, 2005.

[HW07] Huang, Z. and Wang, G. Distance between a catmull-clark subdivision surface and its limit surface. In *Proceedings of ACM Symposium on Solid and Physical Modeling*, pages 233–240, 2007.

[Ind97] Industrial automation systems and integration—Product data representation and exchange—Part 42. International Organization for Standardization (ISO),. *STEP International Standard, ISO 10303-42*, 1997. Reference Number ISO 10303-42 1994(E). First Edition 1994-12-15.

[JDD03] Jones, T. R., Durand, F., and Desbrun, M. Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics*, 22(3):943–949, 2003.

[JLSW02] Ju, T., Losasso, F., Schaefer, S., and Warren, J. Dual contouring of hermite data. *ACM Transactions on Graphics*, 21(3):888–895, 2002.

[Ju04] Ju, T. Robust repair of polygonal models. *ACM Transactions on Graphics*, 23(3):888–895, 2004.

[KBF05] Kasik, D. J., Buxton, W., and Ferguson, D. R. Ten cad challenges. *Computer Graphics and Applications, IEEE*, 25(2):81–92, 2005.

[KBK02] Kahlesz, F., Balázs, A., and Klein, R. Multiresolution rendering by sewing trimmed nurbs surfaces. In *7th Symposium on Solid Modeling and Application*, pages 17–21, 2002.

[KBSS01] Kobbelt, P. L., Botsch, M., Schwanecke, U., and Seidel, H. P. Feature sensitive surface extraction from volume data. In *Proceedings of ACM SIGGRAPH*, pages 57–66, 2001.

[KMP+04] Kettner, L., Mehlhorn, K., Pion, S., Schirra, S., and Yap, C. Classroom examples of robustness problems in geometric computations. In *Proceedings of the 12th annual European Symposium (ESA)*, pages 702–713, Bergen, Norway, September 2004. Springer.

[KMP+06] Kettner, L., Mehlhorn, K., Pion, S., Schirra, S., and Yap, C. Reply to "Backward error analysis ...". In *Lecture Notes in Computer Science*, volume 3980, page 60, 2006.

[Kob96] Kobbelt, L. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. *Computer Graphics Forum (Proceedings of Eurographics)*, 15(3):409–420, 1996.

[Kob98] Kobbelt, L. Tight bounding volumes for subdivision surfaces. In Werner, B., editor, *Pacific Graphics*, pages 17–26, 1998.

[Kob00] Kobbelt, L. $\sqrt{3}$ subdivision. In *Proceedings of ACM SIGGRAPH*, pages 103–112, 2000.

[KS86] Kirkpatrick, D. G. and Seidel, R. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1), 1986.

[LC06] Lai, S. and Cheng, F. Voxelization of free-form solids using Catmull-Clark subdivision surfaces. In *Lecture Notes in Computer Science*, volume 4077, pages 595–601. Springer, 2006.

[LC07] Lai, S. and Cheng, F. Robust and error controllable Boolean operations on free-form solids represented by Catmull-Clark subdivision surfaces. *Computer Aided Design & Applications*, 4(1-4), 2007.

[LDW97] Lounsbery, M., DeRose, T. D., and Warren, J. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, 1997.

[LFKN03] Lanquetin, S., Foufou, S., Kheddouci, H., and Neveu, M. Trois algorithmes d'intersection des surfaces de subdivision. *Revue internationale de CFAO et d'informatique graphique*, pages 247–264, 2003.

[Loo87] Loop, C. T. Smooth subdivision surfaces based on triangles. Master's thesis, Department of mathematics, University of Utah, August 1987.

[Lou94] Lounsbery, M. *Multiresolution analysis for surfaces of arbitrary topological type*. PhD thesis, Department of Computer Science, University of Washington, 1994.

[LSS+98] Lee, A. W. F., Sweldens, W., Schröder, P., Cowsar, L., and Dobkin, D. MAPS: Multiresolution adaptive parameterization of surfaces. *Computer Graphics*, 32:95–104, 1998.

[Man88] Mantyla, M. *An Introduction to Solid Modeling*. Computer Science Press, 1988.

[MB79] Moore, R. E. and Bierbaum, F. *Methods and Applications of Interval Analysis (SIAM Studies in Applied and Numerical Mathematics) (SIAM Studies in Applied Mathematics, 2.)*. SIAM, 1979.

[MMTP04] Ma, W., Ma, X., Tso, S.-K., and Pan, Z. A direct approach for subdivision surface fitting from a dense triangle mesh. *Computer-Aided Design*, 36(6):525–536, 2004.

[Moo66] Moore, R. E. *Interval analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

[MP07] Melquiond, G. and Pion, S. Formally certified floating-point filters for homogeneous geometric predicates. *Theoretical informatics and applications*, 41:57–69, 2007.

[Muk05] Mukundan, H. Surface-surface intersection with validated error bounds. Master's thesis, Department of Ocean engineering and Department of Mechanical Engineering, MIT, 2005.

[NT03] Nooruddin, F. and Turk, G. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, 2003.

[Ove01] Overton, M. L. *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM, 2001.

[Owe98] Owen, S. J. A survey of unstructured mesh generation technology. In *Proceedings of the 7th International Meshing Roundtable*, pages 239–267, October 1998.

[PH98] Patterson, D. L. and Hennessy, J. L. *Computer organization and design: the hardware/software interface*. Kaufmann, San Mateo, CA, second edition, 1998.

[PM02] Patrikalakis, N. M. and Maekawa, T. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer, 2002.

[PM08] Patel, P. S. and Marcum, D. L. Robust and efficient CAD topology generation using adaptive tolerances. *International Journal for Numerical Methods in Engineering*, 75:355–378, 2008.

[PS85] Preparata, F. P. and Shamos, M. I. *Computational Geometry: an Introduction*, chapter 3. Springer-Verlag, 1985.

[PT97] Piegl, L. and Tiller, W. *The NURBS book*. Springer, 1997.

[Req99] Requicha, A. A. G. Geometric modeling: a first course. Course notes, http://www-pal.usc.edu/ requicha/book.html, 1999.

[Sar03] Sarfraz, M. *Advances in Geometric Modeling*. Wiley, 2003.

[SB00] Shu, C. and Boulanger, P. Triangulating trimmed NURBS surfaces. In *International Conference on Curves and Surfaces*, pages 381–388, 2000.

[Sch96] Schweitzer, J. E. *Analysis and application of subdivision surfaces*. PhD thesis, University of Washington, 1996.

[Sch99] Schirra, S. *Handbook of computational geometry*, chapter: Robustness and precision issues in geometric computation. Elsevier Science, B. V. North-Holland, Amsterdam, 1999.

[SD07] Smith, J. M. and Dodgson, N. A. A topologically robust algorithm for Boolean operations on polyhedral shapes using approximate arithmetic. *Computer-Aided Design*, 39(2):149–163, 2007.

[SG05] Shih, F. Y. and Gaddipati, V. Geometric modeling and representation based on sweep mathematical morphology. *Information Sciences – Infmatics and Computer Science: an International Journal*, 171(1-3), 2005.

[She99] Shewchuk, J. R. Lecture notes on Delaunay mesh generation, September 1999.

[Spa98] Spatial Technology, Boulder, CO. *ACIS 3D Toolkit*, 1998.

[Sta98a] Stam, J. Evaluation of Loop subdivision surfaces. In *Proceedings of ACM SIGGRAPH (CDROM)*, 1998.

[Sta98b] Stam, J. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *Proceedings of ACM SIGGRAPH*, pages 395–404, 1998.

[Sug02] Sugier, J. Triangulation of NURBS surfaces through adaptive refinement. *International Journal of Machine Graphics & Vision*, 11(1), 2002.

[SWC00] Steinbrenner, J. P., Wyman, J., and Chawner, J. R. Fast surface meshing on imperfect CAD models. In *Proceedings of the 9th International Meshing Roundtable*, pages 33–41, 2000.

[TTSC91] Toriya, H., Takamura, T., Satoh, T., and Chiyokura, H. Boolean operations for solids with free-form surfaces through polyhedral approximation. *Visual Computer*, 7(2-3):87–96, 1991.

[VZ01] Velho, L. and Zorin, D. 4–8 subdivision. *Computer-Aided Geometric Design*, 18(5):397–427, 2001. Special Issue on Subdivision Techniques.

[Wil60] Wilkinson, J. H. Error analysis of floating-point computation. *Numerical Mathematics*, 2:319–340, 1960.

[WP04] Wu, X. and Peters, J. Interference detection for subdivision surfaces. *Computer Graphics Forum*, 23(3):577–584, 2004.

[WP05] Wu, X. and Peters, J. An accurate error measure for adaptive subdivision surfaces. In *Shape Modeling International*, pages 51–56, 2005.

[Yap01] Yap, C. Survey and recent results: robust geometric computation. Talk at MIT, October 2001.

[Yap04] Yap, C. In Goodman, J. E. and O'Rourke, J., editors, *Handbook of Discrete and Computational Geometry*, chapter : Robust geometric computation. CRC Press LLC, Boca Raton, FL, 2 edition, 2004.

[Yap06] Yap, C. Theory of real computation according to EGC. In *Proceedings of Dagstuhl seminar on Reliable Implementation on Real Numer Algorithms: Theory and Practice*, 2006.

[Zor97] Zorin, D. *Subdivision and multiresolution surface representations*. PhD thesis, Caltech, 1997.

[ZSD+00] Zorin, D., Schröder, P., DeRose, T., Kobbelt, L., Levin, A., and Sweldens, W. Subdivision for modeling and animation. ACM SIGGRAPH course notes, 2000.

[ZSS96] Zorin, D., Schröder, P., and Sweldens, W. Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of ACM SIGGRAPH*, pages 189–192, 1996.

[ZSS97] Zorin, D., Schröder, P., and Sweldens, W. Interactive multiresolution mesh editing. In *Proceedings of ACM SIGGRAPH*, pages 259–268, 1997.