Université de Montréal

Reliable Solid Modelling Using Subdivision Surfaces

par
Peihui Shao

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

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

février, 2013

Université de Montréal
Faculté des arts et des sciences

Ce mémoire de maîtrise intitulé:

Reliable Solid Modelling Using Subdivision Surfaces

présenté par:

Peihui Shao

a été évalué par un jury composé des personnes suivantes:

Pierre Poulin
président-rapporteur

Neil Stewart
directeur de recherche

Jean Meunier
membre du jury

RÉSUMÉ

Les surfaces de subdivision fournissent une méthode alternative prometteuse dans la modélisation géométrique, et ont des avantages sur la représentation classique de *trimmed-NURBS*, en particulier dans la modélisation de surfaces lisses par morceaux. Dans ce mémoire, nous considérons le problème des opérations géométriques sur les surfaces de subdivision, avec l'exigence stricte de forme topologique correcte. Puisque ce problème peut être mal conditionné, nous proposons une approche pour la gestion de l'incertitude qui existe dans le calcul géométrique.

Nous exigeons l'exactitude des informations topologiques lorsque l'on considère la nature de robustesse du problème des opérations géométriques sur les modèles de solides, et il devient clair que le problème peut être mal conditionné en présence de l'incertitude qui est omniprésente dans les données. Nous proposons donc une approche interactive de gestion de l'incertitude des opérations géométriques, dans le cadre d'un calcul basé sur la norme IEEE arithmétique et la modélisation en surfaces de subdivision. Un algorithme pour le problème *planar-cut* est alors présenté qui a comme but de satisfaire à l'exigence topologique mentionnée ci-dessus.

Mots clés : incertitude, robustesse, calcul géométrique, arithmétique en précision finie, arithmétique des intervalles, surfaces de subdivision.

ABSTRACT

Subdivision surfaces are a promising alternative method for geometric modelling, and have some important advantages over the classical representation of trimmed-NURBS, especially in modelling piecewise smooth surfaces. In this thesis, we consider the problem of geometric operations on subdivision surfaces with the strict requirement of correct topological form, and since this problem may be ill-conditioned, we propose an approach for managing uncertainty that exists inherently in geometric computation.

We take into account the requirement of the correctness of topological information when considering the nature of robustness for the problem of geometric operations on solid models, and it becomes clear that the problem may be ill-conditioned in the presence of uncertainty that is ubiquitous in the data. Starting from this point, we propose an interactive approach of managing uncertainty of geometric operations, in the context of computation using the standard IEEE arithmetic and modelling using a subdivision-surface representation. An algorithm for the planar-cut problem is then presented, which has as its goal the satisfaction of the topological requirement mentioned above.

Keywords: uncertainty, robustness, geometric computation, finite-precision arithmetic, interval arithmetic, subdivision surfaces.

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank, first of all, my director of research Neil Stewart. Without his valuable guidance, his encouragement and our regular discussions through all my research, this work would have been impossible.

I also wish to thank Pierre Poulin and Jean Meunier for accepting to read this thesis and be the jury members. I would like to thank other people in the lab LIGUM, Derek Nowrouzezahrai, Mohamed Yessine Yengui, Gilles-Philippe Paillé, Dorian Gomez, Jonathan Dupuy, for our discussions and for their help with many technical details.

Finally, I owe my special acknowledgment to my wife, Yulin, for her encouragement and company in my life, and to my parents, for their support and encouragement these past years.

CHAPTER 1

INTRODUCTION

With the advent of modern computers, numerical analysis has become an important discipline of applied mathematics. Nowadays, in almost all fields of science and engineering numerical computations are extensively used and have propelled the development of these fields. Due to the application of finite-precision arithmetics and uncertainty of data, which is almost inevitable in practice, however, we have to know how reliable these computations are, before we can trust them as acceptable solutions to our real-life problems. Unreliable computations may lead to disastrous accidents, such as the famous Patriot missile failure during the Gulf War that turned out to be caused by an inaccurate calculation of time.

In the field of geometric computing, numerical analysis plays the same role and leads to similar difficulties. The most widely used representation for solid modelling in the CAD/CAM area nowadays is trimmed-NURBS (Non-Uniform Rational B-Splines), which permits manipulation of both analytical and free-form solids with great flexibility and efficiency, making it the industry standard for representing solids. This representation, however, suffers greatly from its inherent limitations in describing complex free-forms—possible gaps and overlaps may exist between NURBS patches. This is a serious problem to the interoperability of different CAD/CAM systems, incurring very high monetary cost to industry: manual work that is needed to repair the artifacts is reported to cost the U.S. automotive industry over $600 million per year [BM99].

[1] Other methods of representation, such as subdivision-surface [AS10, Sch96] and multiresolution [KCVS98, Zor06] models are now finding more frequent use in graphics applications, including game and film animations [DKT98]. While it is likely that the trimmed-NURBS approach will remain standard in computer-aided

[1]The following two paragraphs are taken almost verbatim from a draft document [Ste11] by N. F. Stewart.

design for some time, due to the large amount of legacy code that exists for this kind of representation [Spa99, ISO97], it seems worthwhile to consider what might be achieved in terms of formal guarantees of reliability using subdivision-surface models. Since the costs of unreliable models are so high in the context of computer-aided design, a shift away from trimmed-NURBS representations may eventually occur.

The problem of numerical reliability is often studied in the context of regularized Boolean operations on objects [ASZ07, Req80], because these operations are important and typical of the most complicated operations that must be implemented. It is also true, however, that the problem of computing a regularized Boolean operation is inherently ill-conditioned [ASZ07]. The appropriate response in such a situation (consider, for example, the classical problem of polynomial approximation [FM67, Sec. 19]) is to appropriately reformulate the problem. In this thesis, we take this as the starting point. We seek to reformulate the ill-conditioned problem of computing a regularized Boolean operation, in the context of subdivision-surface models, in such a way that the reformulated operation corresponds to the needs of the user, and in such a way that a reliability proof is possible.

Note that we do not propose new algorithms for Boolean operations on solids here; rather, we will show how to assemble certain known methods in a way that permits us to define operations for which rigorous theorems can be proved. Also, rather than studying general Boolean operations, we study the special case of the planar-cut problem.

The outline of this work is as follows: first, a summary about errors and uncertainty in numerical analysis and especially in geometric computations is presented in Chapter 2. Some examples are also given to illustrate important concepts and methods in error analysis. In Chapter 3 we give an introduction to popular representations (meshes, trimmed-NURBS, and subdivision surfaces) and operations in solid modelling systems. Since error analysis in Boolean operations on subdivision surfaces is our ultimate objective, two approaches to implementation are presented and compared in Chapter 4. We will use the surface-based one in our

work. In Chapter 5 we conclude that the problem of Boolean operations on solids is ill-conditioned and propose to reformulate the problem of Boolean operations so that we can deal with ill-condition appropriately. An algorithm based on cell bindings is then described. We also propose to eliminate self-intersections in the computed solid by converting it into a Bézier complex and applying conditions that exclude self-intersections of composite Bézier curves and patches. In Chapter 6 an interactive approach for the robust implementation of the planar-cut problem with a subdivision-surface representation, is proposed, in the context of computation using the standard IEEE arithmetic and when the correct topologial form of the result is required in the problem definition. As this approach is performed with the backward error analysis, we complement it with the approach that deals with the same problem, but uses the forward error analysis. This is presented in the final chapter. We also give a summary of the thesis, and possible ways to improve the work in the future.

CHAPTER 2

ERROR ANALYSIS FOR GEOMETRIC COMPUTATIONS

In numerical analysis, errors are unavoidable in all kinds of practical algorithms and sometimes render the results completely meaningless. In the development of algorithms, dealing with rounding errors is only one of the problems of numerical analysis. In [Tre92], the definition of numerical analysis is given as "the study of algorithms for the problems of continuous mathematics", which emphasizes that one major task of numerical analysis is to reduce the impacts of possible errors to the final outputs.

Possible sources of errors in numerical analysis are rounding, truncation, and data uncertainty [Hig02]. The use of finite-precision arithmetic is directly responsible for the rounding errors, while the alternatives such as exact arithmetic and interval arithmetic, may not suffer from this type of errors, but they are far from capable of playing the role of finite-precision arithmetic in practice [Hof01]. The problem of robustness in geometric computations, however, involves not only the errors mentioned above, but also certain intrinsic properties of geometric modelling, as will be discussed in this chapter. Overviews of robustness in geometric computations can be found in [ASZ07, Pat02, Hof01].

In this chapter, we present a brief overview of error analysis theory, introducing some concepts such as condition number and backward error analysis. We then talk about errors in geometric computations, with a practical example to illustrate inconsistency in geometric representations. Finally, a framework that can be used to prove the robustness of geometric algorithms with backward error analysis is described.

2.1 Error analysis

2.1.1 Sources of errors

First, errors of the sort found in the field of numerical analysis can be generally found in geometric computations. But we should also be concerned about other types of errors that are due to the natural properties of geometric modelling, and that are often considered difficult in analysis and treatment.

In the field of numerical analysis, there are mainly three kinds of errors: rounding, data uncertainty, and truncation [Hig02]:

- *Rounding errors* (or round-off errors) are unavoidable due to use of finite-precision arithmetic; they are differences between the calculated values and the true mathematic values. This type of error can be exemplified by the difference between the true value of the irrational golden ratio $\varphi = \frac{1+\sqrt{5}}{2}$ and its approximate value 1.6180 using a floating-point number rounded to four decimal places. This is a representation error, which can be amplified during subsequent calculations, even to such an extent that the final result is rendered meaningless.

- *Truncation errors* arise from some numerical algorithms (for example, Euler's method for solving ordinary differential equations) that take a finite number of terms from the infinite Taylor series. This type of error is inevitable even with infinite-precision arithmetic, because truncation of Taylor series to a finite number of terms for these numerical algorithms is a necessary step. Increasing the number of terms renders the computation slower, while omitting more terms reduces the accuracy of the result. Thus the optimization of such algorithms is a compromise between accuracy and computation speed [BS92].

- *Data uncertainty* may arise from measurement uncertainty, from rounding errors in storing the data, or from errors propagated from previous computations.

Most standard textbooks on numerical analysis, for example [Hig02, BF10], generally emphasize finite precision computation and rounding errors, sometimes neglecting other types of errors, such as truncation errors and measuring errors. This may give the impression that the study of reliable numerical methods mainly deals with the study of rounding errors. This is however far from true [Tre92].

In the field of geometric computation, another common source of errors is approximations used in geometric algorithms [Jia08, Section 2.1]. The idea of approximation is seen in various scientific domains, and in free-form modelling it is especially necessary due to the need for efficient computation and rendering of complex analytical surfaces and curves. Low-degree curves are often used to approximate high-degree curves: for example, when computing the intersection of two Bézier/B-spline surfaces, a low-degree approximation can dramatically reduce the computational work that is difficult or even impossible to complete. Polygonal representations that are used as approximations of the NURBS or subdivision-surface objects are often preferred over the original free-form representations, when we have to perform some tasks that do not need precise representations.

It is often needed to make decisions about position relations between geometric elements that are adjacent in space, for example, the decisions we have to make on the two objects S_0 and S_1 in Example 2.2.2 of page 14, which will be discussed later. This is, however, another source of errors in geometric computations.

2.1.2 The IEEE floating-point number system

The newest IEEE standard for Floating-Point Arithmetic (IEEE 754-2008) [IEE08] is now the most widely accepted standard for floating-point computation. It defines basic formats for 32, 64, or 128 bit encoding, arithmetic formats, and interchange formats. It describes five rounding algorithms. This standard also specifies exceptions and the default handling actions.

Despite the fact that floating-point arithmetic has drawbacks due to its impreciseness, it has numerous advantages and is a fundamental "language" in engineering and scientific computation. The IEEE standard for floating-point arithmetic is

well-supported by various general-purpose and specialized programming languages, and has been adopted and optimized in current computer hardware and software [For92].

2.1.3 Backward error analysis

Suppose that a numerical algorithm is modelled by a function $y = f(x)$, with input data x and output data y. The quantity ${y_1}'$ is the computed value of $f(x_1)$, and y_1 is the true value of $f(x_1)$, that is, ${y_1}' \approx y_1 = f(x_1)$. We have two basic forms of error analysis for this algorithm [Hig02, DB07]:

Forward error analysis:

With it, one tries to bound the *forward error*, which is the absolute error in the output $|{y_1}' - y_1|$, or the relative error $|\frac{{y_1}'-y_1}{y_1}|$.

Backward error analysis:

With it, one tries to find the *backward error*, by perturbing the input data by a small value $|\Delta x_1|$ such that the computed output y_1 is exact $f(x_1 + \Delta x_1)$, and bounding this small value $|\Delta x_1|$. We may find many such Δx_1; in this case, we seek to find a minimum value for $|\Delta x_1|$, possibly divided by $|x_1|$, as the backward error.

Backward error analysis is the preferred error-analysis method, whose main advantage is that once the backward analysis has been performed, it makes no reference to the exact solution and reduces the bounding of the forward error to a perturbation analysis, subject to the input having been perturbed by rounding and truncation errors [Hig02].

Example 2.1.1. *(Backward analysis of a linear system)*

Here an example is given to illustrate the idea of backward error analysis. Consider a linear system $Ax = b$; the problem is given by the input matrix A and the input vector b. Through a certain algorithm, we have obtained a numerical solution y which can be viewed as an approximation of the exact solution x. Suppose that y

is the exact solution to some perturbed problem $(A + \Delta A)y = b + \Delta b$*, and denote that* $\delta = \|\Delta A\|/\|A\| + \|\Delta b\|/\|b\|$*, where* $\|\cdot\|$ *denotes the norm of the vector or matrix. We then bound the backward error of this algorithm as the minimum of* δ*.*

2.1.4 Condition number of a problem

To measure how sensitive the output is to small changes in the input, we call a problem *ill-conditioned* when "small" changes in the input can cause "large" changes in the output; otherwise we call it *well-conditioned*. The definitions of "large" and "small" can only give us a coarse impression of how well the problem is formed, i.e., how stable the output is when subjected to a perturbation in the input. In fact, this property of the problem can be measured more precisely by the *condition number* [Hig02].

The condition number of a problem relates its forward error and backward error, according to a simple inequality:

$$e_f \lessapprox \lambda \times e_b \tag{2.1}$$

where e_f, e_b, and λ denote the forward error, the backward error, and the condition number, respectively.

From this relation it is easy to see that a solution to an ill-conditioned problem can produce a much larger error in comparison with the perturbations in the input. In this case, we actually have no choice except to reformulate the problem, with the hope to convert it into a well-conditioned one, since as seen from this inequality, we will get a large error anyway when the condition number is large, due to uncertainty in the input data. Only for a well-conditioned problem can we expect an algorithm that can solve the problem accurately.

Example 2.1.2. *(Condition number of a matrix)*

In Example 2.1.1, suppose further that A *is a square nonsingular matrix and* $b \neq 0$*. A perturbation analysis gives the following result (neglecting second-order*

terms):

$$\frac{\|x-y\|}{\|x\|} \lessapprox \|A\| \cdot \|A^{-1}\| \left\{ \frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \right\} \tag{2.2}$$

where $\|\cdot\|$ *denotes a matrix norm subordinate to the vector norm when we measure the error of the solution* y*. The condition number* λ *for this problem is expressed as* $\lambda = \|A\| \cdot \|A^{-1}\|$*, which is often called the* condition number of a matrix*. We also have the forward error* $e_f = \frac{\|x-y\|}{\|x\|}$*, and the backward error* $e_b = \frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|}$*.*

This inequality takes the same form as (2.1) and shows the relation between the forward error, the backward error, and the condition number in a linear system.

The Hilbert matrix H_n *of order* n *is a square matrix with entries of the form*

$$H_{ij} = \frac{1}{i+j-1}, 1 \leq i, j \leq n.$$

It is considered as a canonical example of ill-conditioned matrices, with the condition number under the L^2 *norm being as large as* 4.753×10^8 *when order* $n = 7$ *[DB07].*

2.2 Robustness in geometric computations

As discussed in the previous section, numerical geometric computations not only fail due to common errors that are often seen in numerical analysis, but also suffer from robustness problems that stem from certain natural properties of geometric modelling, for example, inconsistency in the representations of models and approximations used in geometric operations. In this section, we will detail the uncertainty in geometric representations, discuss the problem of metrics for the measurement of error in geometric modelling, and give a framework to analyze the possible errors in geometric computations.

2.2.1 Inconsistency in the representation

As we will talk about in Sections 3.1.3 and 3.1.4, there exist two main representation schemes for solids: CSG (Constructive Solid Geometry) and B-rep (Boundary

Representation). CSG describes a solid implicitly using a set of primitives and some basic operations, i.e., rigid body motion and the regularized Boolean operations. It guarantees that the resulting solids are water-tight if its primitives are valid r-sets. This scheme is conceptually easy, defining objects in a constructive manner, however, the kernels of modern solid modellers actually use a B-rep representation scheme, which is more flexible and supports a much richer set of operations. B-rep is also more convenient for graphical display. In this work, our discussion is focused on B-rep, which is the representation scheme that most modern geometric modellers adopt in their kernels.

Inconsistency within the B-rep may exist between its topological data and geometric data, and its geometric data may have redundancy for supporting various manipulations and interrogations conveniently, which almost certainly leads to data conflicts. Due to the use of finite-precision numbers, i.e., floating-point representation, and the use of numerical approximations (for example, to approximate high-degree intersection curves with low-degree polynomials) [HS05], the representations are fundamentally inconsistent, especially in the case of solids composed of free-form surfaces. This is a main bottleneck for the downstream applications when the inconsistent data is imported to other systems [Far99].

Example 2.2.1. *(Intersection between two trimmed-NURBS patches)*

One example from [ASZ07] for inconsistency in the representation of models can be illustrated by Figure 2.1 [ASZ07, p. 13]. Topologically, the two trimmed-NURBS patches are a 2-chain sharing an edge, as shown on the right of Figure 2.1. Geometrically, these two patches are supposed to join along the intersection of two faces F and F', restricted to two parametric domains D and D', and their actual boundaries are stored as numerical data by their pre-images, i.e., the two p-curves p and p'. There exist also two other parts of the representation: the boundary curve between F and F', denoted $b(t), t \in [0, 1]$, and a corner vertex v for each 0-cell in the symbolic data. In practice it is very likely that these representations have conflicts, i.e. the images of the p-curves p and p' do not coincide with the boundary curve b, and the corner vertex v may not fall precisely on b. The representation of

trimmed-NURBS will be discussed in more detail in Section 3.3.

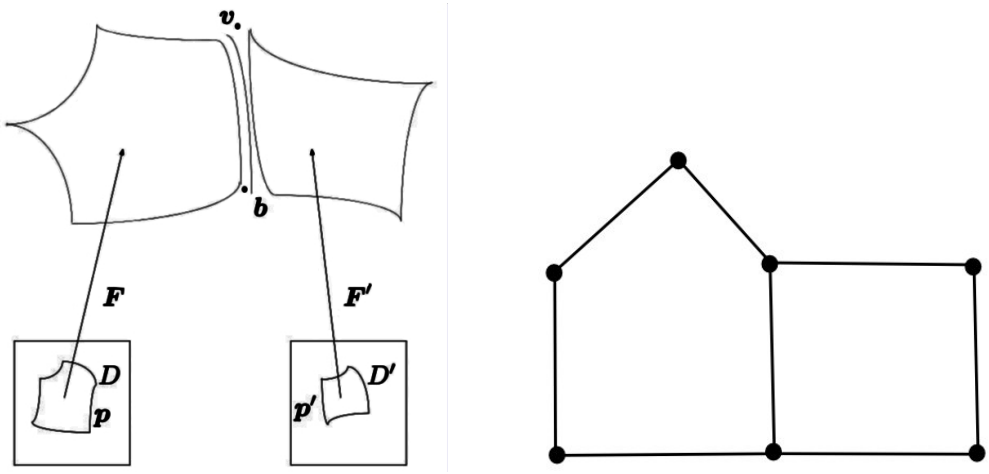

Figure 2.1: Left: two adjoining trimmed-NURBS patches. Right: their topological data [ASZ07].

2.2.2 Metrics for the measurement of error

To judge the quality of a computed result, or to measure the distance between a computed solution and a true solution, metrics for the measurement of error are needed [HS05]. The reason why the metrics for error measurement in solid modelling need special discussion is that, the outputs are generally geometric models, and the difference between the true model and the computed model needs to be quantified.

For some applications in solid modelling, a metric should require that two objects considered to be close, for example, a computed object and the one described in the problem, should have the same topological form. This requirement can have different interpretations: a stringent metric can require an ambient isotopy [APS00] between two close geometric objects. This metric, however, does not take into account narrow cracks or thin protrusions. Metrics that take into account these possible features are discussed in detail in [BS92].

2.2.3 Conditioning of geometric problems

The conditioning theory is important in that when we encounter ill-condition of a problem, we would not expect to find an accurate algorithm for this problem. This can be easily illustrated by a geometric example about Boolean operations between two solids in the following.

Example 2.2.2. *(Conditioning of geometric problems)*

*Consider the problem illustrated in Figure 2.2, which is also mentioned in [HS05]. Note that this example corresponds to a practical real-life requirement of feature-modelling systems. In the figure, a bevel is defined as the regularized Boolean difference between the block S_0 and the wedge S_1. The result from this Boolean operation, $S_0 -^* S_1$ will, however, contain undesirable artifacts unless some extra information is given. Here, even the exact calculation of $S_0 -^* S_1$ would be insufficient for extracting the exact topological form of the result because of uncertainty in the data. A typical example, obtained by actual floating-point computation, is shown in Figure 2.3, where a spurious artifact appears in the exact solution of the set-difference problem. The left of Figure 2.3 shows the result of the Boolean difference operation. The right is produced by double-precision computation and displayed using the OpenGL library, and double precision here is sufficient to show the form of the **exact** set $S_0 -^* S_1$. The corresponding point A, and the corresponding edges a, b, c, and e, are marked in this figure, while the dotted line d on the right is a hidden line in the solid obtained.*

This example demonstrates that uncertainty in the geometric data is almost inevitable with finite-precision arithmetic, and that some applications such as Boolean operations are especially vulnerable when exposed to such uncertain data, because they are ill-conditioned when no extra information is provided about the topology, i.e., even a small error in the input can lead to a large error in the result. For example, a very small perturbation in the wedge position (Figure 2.2) can incur a result that is completely different with regard to the topology, e.g., a solid having thin cracks or protrusions. Figure 2.3 shows such a solid: two faces of the wedges

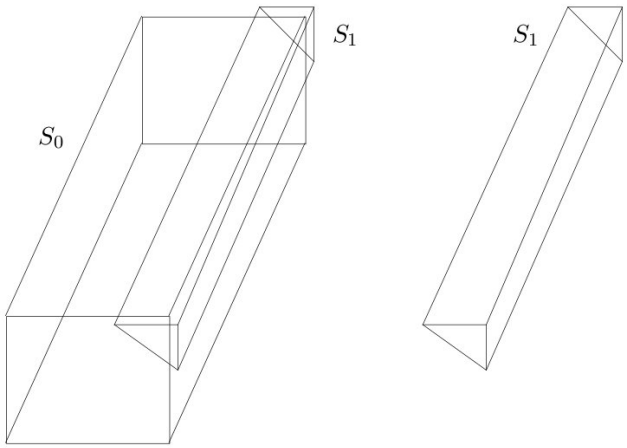

Figure 2.2: Wedge to be subtracted from a block.

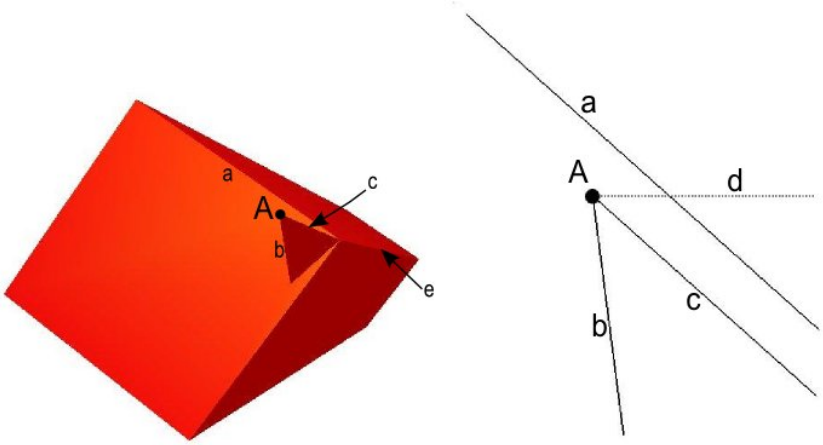

Figure 2.3: Artifact produced using IEEE double precision.

do not coincide with the corresponding faces of the block, and the result has two protruded parts that touch at the edge e of the block, forming a hole (drawn in black). Note that the thickness of the two protrusions has been exaggerated on the left of Figure 2.3 so that they are visible.

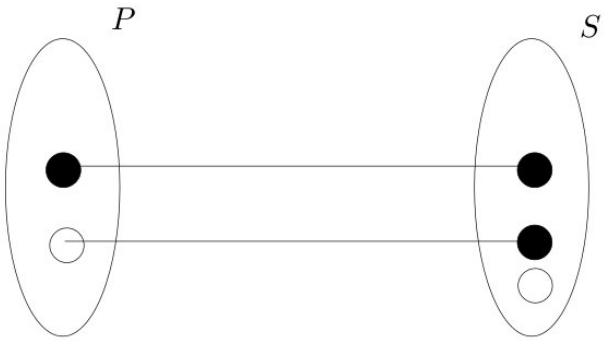

Figure 2.4: A well-conditioned problem.

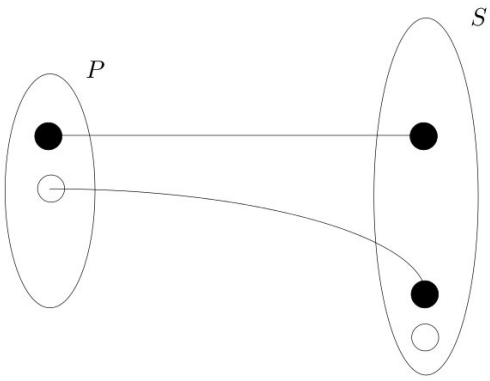

Figure 2.5: An ill-conditioned problem.

The general situation can be illustrated in the following way: in Figure 2.4 [ASZ07, p. 19], a small perturbation does not give us a big perturbation in the result for well-conditioned problems, while in Figure 2.5 [ASZ07, p. 20], a small perturbation is enlarged in the result for ill-conditioned problems. (In these two figures, P denotes the class of problems, and S the class of solutions. Filled circles in P denote the given problems, and unfilled circles in P denote problems that are

presented to the method. The presented problems are always slightly perturbed due to uncertainty in the data. Filled circles in S denote true solutions to the given problems or to the presented problems, and unfilled circles in S denote computed solutions for the presented problems.) Now, we can see that a stable method does not necessarily produce a small error. We have to take the condition of the problem into account.

By now, we can conclude that for a numerical problem in geometric modelling, error in the result has two sources: it may come from the ill-condition of the problem itself, or it may be due to robustness issues of the method that is used to solve the problem. If we can prove that the perturbation resulting from the method is always no larger than the perturbation in the uncertain data, then if the method is stable it provides us the results that are as good as the data warrant.

2.3 Proof of robustness for geometric algorithms

Boolean operations, as a fundamental set of operations in solid modelling, have been supported by most modern modelling systems. They are however ill-conditioned and error-prone, subject to inconsistency in the representations. Detecting and removing irregularities such as cracks and protrusions is a tedious and costly task, and as we have seen, impossible to do without external guidance. Many algorithms that are often called *model healers* have been proposed to repair model artifacts due to imperfect calculations of boundary intersections, for example, the artifacts in Figure 2.3. They can be roughly categorized into surface-based and volume-based algorithms. The first class of algorithms works on the input surfaces, trying to detect and resolve existing artifacts with the models. The second class of algorithms first converts the input into a volumetric representation, tries to detect and repair the artifacts on it, and converts it back into the surface representation by extracting the boundary surfaces. In this work, we will use a surface-based approach. More importantly, however, we aim to provide rigorous guarantees on the precision of the computed result.

The paper [ASZ07] can serve as a framework to prove the robustness of geometric algorithms, but the required error analyses are generally difficult, mainly due to the limitations of the representation of trimmed-NURBS. It is for this reason that we mainly consider subdivision-surface models for our research. The representation of subdivision surfaces will be discussed in more detail in Section 3.4.

CHAPTER 3

REPRESENTATIONS AND OPERATIONS IN SOLID MODELLING SYSTEMS

The notion of solid modelling first emerged in the 1970's, and was largely promoted by the seminal research at the University of Rochester [VR93]. Solid modelling is distinguished from other areas of geometric modelling and computer graphics mainly by its emphasis on physical fidelity [FHK02, Ch. 20]. Today, solid modelling has rapidly become a central area of research and development that involves a growing number of applications. It forms a foundation of CAD, and is indispensable in other modern industries, such as computer vision, medical modelling, architecture, and entertainment.

A brief introduction to solid modelling is first given in this chapter. The mathematical foundations for solids and the principal computer representation schemes (CSG and B-rep) are presented in this part. Then, three popular representation methods—meshes, trimmed-NURBS, and subdivision surfaces, all belonging to the B-rep scheme—are discussed, along with their respective power and limitations in application. Finally, the SGC (Selective Geometric Complexes) representation that is used in our work to enable bindings of topological cells is discussed.

3.1 Solid modelling

3.1.1 Introduction

A solid model represents a digital model of a three dimensional object that can exist physically. Solid modelling is a sub-discipline of computer graphics that concerns the creation, exchange, manipulation, and animation of solid models. The major difference between solid modelling and geometric modelling is that the shapes can be only three dimensional solids in solid modelling, but can be of any dimension in geometric modelling (although mostly of two or three dimensions). Solid mod-

elling has various applications in many industries, such as entertainment, medical imaging, and engineering, where models can be created using general purpose or specialized CAD systems.

For a summary of representation schemes and common operations in solid modelling, one can refer to [RR99, Man87] and [FHK02, Chapter 20].

3.1.2 Mathematics for solid representation

The standard mathematical representation of physically rigid solids is the r-set proposed in [Req80]. An r-set is *bounded*, *regular*, and *semi-algebraic*. For this definition, boundedness indicates that the set has finite contents, i.e., it can be bounded by a finite-size ball. Regularity means that the set is equal to the closure of its interior parts. A set is semi-algebraic when it can be expressed as a finite sequence of Boolean set operations (union, intersection, difference, etc.) of sets of the form $\{(x, y, z) \mid f(x, y, z) \leq 0\}$, where f is a polynomial function. See Figure 3.1 for a non-regular model and a valid solid model.

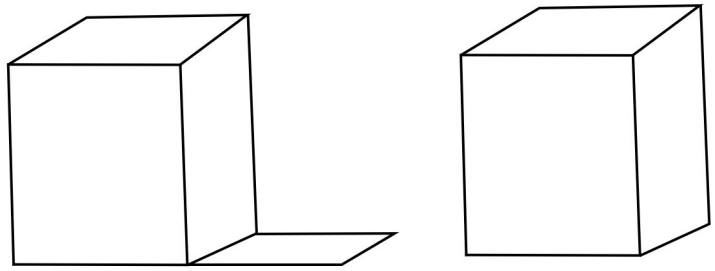

Figure 3.1: Left: an invalid solid model (non-regular). Right: a valid solid model.

An r-set can be manifold or non-manifold. A manifold can be viewed as a topological space that is locally Euclidean, i.e., any small enough neighbourhood near a point resembles the open unit ball, or the half unit ball in a specific dimension D. The open unit ball is the open interval $(-1, 1)$, if $D = 1$, or an open disk, if $D = 2$, etc. In solid modelling, a manifold has a dimension $D = 3$, and its manifold boundary has dimension $D = 2$. A non-manifold model in solid modelling

represents a solid that is "unmanufacturable", for example an object having two parts that touch only at a vertex or along an edge, as shown in Figure 3.2. Some solid modellers are limited to create and manipulate only manifolds. Use of such modellers must avoid certain operations that produce non-manifolds [RR99].

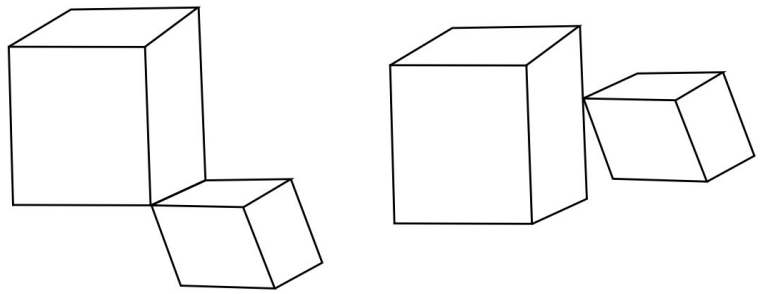

Figure 3.2: Examples of non-manifolds.

3.1.3 Constructive solid geometry

CSG (Constructive Solid Geometry) and B-rep (Boundary Representation) have been the principal representation schemes for solids. In Figure 3.3 [Str06, p. 6], a solid is represented using these two representation schemes, in which "−" denotes the Boolean difference, and "+" denotes the Boolean union. CSG represents solids as a hierarchy of Boolean operations that work on primitives, which are the simplest solid objects used to build more complex objects. Primitives can typically be parameterized, i.e., they can be described by a set of parameters (for example, a cuboid can be described by its central coordinates and its dimensions). The allowable operations on primitives are rigid motions (rotations and translations), positive scalings, and regularized Boolean operations which include union, intersection, and difference. A regularized Boolean operation is performed by imposing the topological interior operation and the topological closure operation on the result obtained from the corresponding classic Boolean operation. (Regularization is usually denoted by "*", see page 13) The reason that a regularized version of Boolean

operations is used in solid modelling, is to guarantee that the result remains valid as an r-set.

The main advantage of CSG is that it assures that the resulting solids are valid as r-sets, if all the primitives are valid. This is guaranteed by the nature of the affine transformation and regularized Boolean operations that are allowed with CSG. This property of CSG is very important for its applications in solid model computation and manufacturing. It is not true, however, that the resulting solids are manifolds when all the primitives are manifolds (see Figure 3.2 for examples).

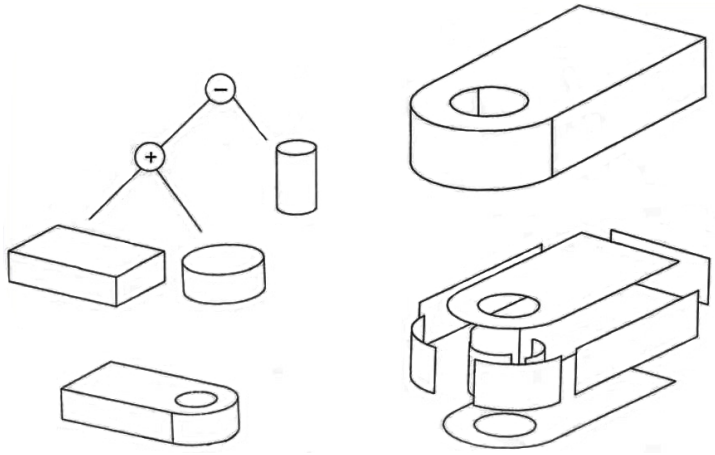

Figure 3.3: Left: a CSG model. Right: a B-rep model.

3.1.4 Boundary representation

B-rep is another popular representation scheme in solid modelling. It represents a solid model by its connected boundary elements—faces, edges, and vertices. In comparison with CSG, B-rep is more flexible, and supports a much richer set of operations, such as sweeping, blending, drafting, shelling, tweaking, and chamfer, which enables it to have extensive uses in different geometric applications. In fact, B-rep is what most solid modellers adopt as their internal representation method.

For efficiency and convenience in different geometric queries and operations, the data of B-rep models generally have two parts: topological data and geometrical

data. Topological data describe the topological connectivity of faces, edges, and vertices, while geometrical data define the geometrical positions of them. The fact that the data have its geometrical form and topological form, however, can potentially cause conflicts in data, as discussed in the last chapter.

The conversion from CSG to B-rep is usually possible [GO97], but its reverse conversion is ambiguous and more complicated in implementation [SV91]. Modern solid modelling systems generally use B-rep in their kernels, but may use multiple schemes to provide better support for model representation and manipulation.

Apart from these two schemes, there are other alternatives to represent solid models, one of which is called the *spatial decomposition scheme* [Sam90]. With this scheme, a solid is described by a list of the non-overlapping spatial cells occupied. The type of cells can be simply cubical, tetrahedra, or analytical shapes bounded by curved surfaces. The arrangement of cells may be regular, using an octree, or irregular, using for example a BSP (Boundary Space Partition) tree. We will use such a representation, namely the SGC (Selective Geometric Complexes), as part of the solution to our problem.

In the following, three popular representation methods in solid modelling are introduced. Each has its own advantages and drawbacks in different geometric applications. Mesh representation is conceptually simple and highly desirable for applications such as real-time displaying and manipulation; trimmed-NURBS is the *de facto* standard in CAD industry, and has been investigated for decades; the representation of subdivision surfaces is promising in that it can overcome some fundamental limitations of the trimmed-NURBS, for example, the difficult problem of the trimming operation.

3.2 Mesh representation

3.2.1 Definition and types of meshes

According to Bern and Plassmann [BP00], 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". Meshes can be classified in different ways, one way is to classify them by the dimensions—two or three. Another way is to classify them by the organization of elements: *structured*, *unstructured* and *hybrid*. Detailed information about mesh classification and the corresponding generation methods can be found in [BP00, Owe98].

3.2.2 Polygonal meshes

A polygonal mesh is composed of vertices, edges, and faces that define the shape of a geometric model, representing the boundary surfaces in the case of a solid model. The faces are usually triangles, quadrilaterals, or other simple convex polygons, of which triangles are the most widely adopted. For a comprehensive summary of polygonal meshes and the related operations, such as smoothing, decimating and remeshing, one can refer to [BPK+07]. Figure 3.4 shows a simple triangular mesh.

The representation of polygonal meshes is extensively used in computer graphics and solid modelling for its simplicity in rendering and representation, and for its power in approximating complex analytical solid models, e.g., NURBS and subdivision surfaces. Today's graphics hardware is most efficient in manipulating and rendering graphics primitives, such as points, lines, and triangles, and complex convex or concave shapes are usually decomposed into graphics primitives before rendering. More complicated cases involve free-form objects, which can not be output to the screen directly and are usually converted to an approximate polygonal representation before rendering.

3.3 Trimmed-NURBS representation

Trimmed-NURBS (Non-Uniform Rational B-Splines) have been commonly used in the CAD/CAM/CAE industry and form part of international and industrial standards, such as STEP [ISO97] and ACIS [Spa99]. They are also supported in various graphics programming interfaces, such as OpenGL, Direct3D, and Pixar's Renderman. NURBS are now the de facto industry standard for the representation and

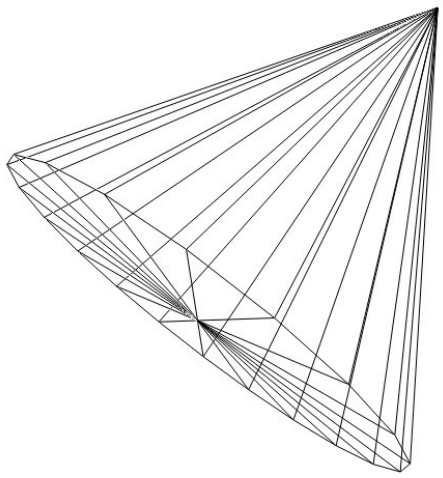

Figure 3.4: The triangular mesh representing a cone.

manipulation of geometric models. NURBS can be used to generate and represent both analytic shapes, such as quadric shapes, and free-form objects, such as car bodies.

3.3.1 Definition

NURBS surfaces can be viewed as generalizations of uniform nonrational B-spline surfaces and rational and nonrational Bézier curves and surfaces. The mathematical form of a NURBS surface of degree (p, q) in the directions u and v, respectively, can be defined as [PT95]

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,j}(u,v) P_{i,j} \qquad 0 \le u, v \le 1 \tag{3.1}$$

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

$$R_{i,j}(u,v) = \frac{N_{i,p}(u) N_{j,q}(v) w_{i,j}}{\sum_{k=0}^{n} \sum_{l=0}^{m} N_{k,p}(u) N_{l,q}(v) w_{k,l}}. \tag{3.2}$$

The $\{P_{i,j}\}$ form a bidirectional control net, where $P_{i,j} \in \mathbb{R}^3$, the $\{w_{i,j}\}$ are the weights, and the $\{N_{i,p}(u)\}$ and $\{N_{j,q}(v)\}$ are the nonrational B-spline basis functions defined on knot vectors.

3.3.2 The trimming operation

The main problem for modelling with trimmed-NURBS is with the trimming operation (see Figure 2.1, left), which is both expensive and error-prone. The trimming operation is a restriction of $S(u, v)$ to a parametric subdomain that is typically not rectangular, thus allowing for topologically arbitrary patches [CM99]. This restriction is specified by the trimming curves and some criteria under which the unwanted region of the original tensor-product NURBS surface patch is removed. The unavoidable gaps stem from the difficulty of representing intersection curves precisely in analytical form: for example, the degree of the intersection curve of two bicubic surfaces can be up to 324 [SAG84] and its algebraic genus can be as high as 433 [KS88]. For this reason, in practice the intersection curves can only be represented approximately.

The possible gaps between adjacent trimmed-NURBS patches can be a serious problem in the domain of solid modelling, especially to the interoperability of different CAD/CAM systems, and manual efforts are sometimes necessary to detect and close gaps. As mentioned in the introduction, this process is very expensive, and was reported to cost the US automotive industry over $600 million per year in lost productivity [BM99].

Various solutions that aim to address the gap problem have been proposed, but a general method that is capable of solving this problem adequately has failed to emerge. Subdivision surfaces, as will be introduced in the next section, overcome the topological limitations that exist with the trimmed-NURBS representation.

3.4 Subdivision surfaces

3.4.1 Introduction

Subdivision surfaces [AS10, CC78, Loo87, DS78, BK04, Sch96] have found more and more applications in arbitrary-topology surface modelling, computer animation and engineering design. Subdivision surfaces are a promising alternative method for geometric modelling, relative to the classical trimmed-NURBS representation.

The main advantage of subdivision surfaces is that a smooth surface can be obtained from an arbitrary coarse initial mesh, through recursive refinement. Subdivision surfaces are conceptually simple, filling a gap between polyhedral surface modelling and spline surface modelling. Subdivision surfaces are also closely related to them in that the control meshes are polyhedra defined by polygonal patches and that the refinement schemes employed in the subdivision procedures are analogous to those for spline surfaces and curves. There are a lot of refinement schemes available in the literature, of which the Catmull-Clark [CC78], Loop [Loo87], and Doo-Sabin [DS78] schemes are the basic ones.

In the following, the classification of various subdivision schemes is discussed. An introduction is given on two classic subdivision schemes—the Catmull-Clark scheme and the Loop scheme. Also, an important class—piecewise smooth subdivision surfaces [HDD$^+$94, BLZ00], which is useful for modelling sharp features and enables us to model smooth surfaces with boundaries—is presented.

3.4.2 Classification

Various subdivision schemes can be classified according to different criteria. First, these schemes can be classified into two categories: interpolating and approximating. With interpolating schemes, the positions of initial vertices and generated vertices remain throughout the process of subdivision; with approximating schemes, these positions are adjusted during the process of subdivision. Approximating schemes generally have greater smoothness, and converge to the limit surface more quickly. Popular subdivision schemes, such as Loop, Catmull-Clark, and Doo-Sabin

schemes, are approximating schemes.

Next, according to the type of meshes that are used for subdivision, we can classify these schemes into one category associated with triangular meshes and another with quadrilateral meshes. For example, the Loop scheme is generally used with triangular meshes, while the Catmull-Clark, Doo-Sabin, and Kobbelt [Kob96] schemes are used with quadrilateral meshes. However, this is not an absolute criterion: the Loop scheme can be applied to polygonal meshes when a triangulation step is performed in pretreatment; the Catmull-Clark scheme can also be used with triangular meshes (it converts a triangular mesh into a quadrilateral mesh in its first round of subdivision).

Finally, subdivision schemes can be classified by different types of spline functions that these schemes generate [AS10]. See Figure 3.5 [AS10, p. 33] for this classification of subdivision schemes.

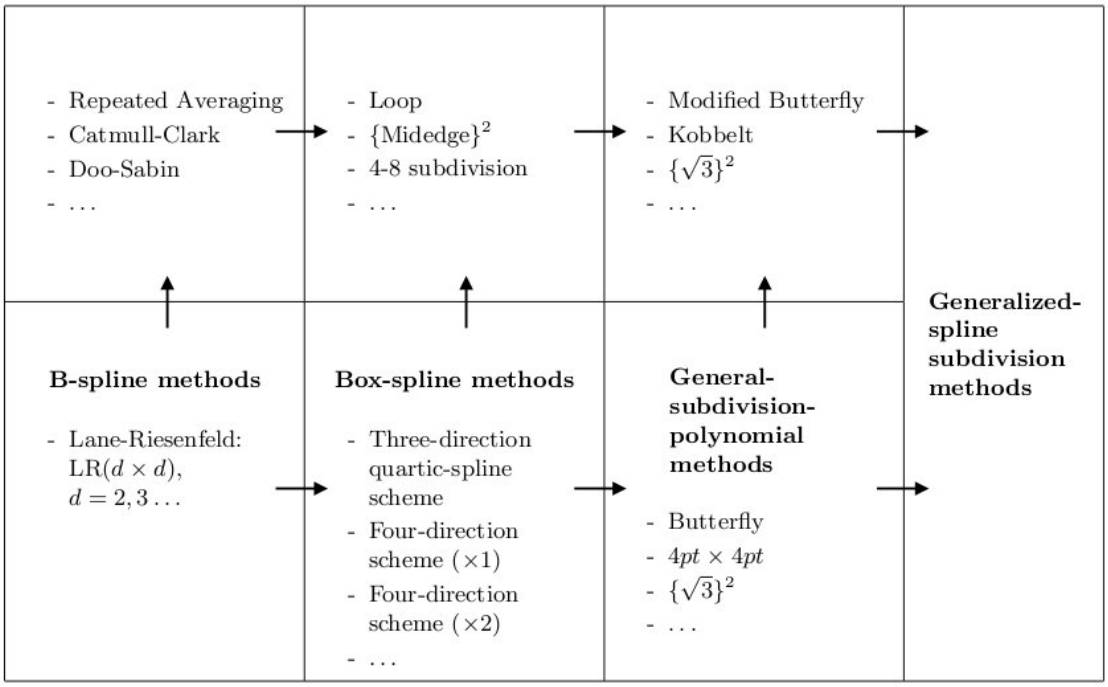

Figure 3.5: A classification of subdivision schemes.

Other criteria can be used to categorize subdivision schemes. For example, the

type of generated meshes, the smoothness of the limit surface, whether the method is dual or primal [AS10, p. 38], etc.

3.4.3 The Catmull-Clark scheme

The Catmull-Clark subdivision scheme was proposed in [CC78], as a generalization of the tensor product of bicubic B-splines. For meshes with arbitrary topology, it generates continuous C^2 limit surfaces everywhere except at extraordinary vertices (valence $n \neq 4$), where they are C^1 continuous.

The stencils of the Catmull-Clark scheme are described in Figure 3.6 for the interior region and Figure 3.7 for the boundary. These stencils define how the value at a vertex should be computed, given values at neighbouring vertices. The coefficients are set as [CC78]:

$$\alpha_n = \frac{n-3}{n}, \beta_n = \frac{2}{n^2}, \gamma_n = \frac{1}{n^2}.$$

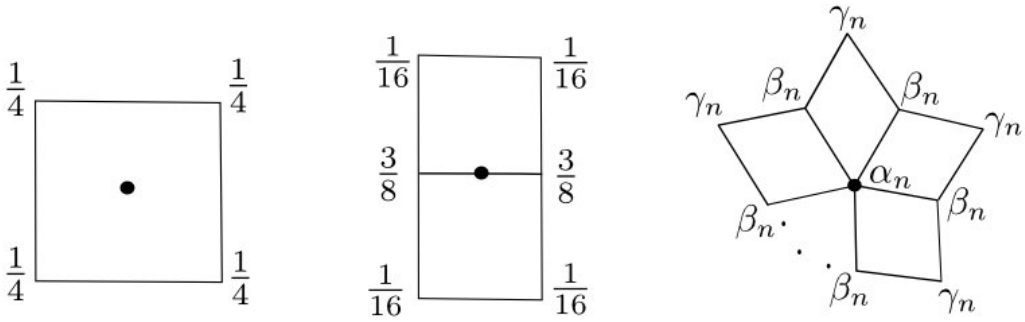

Figure 3.6: Catmull-Clark stencils for the interior region (left: stencil for a face vertex; middle: stencil for an edge vertex; right: stencil for an existing vertex).

The limit surface of Catmull-Clark subdivision surfaces can be evaluated directly, i.e., without recursion, using the method of [Sta98b].

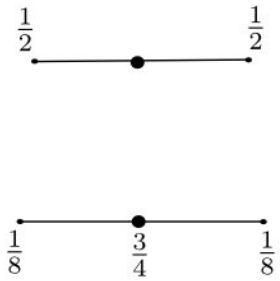

Figure 3.7: Catmull-Clark stencils for boundary vertices (top and bottom: stencils for odd and even vertices, resp.).

3.4.4 The Loop scheme

The Loop scheme [Loo87] is based on a quartic box-spline [dBHR93] of six-direction vectors that generates C^2 continuous limit surfaces everywhere except at extraordinary vertices (valence $n \neq 6$), where they are C^1 continuous. This scheme is applicable for triangular meshes, but can be also applied to polygonal meshes after the polygons are triangulated into triangles.

The stencils of the Loop scheme are described in Figure 3.8 for the interior region and Figure 3.9 for the boundary. The coefficient c_n is set as [Loo87]:

$$c_n = \frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2\right).$$

In [HDD+94], an extension to the classical Loop scheme was proposed and special rules are included to support sharp features (creases, corners, and darts). This will be discussed in Section 3.4.5.

3.4.5 Piecewise smooth subdivision surfaces

The idea of piecewise smooth subdivision surfaces stems from the need to model more general types of surfaces, other than closed surfaces generated by most subdivision schemes, i.e., surfaces that have boundaries and surfaces with features (creases, corners, or darts [AS10, Ch. 7]) in interior regions.

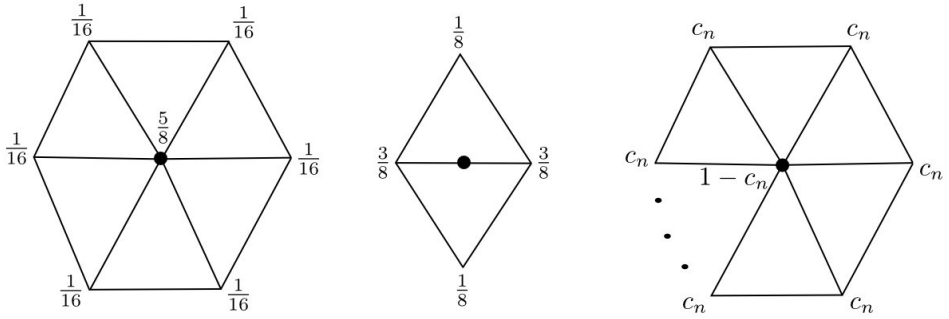

Figure 3.8: Loop stencils for the interior region (left and right: stencils for regular and extraordinary existing vertices, resp.; middle: stencil for an edge vertex).

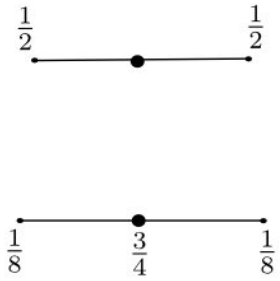

Figure 3.9: Loop stencils for the boundary vertices (top and bottom: stencils for odd and even vertices, resp.).

The piecewise smooth surfaces, as proposed by Hoppe et al. [HDD+94], are a generalization of the Loop scheme [Loo87], aiming to model sharp features as well as boundary curves, by locally modifying the original Loop subdivision rules. The work of [BLZ00] further improves the subdivision rules of the Loop scheme

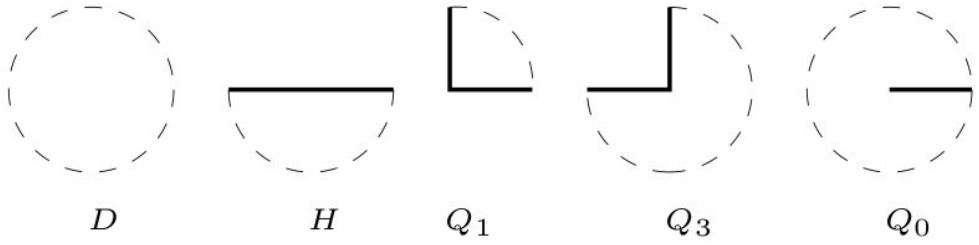

Figure 3.10: Five charts for piecewise smooth surfaces.

and Catmull-Clark scheme, allowing modelling of concave corners, and generating surface patches with prescribed normals. They also modify the boundary rules of Hoppe et al. [HDD+94] such that the shape of the generated boundary does not depend on the interior control points. See Figure 3.10 [BLZ00, Figure 1] for the five local charts for piecewise smooth surfaces: an open planar disk D denotes the interior region, a half-disk H with closed boundary means the boundary region of a piecewise smooth surface, and the additional three charts Q_1, Q_3, and Q_0 represent convex, concave and dart corners, respectively.

3.5 SGC cellular representation

The SGC (Selective Geometric Complex) [RO89] provides a common framework for representing non-regular sets of various dimensions, possibly having internal structures and cracks. SGCs are composed of cells of different dimensions that are mutually disjoint, open connected sub-manifolds. Thus, each vertex, edge and face of a 3D solid is a cell. We associate with each cell an extendible set of *attributes*, with which we are able to select and mark sets of cells that respond

to certain criteria, for example a set of cells can be selected to represent a result of a regularized Boolean union of two other sets.

Manipulation of SGCs can be achieved through a sequence of three primitive processes: subdivision, selection, and simplification. For two SGCs that are subject to an operation, say a regularized Boolean intersection, we first subdivide the cells of these two SGCs, making them compatible with each other, then select cells by setting the attribute "active" to *true*, and finally simplify the obtained SGC after the selection process by removing or merging certain cells (the resulting SGC represents the same point set but contains no unnecessary cells).

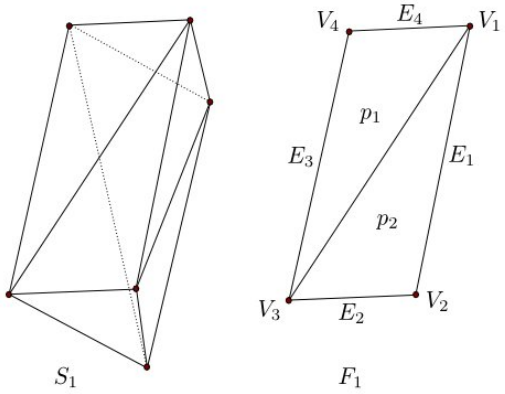

Figure 3.11: The wedge S_1 from Example 2.2.2, page 13, described in SGCs.

One possible avenue is that robustness analysis of the problem of Boolean operations will be performed in a geometric system that takes free-form solids as operands of Boolean operations. These solids are based on primitives that are three-dimensional r-sets with boundaries defined by SGC cellular representations, i.e., we have access to the 0-, 1-, and 2-dimensional cells. The 2-dimensional cells are patches defined by a union of Loop triangles enclosed by crease edges. The 1-dimensional cells correspond to crease edges. The 0-dimensional cells correspond to corners. For example, the wedge S_1 as in Figure 2.2 is represented here, in Figure 3.11, with SGCs. More specifically, the solid S_1, which is a 3-cell, is decomposed into five 2-cells, one of which is denoted as F_1 as illustrated on the right of

the figure. F_1 is defined by two Loop triangles, p_1 and p_2; with the SGC representation, it is also composed of four 0-cells, denoted by V_i, $i = 1, ..., 4$, and four 1-cells, that is E_i, $i = 1, ..., 4$.

The supported operations in this system would be the operations of a CSG representation: regularized Boolean operations and affine transformations.

CHAPTER 4

BOOLEAN OPERATIONS ON SUBDIVISION SURFACES

Boolean operations (union, intersection, difference, etc.) between solids are a fundamental set of operations in solid modelling, and are supported in most contemporary solid modellers. The set of regularized Boolean operations imposes the topological interior operation and the topological closure on the resulting objects obtained from the corresponding classic Boolean operations, ensuring the validity of the computed solids.

Up until today Boolean operations on solids using standard trimmed-NURBS are still haunted by possible irregularities, such as gaps and overlaps, among patches [Far99, KBF05]. On the other hand, subdivision-surface methods have found more and more uses in graphics industry due to their conceptual simplicity and their efficiency in smooth surface construction [BK04]. Previous work on Boolean operations and their robustness on subdivision surfaces includes Biermann-Zorin [BKZ01], Lai-Cheng [LC06, LC07], and Smith-Dodgson [SD07].

The remainder of the chapter is organized as follows. In Section 4.1, we will give two methods for the evaluation of Loop subdivision surfaces. We will then discuss the implementation of Boolean operations on subdivision surfaces in Section 4.2. Our approach will be similar to the surface-based approach of Biermann and Zorin [BKZ01].

4.1 Evaluation of Loop surfaces

In our system that reformulates the problem of Boolean operations on solids, which will be described in detail in Section 5.5, we need to test whether two solids that are involved in a Boolean intersection are close enough to incur a topological ambiguity, such as the one described in Example 2.2.2. It is for this reason that we describe here the two methods for Loop surface evaluation: Stam's method can be used

in evaluating precisely Loop subdivision surfaces, and the Wu-Peters method may be employed to evaluate Loop subdivision surfaces in an approximate way, and to determine the space relations of two solids that are near in space with much efficiency, which fits our needs very well.

4.1.1 Stam's method

Direct evaluation, i.e. non-recursive evaluation, of Loop surfaces can be achieved using Stam's method [Sta98a]. With it, first we translate each box-spline patch into a triangular Bézier patch with a different set of control points, and we can then evaluate or display graphically the generated Bézier triangular patches with no difficulties. Note also that we subdivide the Loop surface once so that there exists at most one extraordinary vertex for each Loop patch.

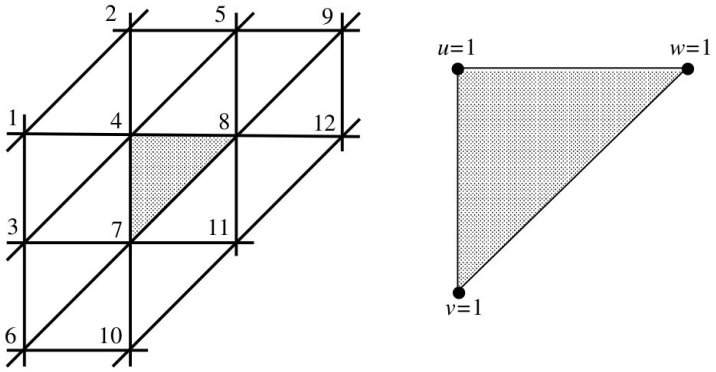

Figure 4.1: A regular Loop patch defined by its 12 control points.

Since the Loop scheme [Loo87] is based on a quartic box-spline of six direction vectors (see Section 3.4.4), a regular box-spline triangular patch (each of its vertices has valence $n = 6$), defined by 12 control points, as shown in Figure 4.1 [Loo87, p. 2], is itself a triangular Bézier patch. Each patch can be expressed with the Bézier representation:

$$p(u, v, w) = \sum_{i=0}^{14} p_i b_i(u, v, w)$$

where $b_i(u, v, w), i = 0, ..., 14$ form an array of 15 Bernstein functions:

$$\{u^4, 4u^3v, 4u^3w, 6u^2v^2, 12u^2vw, 6u^2w^2, 4uv^3,$$
$$12uv^2w, 12uvw^2, 4uw^3, v^4, 4v^3, 6v^2w^2, 4vw^2, w^4\}$$

and $p_i, i = 0, ..., 14$ are the 15 control points for the patch. These control points can be obtained by multiplying a conversion matrix $Q_{15\times12}$ (given in [Sch96, p. 21]) and the original 12 control points of a Loop patch. This fact is used in particular to find the eigenstructure of the subdivision matrix corresponding to the Loop method.

For a Loop patch that has an extraordinary vertex of valence $n \neq 6$, the method of Stam [Sta98a] can be used to compute the eigenstructure of the subdivision matrix. Once the eigenstructure is known, a triangular patch is defined in terms of a projection of initial control vertices and a vector of eigenbasis functions.

4.1.2 The Wu-Peters method

The Wu-Peters method proposed in [WP04] is suitable for approximate evaluation of Loop subdivision surfaces, and especially efficient in interference detections between Loop surfaces where exact evaluation of Loop patches is not necessary. When it is employed for interference detections, approximate bounds, which are much easier to establish than the complex eigenstructure method of [Sta98a], are computed to support the tests. The Wu-Peters method generally uses the interval arithmetic of [Kob98] to construct bounds for Loop patches, but does not follow the normal-direction bounding of [Kob98]; it establishes the bounding volumes, instead, using interval triangles.

For interference tests, we need to bound the limit patches. To bound the limit patches appropriately, however, is a matter of a consideration of accuracy and speed for the application. In the Wu-Peters method [WP04], we need to precompute bounds on the basic functions for the Loop scheme, then for each Loop patch, we bound its x, y, and z components of the corresponding limit surface. This is followed by the construction of an offset triangle that encloses the Loop patch,

reducing the intersect tests between free-form solids represented by Loop surfaces to triangle-triangle tests, which can be accomplished by adopting Guigue and Devillers' method [GD03], possibly accelerated by using bounding volume methods, such as the OBB (Oriented Bounding Box) method [GLM96].

Bounds for a Loop patch

For regular Loop patches, all vertices have valence value $n = 6$. For irregular patches, extraordinary vertices have different valence values $n \neq 6$. We assume that every extraordinary vertex is surrounded by regular vertices. This can be guaranteed by once uniformly subdividing the initial Loop surface before we compute the component bounds of the mesh. This has been a common method to simplify the computation and analysis of subdivision surfaces. In deriving the basic function bounds and computing component bounds, the neighbourhood of a Loop patch, which is composed of $n + 6$ vertices, is labeled in such a way (illustrated in Figure 4.2) that the vertex numbered 0 is the extraordinary vertex (if there is any) that needs special treatment.

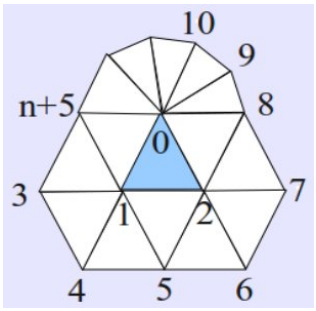

Figure 4.2: A Loop patch with valence n and its neighbourhood.

The bounds for Loop basis functions are provided in [WP04], in which lists of bounding values for mesh vertices with different valences are derived and given for direct use. A Loop limit patch $F(u, v)$ can be described as a linear combination of control points weighted by a set of basis functions in the u-v parametric domain

[WP04]

$$F(u,v) = \sum_{i=0}^{n+5} c_i b_i(u,v) \tag{4.1}$$

where $b_i(u,v)$, $i = 0, ..., n+5$ are the basis functions, and c_i are the control points of the patch. Extracting a linear function $l(u,v)$ that interpolates the three vertices c_0, c_1, and c_2, which define the corresponding mesh patch, gives

$$F(u,v) = l(u,v) + \sum_{i=3}^{n+5} d_i b_i(u,v) \tag{4.2}$$

where d_i denotes the difference between c_i and $l(u_i, v_i)$ for $i = 3, ..., n+5$, and $d_i = 0$, for $i = 0, 1, 2$. As mentioned before, the bounds for b_i are directly provided in [WP04], so we can bound each component of the limit mesh patch $F(u,v)$ within the intervals

$$\begin{array}{ll} \left[l(u,v) + \sum_{i=3}^{n+5} d_i b_i^-(u,v),\ l(u,v) + \sum_{i=3}^{n+5} d_i b_i^+(u,v)\right] & \text{if } d_i > 0 \\ \left[l(u,v) + \sum_{i=3}^{n+5} d_i b_i^+(u,v),\ l(u,v) + \sum_{i=3}^{n+5} d_i b_i^-(u,v)\right] & \text{otherwise.} \end{array}$$

By considering the x, y, and z components in the above inequalities, we can have component-based bounds. Now the bounds for the Loop patch have been established.

Interference tests

Interference tests between Loop surfaces can be performed by creating an offset triangle that bounds each Loop patch. For convenience in interference tests, we construct a base triangle for the offset triangle, which is defined as

$$\left(\frac{x_i^- + x_i^+}{2}, \frac{y_i^- + y_i^+}{2}, \frac{z_i^- + z_i^+}{2}\right), i = 0, 1, 2$$

and then an offset triangle is obtained by defining a sphere with radius ρ, whose center moves over the base triangle mentioned above. This radius is defined as

$$\rho = \max\{r_1, r_2, r_3\}$$

where r_i, $i = 0, 1, 2$ is defined by

$$r_i = \left[\left(\frac{x_i^+ - x_i^-}{2} \right)^2 + \left(\frac{y_i^+ - y_i^-}{2} \right)^2 + \left(\frac{z_i^+ - z_i^-}{2} \right)^2 \right]^{1/2}.$$

By this point, the interference test between two patches is reduced to a test of two corresponding offset triangles, with the error tolerance set to

$$\epsilon = \rho_1 + \rho_2,$$

where ρ_1 and ρ_2 are the radii of the spheres corresponding to the respective Loop patches. This triangle-triangle test can be solved using Guigue and Devillers' method [GD03] with the above mentioned maximum acceptable error.

Note that interference tests between two solids represented by Loop subdivision surfaces cannot be absolutely precisely performed due to the nature of the free-form modelling. Although Stam's method can be used to precisely evaluate a Loop surface, it does not provide us bounds of Loop patches for efficient interference tests. On the other hand, the Wu-Peters method establishes tight bounds for Loop patches, facilitating our interference tests, which are not to be performed exactly though: this method will not omit possible intersections, but when we conclude that there are intersections using this method, it may turn out that there are actually no intersections. This false-positive prediction, however, will not pose a problem for us, since we only intend to employ the Wu-Peters method to determine whether two solids involved in a Boolean intersection are close enough to incur topological ambiguities, for example, the ambiguities illustrated in Example 2.2.2, in the result solid.

4.2 Boolean operations on subdivision surfaces

As already mentioned, performing Boolean operations in solid modelling is a basic problem that has attracted much attention [RS97, AD03, BKZ01, LC07]. While Boolean operations on solids represented by polygonal meshes (for example, triangular or quadrilateral meshes) or quadric surfaces (spheres, cylinders, cones, etc.) are generally supported in solid modelling systems, they are not reliable on free-form solids.

As mentioned in the last chapter, trimmed-NURBS patches have intrinsic difficulty in keeping patches joined exactly with no gaps or overlaps, while for subdivision surfaces, trimming curves can be guaranteed with exact interpolation, for example, through the use of combined subdivision [Lev99a, Lev99b]. (The details of subdivision surface trimming can be found in [LLS01], and combined subdivision surfaces are not used in this thesis.) Therefore, the application of subdivision surfaces may be more reliable and less error-prone in geometric computations and we think that a reliable method of Boolean operations with subdivision surfaces is promising in practice.

In the following, two approaches to implement Boolean operations on solids represented by subdivision surfaces are presented: the surface-based approach of [BKZ01] and the voxel-based approach of [LC07]. In our work, we will use the surface-based approach to compute Boolean operations of subdivision surfaces. We generally follow the method of [BKZ01] and do not intend to make major improvement to it, because the focus of our work lies on the reformulation of the problem of the Boolean operations, with associated guarantees, as discussed in the next chapter.

4.2.1 The surface-based approach

In [BKZ01], a surface-based algorithm generates the control mesh of the approximate result surface, followed by parameter optimization and fitting of this approximate surface. The intersection curve of two subdivision surfaces is computed using

a refined version of the control meshes, and a perturbation scheme [Sei94] is applied to determine whether a point is above or below the plane of a triangle. Note that the method proposed in [BKZ01] ensures the topological *well-formedness* of the result, however, it fails to ensure that its topology is always correct relative to the true solution and that the result is geometrically accurate.

Given two solid objects defined by meshes M_1 and M_2, respectively, we want to obtain the object represented by subdivision surface M approximating the true result, i.e., the regularized Boolean intersection of M_1 and M_2. We are especially interested in the robustness problem of the intersection-curve computation and in the verification of well-formedness of the final resulting surface.

Intersection curve

The objective of the intersection-curve computation is to find an approximate curve to the true intersection of two subdivision surfaces. The reason for an approximation is that the exact intersection of subdivision surfaces is very complex and generally not necessary: the resulting intersecting curve usually has a very high degree and may have handles [LC07]. Therefore it is generally difficult to represent the intersecting curve exactly, and an approximation to this curve is a common approach. For a summary of intersection problems in solid modelling, refer to [FHK02, Ch. 25].

This computation involves the bounding-box preprocessing and the triangle-triangle intersection test. The use of bounding-box preprocessing excludes unnecessary computation of triangle-triangle intersections, and the triangle-triangle test relies on a point-plane test to determine whether an edge intersects a triangle; this, in turn, is based on a test that determines whether a point falls above, below, or on the plane of a triangle. The latter can be realized by the above-predicate [For89] that determines the position relation between a point q_0 and the plane through q_1,

q_2, and q_3, which is expressed as followed:

$$D(q_0, q_1, q_2, q_3) = -\begin{vmatrix} q_{0x} & q_{1x} & q_{2x} & q_{3x} \\ q_{0y} & q_{1y} & q_{2y} & q_{3y} \\ q_{0z} & q_{1z} & q_{2z} & q_{3z} \\ 1 & 1 & 1 & 1 \end{vmatrix} \tag{4.3}$$

The perturbation method [EM90, EC95, Yap90, Sei94] is often used to resolve degenerate cases. The main advantage of the perturbation over a direct analysis of degenerate cases is that it provides a straightforward approach to treat degenerate cases non-trivially, i.e. as a limit of perturbed cases. In [Sei94], however, it is noted that in some geometric examples, the perturbation method may be more costly than a direct treatment.

A perturbation ε is applied to the four points in (4.3) so that each of these points can be expressed as a linear function $q_i(\varepsilon) = q_i + \varepsilon r_i$, where r_i is a random direction for $i = 0, 1, 2, 3$. The sign of $D(q_0, q_1, q_2, q_3)$ can be determined instead by:

$$\lim_{\varepsilon \to 0^+} \text{sign } D(q_0(\varepsilon), q_1(\varepsilon), q_2(\varepsilon), q_3(\varepsilon)). \tag{4.4}$$

The sign can be computed without much difficulty, if it can not be determined with the current perturbation, we choose another perturbation and recompute the sign.

The triangle-triangle intersection based on the above perturbed predicate can be implemented using Guigue and Devillers' method [GD03], which proves more efficient than the algorithms of Möller [Möl97] and Held [Hel98]. It is based exclusively on orientation predicates and needs fewer intermediate computations. The idea of this algorithm can be illustrated by Figure 4.3 [GD03, p. 6]: suppose we have two triangles T_1 and T_2 with vertices p_1, q_1, r_1, and p_2, q_2, r_2, respectively. π_1 and π_2 denote the respective planes on which the triangles T_1 and T_2 lie, and they intersect at a line L. Let i, j, k, and l denote the intersection of L with four edges p_1r_1, p_1q_1, p_2q_2, and p_2r_2, respectively. The algorithm can then be summarized as follows:

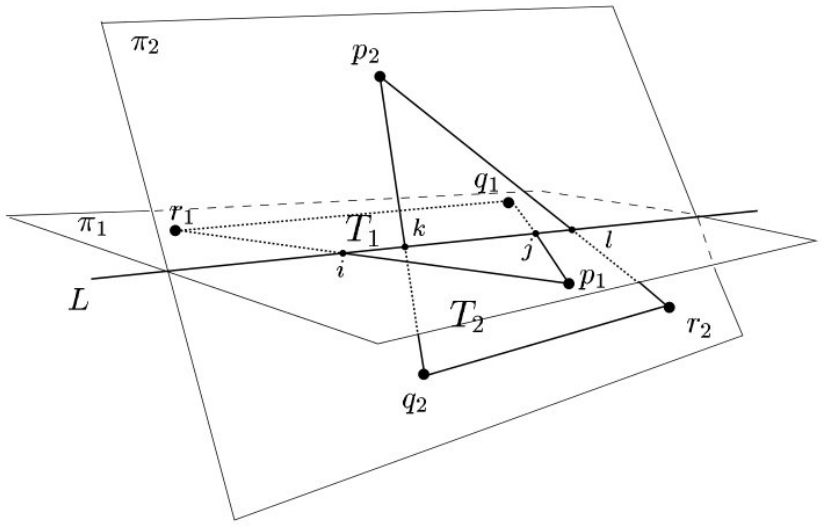

Figure 4.3: Triangle-triangle intersection.

Step One: Each vertex of one triangle is given an orientation test with regard to the plane of the other triangle. For each triangle, the classification has three possibilities: all predicates are non-zero and has the same sign, in which case "no intersection" is reported; all predicates are zero, which reduces to the three-dimensional triangle-triangle intersection; the predicates have different signs (only in this case we need to impose the orientation tests on the other triangle).

Step Two: When the triangles pass the above tests, the two triangles are permuted so that they satisfy the vertex ordering of Figure 4.3: the vertex p_1 (resp. p_2) is the only vertex that lies on the positive side of π_2 (resp. π_1) in general cases, or the condition is relaxed to the non-negative side when two vertices of the triangle lie on one side and the other vertex is on the plane.

Step Three: A conclusion can be reached: for the canonical situation as depicted in Figure 4.3, the condition for the intersection of these two triangles is $(k \leq j) \cap (i \leq l)$, which reduces to two predicates: $D(p_1, q_1, p_2, q_2) \leq 0$ and $D(p_1, r_1, r_2, p_2) \leq 0$.

Error estimation

The surface-based method computes intersection between limit meshes as an approximation of the true intersection between two free-form solids, incurring potential errors. We must try to bound the possible error resulting from this approximation. A variant of Guigue and Devillers' method using the CGAL implementation for evaluating predicates is used for this thesis.

4.2.2 The voxel-based approach

This voxel-based approach is presented by Lai and Cheng [LC06, LC07] to perform error-controllable Boolean operations on free-form solids represented by Catmull-Clark subdivision surfaces. Our proposed work will not use this voxel-based approach. We summarize this approach, however, as an alternative method that may be worth further investigation for the implementation of Boolean operations on subdivision-surface models.

This voxel-based approach works in two steps of voxelization: first, two objects described with the subdivision surfaces are voxelized based on a midpoint subdivision of the parameter space, see Figure 4.4 [LC07, p. 491]. This *global voxelization* is performed on the boundaries recursively until all subpatches are small enough to be voxelized by their four corners, and then propagated to the interior regions by a flooding operation. The second voxelization is called the *local voxelization* that is only performed on intersecting regions (*I-subpatches* in [LC07]), which are usually very small, to improve the computation accuracy of the intersection curves.

Given an error tolerance ε, the condition, such that the distance between each surface patch of the resulting solid and the corresponding quadrilateral that approximates it, is not greater than this tolerance, can be derived with no difficulty.

A major benefit of the voxelization in this approach is that no degeneracies need to be treated specially, since these are difficult to handle in methods that use polyhedral approximation to the surfaces when computing the intersecting curves.

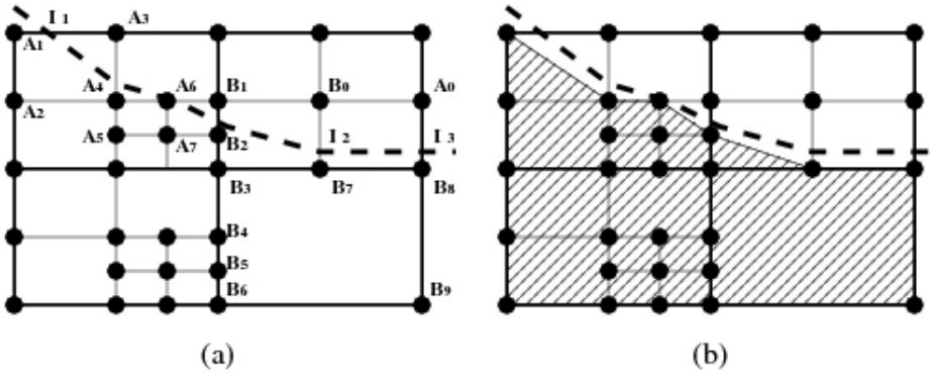

Figure 4.4: Intersection based on 2D parameter space subdivision.

CHAPTER 5

THE REFORMULATION OF BOOLEAN OPERATIONS

As we showed in Chapter 2, the problem of Boolean operations of geometric models is ill-conditioned when it is subject to uncertainty in the geometric data and no extra information is provided about the topology. For an ill-conditioned problem, a small error in the input can result in a large error in the solution. A reformulation of the problem is then necessary to detect and resolve existing uncertainty.

Our proposed method for this reformulation can be summarized as the following: given two solids that are described as their bounding Loop surfaces and that are operands of a regularized Boolean operation, for example an intersection, we represent these two solids with the SGC (Selective Geometric Complex) scheme and impose binding of SGC cells of Boolean operations to resolve uncertainty, when some uncertainty of the operation is detected; we also propose to eliminate self-intersection of the result after binding, using the theorems from [APS98], such that the result is a well-formed Bézier complex. Evaluation and estimation of Loop surfaces, and the two approaches to the implementation of Boolean operations were already discussed in the last chapter.

This chapter first emphasizes the necessity of the reformulation, and then elaborates the proposed SGC-binding-based external intervention when ambiguities are detected. Elimination of self-intersection that is based on [APS98] permits us to ensure well-formedness of the computed solid. We then describe our theoretical goal: theorems showing that such external intervention is *sufficient and necessary* to eliminate the detected ambiguities.

5.1 The necessity of reformulation

[1] The central difficulty associated with standard Boolean-operation methods is that they are attempting to solve a problem that is intrinsically ill-conditioned [ASZ07]. This means that small perturbations of the input arguments may lead to an incorrect result, and furthermore, the perturbations due to the user's inability to accurately specify the input objects are larger than the small perturbations just mentioned. It follows that, in the general case, any attempt to compute Boolean operations without supplementary information must inevitably fail, since the numerical method cannot possibly take account of the perturbations due to inaccurate specification.

The principal consequence of these remarks is that we must reformulate the problem of finding Boolean operations in order to incorporate supplementary information in the ill-conditioned case. We emphasize again that this reformulation is forced upon us: in the ill-conditioned case, it will be *impossible* to compute reliably the correct regularized operation using finite-precision floating-point arithmetic [IEE08]. Furthermore, even if finite-precision arithmetic were replaced by exact arithmetic, in the ill-conditioned case the computed answer will almost certainly be unacceptably wrong, because the user cannot properly specify the problem to be solved: it is ill-posed, and the inaccuracy in the specification will be sufficient to render the solution unacceptable. In short, overcoming the difficulties in computing Boolean operations is not a matter of designing better algorithms, nor of doing better numerical analysis. It is, rather, a matter of obtaining enough information to ensure that the correct solution is well-defined by the data.

We note here that this situation is not unusual when dealing with numerical methods. We have already mentioned the example of polynomial approximation [FM67] in the introduction, and more generally, similar considerations led to a revolution in the approach to the solution of linear equations and the eigenvalue problem [Wil65].

[1] This section is taken almost verbatim from a draft document [Ste11] by N. F. Stewart.

5.2 Binding of cells

We specify the object as composed of primitives that are three-dimensional r-sets with boundaries defined by SGC cellular representations. An SGC model is composed of mutually disjoint cells, which can be 0-, 1-, and 2-dimensional; this representation can handle all types of manifold and nonmanifold objects.

The purpose of binding is to permit external removal of ill condition, for example, ill condition as described in Example 2.2.2. Briefly, binding means that when cells are close, they will be merged. The idea of binding is, to some extent, inspired by the decomposition in [RO89]: it corresponds to one of the three primitive low-level operations (see Section 3.5 for details) that are the basis for defining and implementing high-level topological, Boolean, and structural operations. Both our approach and that of [RO89] try to make two operands "compatible" with each other. In our work, however, making operands "compatible" is not simply done by splitting topological cells, but rather achieved by refining Loop triangles. The goal of our binding is not to build a basis for implementation of higher-level operations, for example a Boolean intersection, but rather to enable elimination of possible real-life ambiguities of Boolean operations.

Implementation of the bindings

Suppose we have two solids A and B described by Loop subdivision surfaces that are concerned in a Boolean operation (union, intersection, difference, etc.). The binding operations between them can be further specified by means of six cases (we suppose that the bindings, below, are all from A to B):

VV:

A vertex of one operand to another vertex of another operand.

VE:

A vertex of one operand to an edge of another operand. We need to first refine triangles that are adjacent to this edge before the binding.

VF:

A vertex of one operand to a face of another operand. We need to refine all triangles in B that contain this vertex before the binding.

EE:

An edge of one operand to another edge of another operand. In this case, we must perform two VE bindings before we fit the edges together.

EF:

An edge of one operand to a face of another operand. We require two VF bindings, and one EE binding.

FF:

A face of one operand to a face of another operand. We require three EF, and three EE bindings.

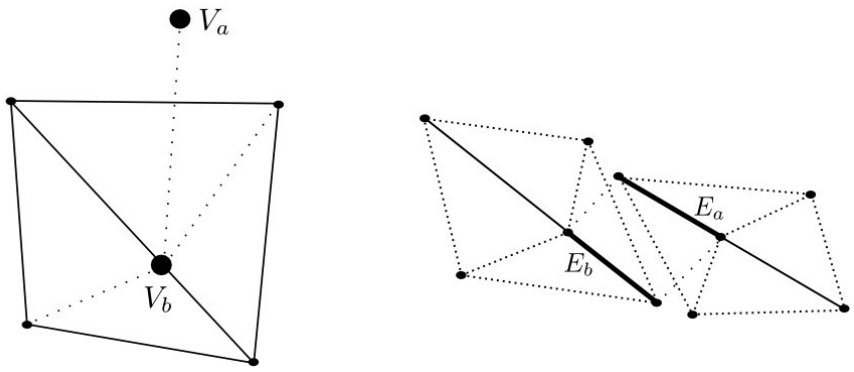

Figure 5.1: Left: a VE binding. Right: an EE binding is composed of two VE bindings followed by a refinement.

In Figure 5.1, a vertex V_a in A is to be bound to some edge in B, which requires that a vertex V_b on this edge and near V_a is determined, and that triangles adjacent to this edge are refined. This figure shows also that an EE binding is implemented based on two VE bindings and a refinement that makes two new edges (E_a and

E_b) coincide geometrically. More complex bindings can be implemented using simpler bindings. But how bindings in these cases work, especially what strategy of refinement should be used and to what extent an automatic binding without external intervention is possible, remains under investitation. While automatic model "healers" may reduce manual work, some artifacts (for example, computers cannot decide whether a crack between two cubes results from the user's intention or from erroneous computations) are literally impossible to detect and eliminate without external intervention, and the focus of our work is, therefore, to allow external intervention to eliminate artifacts using bindings, but *only when necessary.*

5.3 Verification of Bézier complexes

The theorems from [APS98] can be used to ensure the well-formedness of Loop patches after we compute the result of Boolean operations, possibly subject to bindings when bindings are necessary to resolve ambiguities.

In [APS98], conditions are given that exclude self-intersections of composite Bézier curves and patches. These conditions can be categorized into two groups: necessary and sufficient conditions, which give sharp criteria, and sufficient conditions, which are much easier to compute in practice but give less-sharp criteria. Perturbation analyses are also provided to obtain maximum perturbations that can be tolerated to the control points of composite curves or patches that have been reported with no self-intersections.

5.4 Resolution of ambiguities

In this section we illustrate how topological ambiguities of the result can be resolved using bindings. Ambiguities, when detected, must be eliminated before we are able to obtain the correct topological form of the result solid. Examples of such external removal are manual by the user, or automatic using a "snapping" method; feature modelling systems may also ensure that no ambiguities exist by their persistent naming mechanisms.

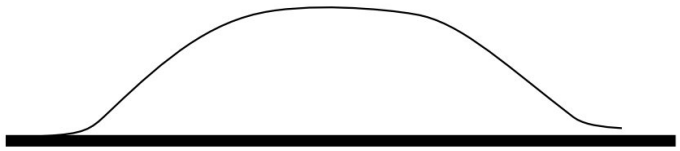

Figure 5.2: Boolean union of a spring clip and a block.

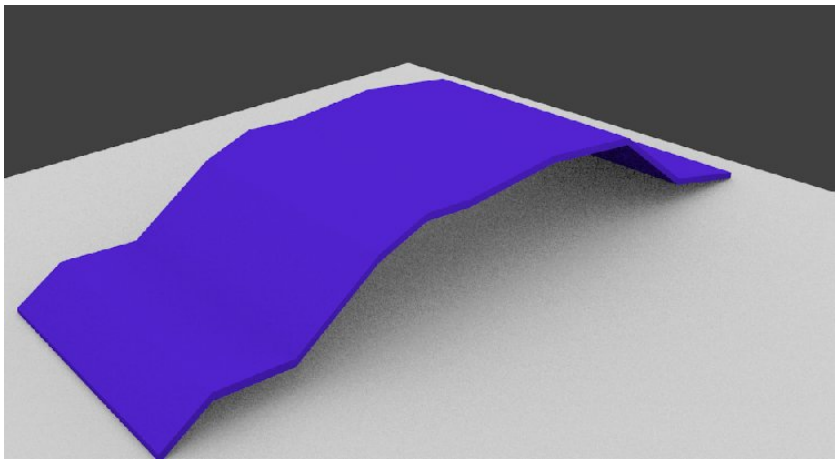

Figure 5.3: Automatic removal of ambiguities using “snapping”.

In CAD systems, "snapping" is usually implemented as a tool that enables convenient "bindings" of elements, such as vertices, edges, and faces, which proves very useful when modifying an object or aligning elements of an object to those of another object. The example of computing the Boolean union of a spring clip and a block (shown in cross-section) from [ASZ07] (Figure 5.2), is modelled and rendered in three dimensions with Blender 2.59 [Ble11], as shown in Figure 5.3. Ambiguities exist because the system cannot apprehend the correct "design intent" concerning the right end of the spring clip. But if "snapping" is enabled (which shows that the user intent is to join some vertices, edges, or faces), the two ends of the spring clip can be guaranteed to touch the block surface: we move the clip to make sure that one end, say, the right end, of the clip, touches the block's top surface, then make a rotation of the clip around its right end such that the left end touches the surface too. (Note that in Blender, only "snappings" from vertex to vertex/edge/face are supported, while other types of "snappings" such as from edge to edge, edge to face are not supported yet.) In this way, the problem of set-union in this case can be reformulated to be well-conditioned.

The tool "snapping" in most contemporary CAD systems is, however, limited to simple cases in which the objects are represented by polygonal meshes. Automatic "snapping" for free-form meshes has its own difficulties: first, it is much more costly in computation to guarantee that a vertex is bound with an edge or a surface in free-form modelling; second, more sophisticated cases would require edges or surfaces to be refined in "snapping", in a similar way as described in Section 5.2. In any case, we emphasize that the goal of the thesis is not to invent better or more general snapping algorithms. Rather, the goal is to specify the conditions that must be satisfied by any algorithm in order to guarantee correctness.

For another example we try to resolve the ambiguities in computing the Boolean union of two solids S_0 and S_1 in [ASZ07], whose cross-sections are illustrated in Figure 5.4. We rotate the cube S_1 by 45 degrees and translate it to a position such that one of its faces touches perfectly the face of S_0 with outer normal $\mathbf{n}$, after which we then compute the Boolean union of the two solids. This inevitably leads

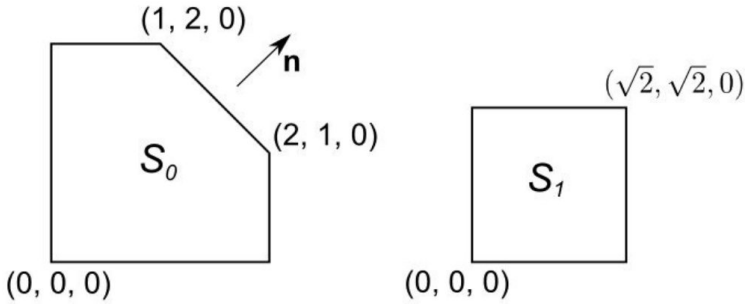

Figure 5.4: Two solids S_0 and S_1.

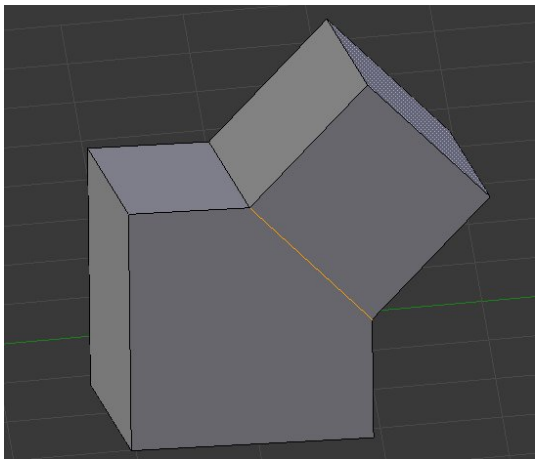

Figure 5.5: Manual resolution of ambiguities to meet the user intent.

to a numerical problem and ambiguities in the "user intent" because the computed solid that represents the result of the Boolean union would almost certainly have gaps or overlaps.

If polygonal meshes are used in this example, the reason for this problem is that the data used for defining S_1 have uncertainty due to the rotation and translation of S_1 and the approximate value of $\sqrt{2}$. Automatic "snapping" can be used (which shows that the user intent is to join some vertices, edges, or faces), in this case, to guarantee that the two faces of S_0 and S_1 coincide, and that the final result has no numerical problems. But if free-form meshes are used instead, we will finally need to manually resolve the ambiguities, using bindings of SGC cells (details are given early in Section 5.2). More specifically, we need to perform an FF binding that makes sure two faces from S_0 and S_1 respectively coincide if this is the correct "user intent". Figure 5.5 illustrates in Blender a result from two solids represented in polygonal meshes. How to define the required refinement properly in Section 5.2, however, remains further work.

The third method that may be able to resolve ambiguities is the persistent naming mechanisms [AMP00, MP02, Mar05] that are used in feature-based parametric CAD systems, such as Cobalt, Autodesk Inventor, and Creo Parametric (formerly known as Pro/ENGINEER), to guarantee correct and consistent topological data. Such CAD systems enable reevaluation of models when parametric specifications that define the characteristics of the models are modified in later modelling and are also known as *procedural* or *history-based*. The persistent naming mechanisms can make reevaluation in various stages of modelling consistent without mismatch and make them able to capture different "design intents". The persistent naming mechanisms include rules to eliminate ambiguities such as described earlier in Example 2.2.2, page 13, so that we can obtain the correct result.

Problems with this mechanism, however, exist which may result in mismatching objects in reevaluation and failure to apprehend the "design intent" [MP02]. Of course, another limitation is that this approach works only with feature-based parametric CAD systems on which Boolean operations on free-form objects are

supported.

5.5 Outline of a general solution

Based on the discussion in the previous sections, we could envisage implemention of a system that resolves the possible topological ambiguities with Boolean operations, say the Boolean intersection. One possibility for the organization of the overall system might be along the lines of exception-handling systems of a programming language, such as C++ and Java. The general outline of the low-level intersection algorithm might be as follows:

(1) Decompose the two solids represented by Loop subdivision surfaces into SGC cells.

(2) Perform an interference test using the Wu-Peters method and determine the zones of possible ambiguities where bindings will be necessary.

(3) Bind corresponding cells of the two solids in the ambiguous zones, thus enabling the resolution of possible ambiguities.

(4) Compute the intersection of the two solids using the surface-based method (see Section 4.2.1, page 41) and the topological bindings mentioned above.

(5) Use the theorems of [APS98] to verify the Bézier complex converted from the resulting solid is well formed.

In our approach, bindings of cells would be performed externally, either by asking for intervention from the user, automatic resolution by "snapping", higher-level resolution of feature modelling systems, or some other method.

In fact, in Chapter 6, we restricted our attention to a special case, namely, the planar-cut problem. As for the long-term theoretical goal mentioned at the beginning of the chapter, we would like to show that the algorithm provides a solution with the correct topological form, and with small geometric error. To

do this, the algorithm may require higher-level external intervention (automatic snapping, user intervention, feature-modelling mechanisms, etc.). We aim to give theorem showing that, in addition to being sufficient, such intervention is necessary, in the sense that the algorithm will only request intervention for problems that are very close to an ill-conditioned problem, which cannot therefore be solved without such intervention.

The algorithm given above should therefore be viewed as a low-level intersection algorithm which requires intervention from a higher-level system.

CHAPTER 6

MANAGING UNCERTAINTY OF GEOMETRIC COMPUTATION WITH BACKWARD ANALYSIS

The robustness problem in geometric modelling has been studied over decades. In the context of computation using ordinary IEEE floating-point arithmetic, the problem has traditionally been treated as one of error control. However, taking into account the fundamental ideas of error analysis presented in earlier chapters, such as data uncertainty and the condition of the underlying problem, and using a criterion that requires correct topological form, the robustness problem becomes one of managing uncertainty.

Algorithms for Boolean operations on geometric objects defined by subdivision surfaces have been proposed by different authors in the literature. These works mainly aim to obtain results that are geometrically correct and that are valid in topology. Note that an algorithm may produce an object that has valid topological form (i.e., it is topologically well-formed), but the topological form may not be correct (i.e., it has a topological form different from that of the true solution). We focus our attention on ensuring that the computed objects have the correct topological form.

As described in previous chapters, when correct topological form of the result is of concern, and when ordinary IEEE floating-point arithmetic is used in the implementation, an interactive approach is forced upon us: it is necessary that a higher-level process provide (on request) information about the correct topological form.

This chapter describes an algorithm to implement this idea in the case of the planar-cut problem. The contents of this chapter, beginning in Section 6.1, are a slightly modified version of the published paper [SS13].

Note that the planar-cut problem, i.e. the planar cut of an object represented by a subdivision-surface mesh, instead of the more general problem of computing

Boolean operations of two solids defined by subdivision surfaces, is investigated for reasons of simplicity. This problem is simple to visualize yet far from trivial. This makes it possible to study the main idea of managing uncertainty in geometric computations, without having to worry about the complicated technical details that are found in the implementation of general Boolean operations.

6.1 Introduction

The robustness problem in geometric modelling has been studied over a period of decades. It raises many interesting theoretical questions [HS05, Yap08], and it is also of great practical importance, since the cost of unreliable computation in this application area is measured in billions of dollars [BM99, Far99].

In the context of computation using ordinary IEEE arithmetic [IEE08], the problem has traditionally been studied as one of error control. Unfortunately, most analyses carried out have neglected to include certain of the fundamental ideas of error analysis [DB07], such as uncertainty in the data, and the condition of the underlying problem. In fact, even a clear definition of a criterion to measure error is often lacking [HS05, ASZ07]. If the problem is reformulated to take these ideas into account, with an error criterion that requires correct topological form, then it immediately becomes clear that the problem may be ill-conditioned, and the question of practical concern becomes one of managing uncertainty. This uncertainty arises, for example, out of uncertainty in the input data, storage, and roundoff error due to the use of finite-precision floating-point arithmetic, and (for certain methods of representing geometric objects) the approximation of high-degree polynomials by polynomials of low degree. Given the practical question just mentioned, it is of interest to formulate appropriate algorithms for the management of the uncertainty in the input to geometric algorithms, and management of uncertainty that may be created by such algorithms.

In this chapter we describe an interactive approach for the robust implementation of geometric operations on objects defined by subdivision surfaces, in the case

when we require correct topological form of the result, when there is uncertainty in the data, and when the computations are done using ordinary IEEE floating-point arithmetic, with supplementary use of interval arithmetic [MKC09, CXS12, MPF12, P1712]. It is shown that in this context, an interactive approach is forced upon us: for the implementation to provide correct results, it is necessary that a higher-level process provide (on request) information about the topological form of the result. This is done by having the algorithm throw an exception, to be caught by a higher-level process, when supplementary topological information is required.

Since the boundaries of subdivision-surface objects can vary in an almost arbitrary manner, it may seem unduly optimistic, at first glance, to seek rigorously correct topological form. The reason we can hope to do this, of course, is that we permit the algorithm to trap and request supplementary information. We do not, however, want the algorithm to do this indiscriminately. The goal is then to find an algorithm that will request supplementary information only when it is demonstrably impossible to proceed in any other way.

We are ultimately interested in applying our approach to the general problem of computing Boolean operations on geometric objects defined by subdivision-surface meshes, but in order to study the main idea without having it obscured by complicated details, we consider a simpler problem. Specifically, we give an algorithm to compute the planar cut of an object defined by a subdivision-surface mesh—more precisely, the object defined by Loop subdivision [Loo87, AS10]. The planar-cut problem arises frequently in practice [Gra00], it is far from trivial, and yet it is simple to visualize (see Figure 6.1, which comes from [CGA12]). The algorithm uses an adaptive version of Loop subdivision, based on RG (Red-Green) triangulation [MS02, p. 180], and a multi-step rule proposed in [PP09, Sec. VI-A].

In order to avoid possible confusion, we distinguish between the often discussed problem of interactive editing of subdivision surfaces [ZSS97, AFR02, CLL07], and the problem discussed here. Interactive editing is a process for specifying surfaces, while the algorithms we discuss are applied to surfaces that have already been defined, except possibly for uncertainty in their defining data. The interactive

approach we propose refers to algorithms for Boolean operations, or for the solution of the planar-cut problem, and this interactivity has nothing to do with interactive surface editing.

Algorithms for Boolean operations on geometric objects defined by subdivision surfaces have been proposed previously in the literature, including for example the surface-based approach of [BKZ01], and the voxel-based approach of [LC07]. Another reference, for the general problem of surface intersections, is [Pat02]. Our algorithm can be viewed as a version of the one in [BKZ01], simplified because we restrict our attention to Boolean intersection, in the case when one of the two objects to be intersected is a half space. As noted in the next paragraph, the actual algorithm used is not the subject of the chapter. The main point is how to deal with the uncertainty in the input data, and with the requirement of correct topological form (which is in contrast to [BKZ01], where it is only required that the solution be topologically valid). For sameness of topological form we take the classical requirement that there should be a homeomorphism linking the true and computed solutions: see the discussion in [ADPS95].

One of the advantages of using a subdivision-surface representation, rather than the traditional trimmed-NURBS representation, is that many of the difficulties of dealing with inconsistent data [ASZ07] are avoided. This is true in particular for the analysis of error related to the approximation of high-degree polynomials by polynomials of low degree, mentioned above.

In Section 6.2 it is observed that many geometric problems, such as the problem of Boolean intersection, are problems for which the condition number is discontinuous (as function of the backwards error), but for which it is useful to continue the solution process through the discontinuity. Then, in Section 6.3, it is observed that the cutting-plane problem (which is closely related to a very special case of Boolean intersection) is also such a problem, and we give an algorithm to solve it. The geometric methods underlying the presented algorithm are not the subject of the chapter: in fact our algorithm relies almost exclusively on ideas previously presented in the literature [Loo87, MS02, PP09, Wu05, WP04, Hoh91, PT95]. The

main contribution of the chapter is in Section 6.4, where we discuss the idea of an exception mechanism to deal with the possible ambiguity of the topological form of the result. In particular, we show that the interactive approach is forced upon us. If the algorithm throws an exception, and asks for help in determining the topological form, it is usually because a small perturbation of the input data could lead to a change in the topological form of the solution, and our goal is to find algorithms which make such a request for supplementary topological information *only* when it is necessary. Section 6.5 concludes the chapter, with some remarks on our prototype implementation, and some remarks on future work.

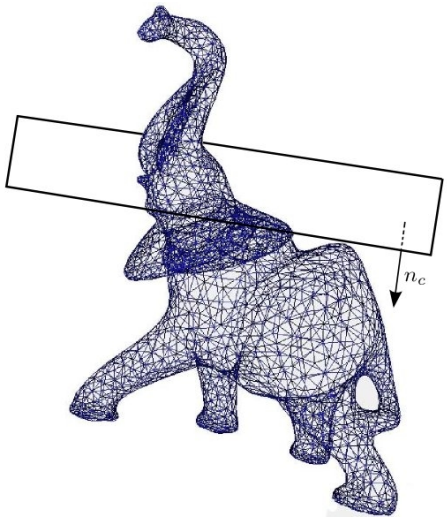

Figure 6.1: Intersection of a plane and a locally-planar mesh.

6.2 Uncertainty and condition numbers

In the context of Boolean operations on geometric objects, in the presence of uncertainty, and with a requirement of correct topological form, a mechanism for interaction with a higher-level process is unavoidable.

Consider for example the problem of specifying a bevel in a feature-based solid-modelling system [SR04]. The semantics of such specifications rely on the use of Boolean operations: for example, in Figure 6.2, the bevel is specified as the

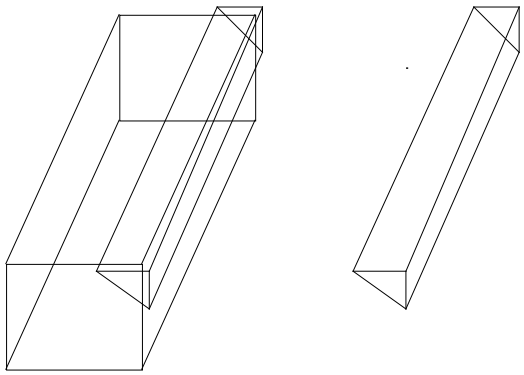

Figure 6.2: Bevel specified by a Boolean difference.

set difference between the block and the narrow wedge-shaped object. Both of these objects are defined with uncertainty, possibly resulting from previous operations that used finite-precision arithmetic (say, a rigid motion), or from some other sources. Calculation of the set difference can easily lead to an object with topological form different from that of the correct result, and the correct topological form must therefore be imposed by a higher-level process (in the present example, by a "persistent-naming" mechanism, which notifies the algorithm of overlaps in the input objects [SR04]).

Other examples can easily be given [ASZ07]. For example, if a flexible springclip is to be joined to an object at one end of the clip, but left separated from the object at its opposite end, and if again there is uncertainty in the defining data, then it will be impossible using finite-precision arithmetic to resolve the topological form without supplementary information.

Since there may be other sources of uncertainty in the data defining the problem, besides those related to preliminary use of floating-point arithmetic, the level of data uncertainty will often be much larger than floating-point roundoff error. An example of another source of uncertainty is the interactive input of objects on a screen. This example also illustrates another possible "higher-level process", namely a procedure for the interactive entry of objects. In this case the correct topological form would be obtained from the human user.

These ideas can be placed in the context of a standard error analysis [DB07] by viewing the error as infinite if the topological form is incorrect. This means that the computational problem is ill-conditioned for inputs for which small changes in the input data can lead to a change in topological form, and the condition number therefore has an infinite discontinuity near data points corresponding to these inputs. On the other hand, we wish to continue the computational process through the discontinuity, i.e., we do not want to abandon the computation in the case of such ill-condition. External intervention to resolve the ambiguity is consequently necessary. Our criterion for a good algorithm will be that external intervention will be requested only when it is necessary, i.e., only when small perturbations of the uncertain input data could lead to a change in topological form. We seek to define algorithms that meet this criterion as often as possible.

6.3 Planar cut of a locally-planar mesh

A (triangular) *logical mesh* M is a finite collection of faces, each defined by a set (not necessarily ordered) of three vertices $\{\ell_0, \ell_1, \ell_2\}$, along with their associated edge sets $\{\{\ell_0, \ell_1\}, \{\ell_1, \ell_2\}, \{\ell_2, \ell_0\}\}$ [AS10, p. 10]. The logical mesh is *locally planar* if each edge belongs to at most two faces and if, for any vertex ℓ, the j faces ϕ_i incident at ℓ ($i = 0, \ldots, j-1$) can be ordered in such a way that ϕ_i meets ϕ_{i+1} at an edge containing ℓ for $i = 0, \ldots, j-2$. The locally-planar mesh is in addition a *mesh without boundary* if, for each vertex ℓ, the corresponding ordered faces satisfy the condition that ϕ_{j-1} meets ϕ_0 along an edge. Note that "locally planar" does not mean geometric planarity: no geometric information is specified in a logical mesh. The valence of vertex ℓ, by definition equal to the number of edges incident at ℓ, is denoted by n.

If we now proceed to associate with each vertex ℓ ($\ell = 0, \ldots, L-1$) a control

point $p_\ell \in \mathbb{R}^3$, and collect the p_ℓ (written as 1×3 row vectors) in an $L \times 3$ matrix

$$p = \begin{bmatrix} p_0 \\ \vdots \\ p_{L-1} \end{bmatrix}_{(L\times 3)}, \tag{6.1}$$

then $\mathcal{M} = (M, p)$ is called a *polyhedral mesh* [AS10, p. 15]). This is the geometric mesh in $\mathbb{R}^3$. An example of a connected, triangular, locally-planar polyhedral mesh without boundary is illustrated in wire-frame format in Figure 6.1.

In this chapter we restrict our attention to (finite) locally planar meshes without boundary. The hypothesis of local planarity is implicit in most discussions of subdivision surfaces, although it is usually not explicitly specified. More general subdivision schemes involving meshes that are not necessarily locally planar have been studied, for example in [YZ01].

Under weak conditions, application of the Loop subdivision method [Loo87, AS10], to a locally-planar triangular mesh without boundary, converges uniformly to a smooth limit surface $S = S(u, v)$, which can be parametrized locally by two variables u and v. If the control point p_ℓ corresponding to each vertex of each triangle in the mesh is associated with its limit position, the triangles imply a decomposition of S into triangular curvilinear patches with the limit positions at the patches' corners [AS10, p. 111]. Further, if the vertices of the triangle are regular ($n = 6$), then the patch can be expressed as a polynomial over the triangle (in fact, it is a quartic Bézier surface [Sch96, Lai92]). Even at extraordinary (non-regular) vertices, the surface is C^1.

Let the normal vector $n_c \in \mathbb{R}^3$ and the scalar σ define the cutting plane $P_c = \{x \in \mathbb{R}^3 : n_c \cdot x = \sigma\}$. The *contouring problem* [Gra00] can be defined as computing the intersection between the plane P_c and the surface S. In fact, however, we will study a slightly more elaborate problem, namely, the *planar-cut problem*. For simplicity, we assume that the initial mesh is connected, so that the corresponding limit surface is a connected set, and we assume that the surface does not selfinter-

sect. (Non-selfintersection of single patches, and non-intersection of both adjacent and non-adjacent triangular surface patches, can actually be checked computationally using the methods of [APS98, GS01].) Let $Int(S)$ be the finite region in $\mathbb{R}^3$ defined by S. The planar-cut problem involves the computation of the boundary B of the intersection between $Int(S)$ and the half space $H_c = \{x \in \mathbb{R}^3 : n_c \cdot x \geq \sigma\}$. These are sets in $\mathbb{R}^3$: they are illustrated in Figure 6.3. In Figure 6.1, which shows an approximation to a surface, if the normal n_c defining the cutting plane points downward as shown, then the trunk and the top of the head of the elephant are cut off, and the resulting opening is covered by a planar surface.

As already mentioned, Loop subdivision takes a triangular mesh and produces a sequence of refined triangular polyhedral meshes that converge uniformly to the surface S [AS10]. Note that the limit surface is completely determined by the control points in the given mesh, and the fact that Loop subdivision is used. The adaptive version of Loop subdivision described in Section 6.4.1, based on Red-Green (RG) triangulation, changes the approximating meshes that are calculated as the process proceeds, but it does not change the limit surface.

The planar-cut problem is defined by the initial mesh $\mathcal{M}^0$, the vector n_c, and the scalar σ; an admissible approximate solution to the problem is a well-formed (conformal) triangular mesh which is *topologically the same as* the boundary B of $Int(S) \cap H_c$. Since small perturbations of the geometric input data (the control points p_ℓ of the initial mesh $\mathcal{M}^0$) may cause the topological form of B to change,

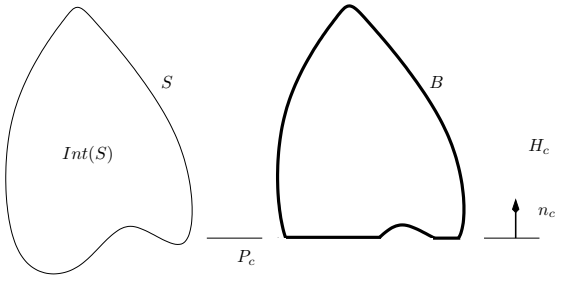

Figure 6.3: Left: cross-sections of S and $Int(S)$. Right: cross-section of boundary B (thick line).

the problem is ill-conditioned. For example, in Figure 6.4, if all of the control points p_ℓ undergo an identical upward translation, as indicated by the vertical arrow in the figure, the topological form of B will change. This follows because the surface will be translated by an amount equal to the translation in the control points, due to the affine invariance of the subdivision process [AS10, p. 39].

An approximate and high-level description of our algorithm is given now (details appear in the next section of the chapter). The initial mesh $\mathcal{M}^0$ is assumed given. In a first phase, triangles in $\mathcal{M}^0$ near the plane P_c are refined using RG triangulation, until a desired error criterion, based on distance between triangle vertices and the plane P_c, in the direction n_c, is satisfied. Nearness to P_c is determined by using the Wu-Peters bounding box [Wu05]. An important side effect of this first phase is the creation of a list C (the notation is intended to suggest "close") of triangles which, according to the Wu-Peters bounding-box, might possibly have an associated triangular surface patch that intersects P_c. Although Red-Green triangulation is not described until Section 6.4.1, we remark here that green triangles are treated as pairs, with the Wu-Peters test applied to the red mother triangle. The red-green mesh obtained at the end of this initial phase is denoted $\mathcal{M}^{\nu_0}$.

A triangle is said to generate a simple intersection if the intersection between its triangular surface patch and P_c consists of a single curve cutting across the patch. (The curve must not pass through a corner of the triangular surface patch.) In the second phase, the algorithm processes the triangles in C, subdividing $\mathcal{M}^{\nu_0}$ if

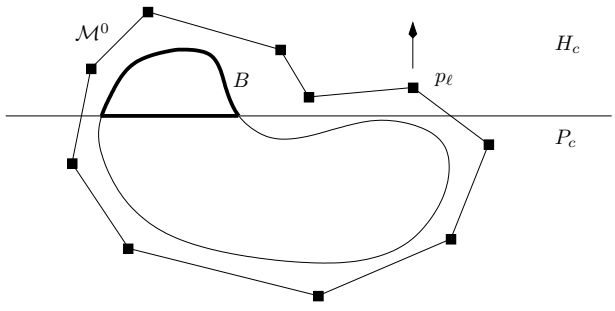

Figure 6.4: Cross-section of B: risk of change of topological form.

necessary, replacing triangles that do not generate a simple intersection.

If subdivision occurs, the set C is updated. Subdivision may result in smaller triangles for which the associated triangular surface patch cannot intersect P_c. These triangles are deleted from C. Triangles generating a simple intersection are labelled to indicate the pair of edges joined by a curve which forms the intersection between the triangular surface patch and P_c. There are three cases, as illustrated in Figure 6.5. In the figure we have deliberately shown a triangle, with labels specifying intersections of edges, that does not quite intersect P_c. We do this to emphasize that it is the intersections associated with the associated triangular surface patch that are relevant, and not the intersections with the triangle itself. The Loop subdivision method is an approximating method, and not an interpolating method [AS10]; consequently the limit of a triangle corner could be on the opposite side of P_c.

In a third phase, the intersection path, between the mesh and P_c, is traced, using the elements of C. (The reason for separating the second and third phases is that the process of elimination of non-simple intersections can in principle lead to subdivision of neighbouring triangles. If the second and third phases were done together, for each triangle, the intersection curve of the neighbouring triangle might already have been traced, and this information would have to be transferred to the children of the triangle. In the context of red-green subdivision, this is not a straightforward process.)

The fourth and final phase of the algorithm is to merge the modified mesh

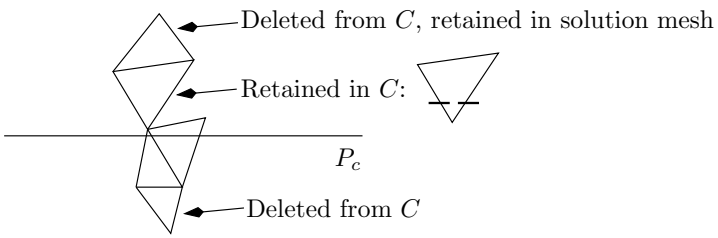

Figure 6.5: Labels on triangles in C.

with a conformal mesh approximating the planar set $Int(S) \cap P_c$. This conformal planar mesh can be computed using any appropriate triangulation. We used the constrained Delaunay triangulation [She00] provided by CGAL [CGA12]. The merged mesh is an admissible approximate solution in the sense defined above.

Some further remarks should be made here, concerning the second phase of the algorithm. For the planar-cut problem, to specify the topological form of the result it is necessary and sufficient to specify the topological form of $S \cap P_c$. The topological form of $\mathcal{M}^{\nu_0} \cap P_c$, however, may not be the same as that of $S \cap P_c$. Consequently, to obtain an admissible approximate solution, it may be necessary to modify $\mathcal{M}^{\nu_0}$, by subdivision, in order to find an equivalent mesh $\mathcal{M}^{\nu}$ (one with the same limit surface S) for which the topological form of $\mathcal{M}^{\nu} \cap P_c$ is the same as that of $S \cap P_c$. This task is essentially accomplished in the second phase, by decomposing the intersecting part of the mesh into triangles with simple intersections.

It may happen, however, that even after repeated subdivision, such a decomposition cannot be found, so that it is not possible to determine the topological form of $S \cap P_c$. If the subdivision reaches a triangle size that is on the order of the data uncertainty, the algorithm traps and asks to be informed of the topological form of $S \cap P_c$. Thus, if the algorithm traps, there exists a perturbation of S, approximately equal to the size of the data uncertainty, which could change the topological form. Because of the subdivision, however, this does not necessarily mean that a perturbation of the *original input data* could lead to a change in topological form. Restating our previously stated goal, we would like to establish weak conditions guaranteeing that the algorithm will trap and ask for additional information only in the case when a perturbation of the original data, approximately equal in size to the data uncertainty, could lead to a change in topological form. This question is discussed further in Section 6.5. The case when the curve passes through a corner of the patch is dealt with by perturbation, as described by the pseudocode below.

6.4 The uncertainty-management process

In this section we present the techniques used by our algorithm to take account of the inherent uncertainty, as outlined at the end of the previous section. We begin in Section 6.4.1 by describing the basic subdivision process used. We then present certain fundamental techniques used in the algorithm, including the Wu-Peters bounding box, a specialization of the Hohmeyer test for detection of loop intersections, and the Variation Diminishing Property for Bézier curves. Finally, the algorithm to trace the graph defining the topology of $S \cap P_c$ is described in Section 6.4.6.

6.4.1 Red-Green Loop triangulation

Our algorithm uses the method of RG triangulation [MS02]. Our implementation of this method uses a multi-step rule, suggested in [PP09, Sec. VI-A], to determine appropriate geometric values for the vertices $p_\ell \in \mathbb{R}^3$, and their successors $p_\ell(\lambda)$, where the non-negative integer λ specifies the level of subdivision ($p_\ell(0) \equiv p_\ell$). For $\lambda = 0$ we have $\ell = 0, \ldots, L-1$, while for positive λ the index ℓ runs over a larger range. Note that here, "successor" refers to successive values of the vertex in the subdivision process.

(The ambiguity in the notation p_ℓ, due to the fact that the indexing depends on the level, should lead to no confusion. The vertices are not stored in indexed form: rather, they are part of a standard half-edge data structure for meshes. Thus, the notation p_ℓ simply refers to some specific vertex in the mesh.)

Each subdivision of a triangle effected by ordinary Loop subdivision uses the primal $pT4$ schema [AS10, p. 17], in which a new logical vertex is introduced in each edge, and the three new vertices are joined by new edges. Thus, each triangle is decomposed into four new triangles. A consequence of this fact is that if the mesh is subdivided uniformly, the amount of data to be stored increases proportionally to 4^λ. Since this function increases very quickly with λ, various methods of adaptive subdivision have been proposed for practical use. Thus, the

RG triangulation method used in our algorithm permits triangles in one part of the mesh to be subdivided while triangles in another part are not.

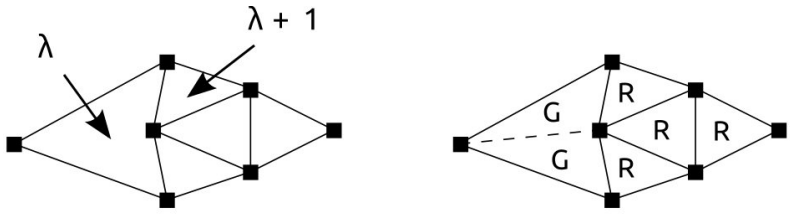

Figure 6.6: Mesh made conformal by green subdivision.

The main difficulty with adaptive subdivision is that if a triangle is subdivided, but its neighbour is not, then a non-conformal mesh results, as illustrated in Figure 6.6 (left), where one triangle has been subdivided to level λ, and the other to level $\lambda + 1$. To correct this, a new (green) edge is introduced, and the triangles formed by this new edge are labelled green, in contrast to those obtained by the $pT4$ subdivision scheme, which are labelled red. Introduction of a new edge in this way is referred to as *green refinement*, which is in contrast to the standard subdivision, referred to as *red refinement*. See Figure 6.6 (right).

Given certain triangles marked for refinement, a conformal mesh can be maintained using the following algorithm [MS02]:

(1) Eliminate all green refinements by restoring the mothers of green triangles. If a green triangle was marked for refinement, the restored mother is also marked for refinement.

(2) Refine all red triangles marked for refinement using the $pT4$ schema.

(3) While there exist triangles with more than one non-conforming vertex, refine them using the $pT4$ schema.

(4) Apply green refinement to all triangles having exactly one non-conforming vertex.

The algorithm is implemented recursively. Green and red faces are tagged as green or red in the half-edge data structure representing the mesh. In our context the application of green refinement, in Step (4), and the elimination of green refinement in Step (1) of the next subdivision, can be eliminated. One green refinement is necessary at the very end of the process to make the result conformal.

Proposition 1. [MS02, p. 181]. *The RG triangulation algorithm terminates with a conforming triangulation.*

There remains one other difficulty, however, in the context of Loop subdivision. Any particular vertex ℓ of valence n may have adjacent triangles that have been subdivided at (up to $\lceil \frac{n+1}{2} \rceil$) different levels, as illustrated by the vertex labelled $p_\ell(\lambda_0)$ in Figure 6.7.

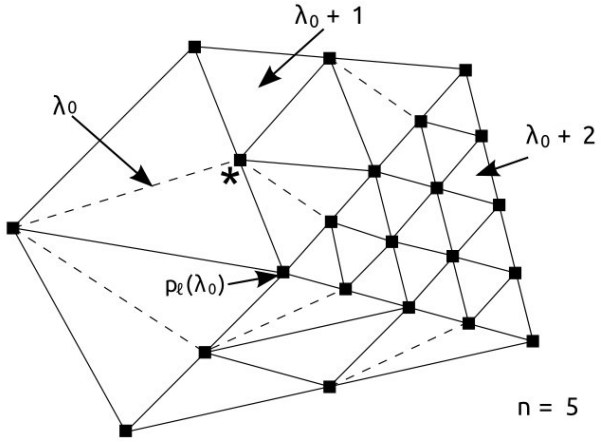

Figure 6.7: Adjacent triangles subdivided at different levels.

This means that the geometric value $p_\ell \in \mathbb{R}^3$ is ill-defined. We resolve the ambiguity by storing the value of p_ℓ at the level λ_0 at which the vertex was inserted; this value is denoted by $p_\ell(\lambda_0)$. For a vertex inserted as a result of subdivision at level λ that does not keep the mesh conformal (as in the case of the middle vertex in Figure 6.6, left), this means that we store $p_\ell(\lambda+1)$, just as we would in the case of conformal insertion of a vertex. We also store $p_\ell(\infty)$, the limiting value of the

vertex on the surface S, which can be calculated [PP09] by

$$p_\ell(\infty) = \left(1 - \frac{8\alpha_n}{3+8\alpha_n}\right) p_\ell(\lambda_0) + \frac{8\alpha_n}{n(3+8\alpha_n)} \sum_{i=1}^{n} p_\ell^i(\lambda_0).$$

Here, $p_\ell^i(\lambda_0)$, $i = 1, \ldots, n$, denotes the geometric value at one of the n one-neighbours of $p_\ell(\lambda_0)$ at subdivision level λ_0, and

$$\alpha_n = \frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\left[\frac{2\pi}{n}\right]\right)^2.$$

(A one-neighbour of a vertex is another vertex in the mesh, joined at level λ_0 to the original by a single edge.) Unavailable one-neighbours (for example, the neighbours of the vertex indicated by a "*" in Figure 6.7, at level $\lambda_0 + 1$) are computed on the fly, and not retained in the data structure.

We note in passing that in the example of Figure 6.7 we must have $\lambda_0 = 0$, since a vertex cannot be inserted with valence $n = 5$.

Proposition 2. [PP09, Sec. VI-A]. *Storage of the two values $p_\ell(\lambda_0)$ and $p_\ell(\infty)$ permits subsequent calculation of p_ℓ at any level $\lambda_0 + \lambda$ by means of the multistep rule*

$$p_\ell(\lambda_0 + \lambda) = \gamma_n(\lambda) p_\ell(\lambda_0) + (1 - \gamma_n(\lambda)) p_\ell(\infty),$$

where

$$\gamma_n(\lambda) = \left(\frac{5}{8} - \alpha_n\right)^\lambda.$$

The values of $p_\ell(\lambda_0)$, $p_\ell^i(\lambda_0)$, $p_\ell(\infty)$, and $p_\ell(\lambda_0+\lambda)$ are computed using ordinary floating-point arithmetic. The $p_\ell(\lambda_0)$ are computed (using Loop subdivision at Steps (2) and (3) of the RG triangulation algorithm) as convex combinations of values at the previous level of subdivision, and the error in their calculation can be bounded tightly using the standard model for floating-point arithmetic [DB07, Ch. 2]. Similarly, it is straightforward [DB07, p. 107] to take account of the other errors involved in the calculation of $p_\ell(\lambda_0 + \lambda)$ (the details are omitted). Thus, making the mesh available to the higher-level process, we can replace the $p_\ell(0) \equiv p_\ell$

by the subdivided mesh in the problem definition, and affirm that the method has solved a problem corresponding to a surface that has suffered perturbations, which are assumed to be smaller than those corresponding to the data uncertainty. Our point of view is that of the backward error analysis [DB07, p. 113]: the algorithm will give a correct result for a problem which has been perturbed by less than the data uncertainty, provided only that possible changes in topological form are detected on the fly.

We note here that alternate weights are often used in the Loop subdivision formulas (see [AS10, Note 18, p. 328] and[CGA12]). The statement of Proposition 2 can easily be modified to correspond to this alternate choice of weights.

Figure 6.8 shows the results of an RG triangulation using the Loop method with

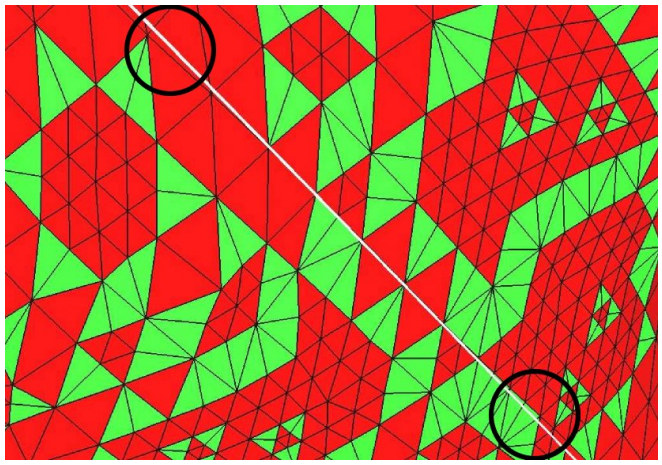

Figure 6.8: Result of Red-Green Loop triangulation.

the multistep rule of Proposition 2. The mesh is viewed at a very slight angle to the cutting plane (shown in white), which traverses the figure diagonally from top-left to bottom-right. The cutting of triangles near the top and bottom of the figure as indicated by the circles seems unambiguous, but recall, that it is the cutting of the associated triangular surface patches that is of primary concern. Dealing with this distinction is the main task of the second phase of the algorithm, described in the next subsection. Examples illustrating the distinction are also given there.

6.4.2 Triangles generating simple intersections

The initial phase of the subdivision, described in the previous section, is unremarkable. A Wu-Peters bounding box [Wu05] provides a criterion to decide which triangles intersecting P_c should be subdivided first, and to estimate the error in the approximation. The Wu-Peters bound is obtained by extracting a linear function from each component of the patch, such as the x-component, and finding a bound of the form

$$\begin{aligned} x^+ &= l(u,v) + \textstyle\sum_{i=3}^{n+5} \max\{d_i, 0\} b_i^+ + \min\{d_i, 0\} b_i^- \\ x^- &= l(u,v) + \textstyle\sum_{i=3}^{n+5} \max\{d_i, 0\} b_i^- + \min\{d_i, 0\} b_i^+ \end{aligned} \tag{6.2}$$

where $l(u,v)$ is the linear function. (In order to define d_i, b_i^+, b_i^-, and $l = l(u,v)$, the parametric domain is defined locally to be a portion of the plane [Wu05, WP04]. The surface is then defined in terms of auxiliary basis functions having a linear precision property, and b_i^+ and b_i^- are precomputed linear upper and lower bounds for these basis functions. The function $l(u,v)$ is a linear function approximating the patch, and the d_i can be interpreted as the errors in this approximation corresponding to control points neighbouring the patch. A complete description is given in [AS10, pp. 280-283].) Subdivision is carried to a deeper level in regions of the mesh near P_c, until an externally supplied error criterion is satisfied. This produces the mesh $\mathcal{M}^{\nu_0}$ and the list C mentioned in Section 6.3. The bounds in (6.2) are reminiscent of formulas for interval arithmetic, and interval arithmetic is in fact used to compute C.

Figure 6.9 shows an example, which uses a base mesh from [CGA12]. The triangles in C, which according to the Wu-Peters test might intersect the plane P_c, are shown in red and green. In this figure, the mesh is again viewed at a very slight angle to the cutting plane.

The algorithm's next task, in the second phase, is to identify triangles, in $\mathcal{M}^{\nu_0}$ or in some subdivided mesh $\mathcal{M}^{\nu}$ with a possibly slightly perturbed limit surface S, in such a way that the topological form of $\mathcal{M}^{\nu} \cap P_c$ is exactly the same as that of

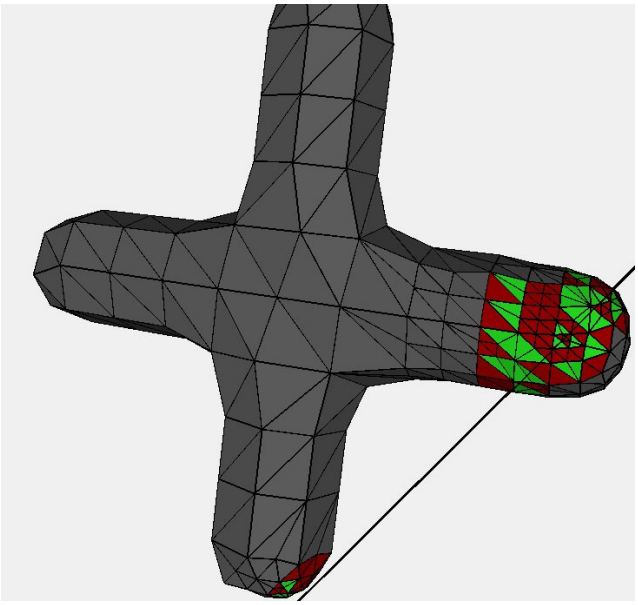

Figure 6.9: An example of C (triangles shown in red and green).

$S \cap P_c$, and to replace them by triangles having only simple intersections. This is not a straightforward problem. For example, a triangle that does not intersect P_c may nonetheless have an associated triangular surface patch which cuts the plane in a curve traversing the patch, in an internal loop within the patch, or both. The last-mentioned case is illustrated in perspective and in cross-section, respectively, in Figure 6.10 (left and right). In the left figure, the line intersection is shown by a solid curve, and the loop intersection by a small dotted circle. Similarly, a triangle might intersect P_c in a straight line, while the associated triangular surface

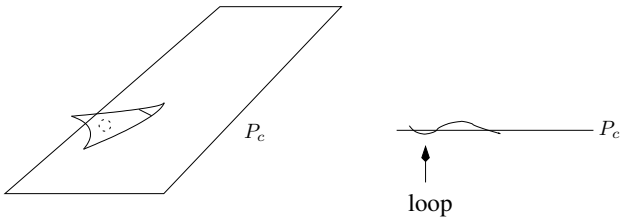

Figure 6.10: Possibility of a loop intersection with P_c.

patch intersects P_c in both a curve and a loop, only in a curve, only in a loop, or not at all. Small changes in the data may change the situation from one case to another, and we wish to correctly determine which case holds using finite-precision and interval arithmetic.

As already suggested, the principal approach used to resolve these ambiguities is to produce an equivalent mesh $\mathcal{M}^\nu$ by appropriate further subdivision of $\mathcal{M}^{\nu_0}$. It is possible, however, that even after further subdivision it is not possible to determine the topological form of $S \cap P_c$. In this case an exception is thrown, in order to request the required information. As mentioned at the end of Section 6.3, the algorithm should not throw an exception if there is no ambiguity in the topological form of the result. Consider, for example, the situation illustrated in Figure 6.11, where S intersects P_c at a sharp angle. If the surface does not vary quickly, relative to the level of uncertainty, then there is no ambiguity in the topological form, even though some of the triangle elements (vertices and edges) in $\mathcal{M}^\nu$ might be very close to the plane, and therefore the algorithm should not trap. Furthermore, we would like to avoid further subdivision whenever possible, to reduce the computational cost of the algorithm. Consequently, the algorithm does not merely subdivide repeatedly in the areas of possible intersection with P_c until some very small triangle size is attained. Instead, it uses the tests described in subsequent subsections, which detect the topological nature of the intersections between the triangular surface patch and P_c. The tests may succeed, even for fairly large triangles, so that the

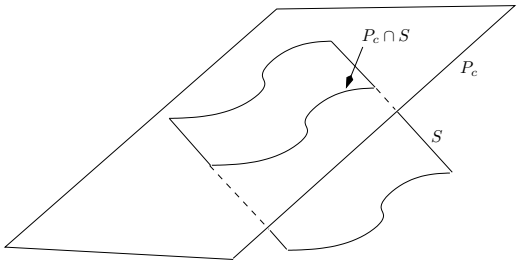

Figure 6.11: No ambiguity in the topological form.

need for repeated subdivision will usually be avoided. On the other hand, there is no guarantee that the algorithm described here avoids this.

It is reasonable here to introduce a simplifying assumption, which restricts slightly the class of problems for which the method is applicable. In general there might be planar subsurfaces of S that lie in P_c, and to permit specification of this kind of case, by a higher-level process catching the exception, it might in general be useful to use Selective Geometric Complexes [RO89] with cells of dimension 2 in the specification of $S \cap P_c$, for purposes of communicating with the higher-level process. (An alternative description of this idea is that $S \cap P_c$ might be represented as a hypergraph with hyperedges sharing three vertices.) For simplicity, however, it is probably sufficient in practice to consider this case to be a loop intersection, and require, for purposes of communicating with the higher-level process, that the intersection of $\mathcal{M}^\nu$ and P_c does not involve two-dimensional sets. Note that, in any case, triangular surface patches can be constrained to lie in the plane by specifying that one-neighbour vertices of the associated triangle lie in the plane. The question of the level of generality provided in the interface with the higher-level process is discussed further in Section 6.5.

6.4.3 Detection of loops within a surface patch

In order to eliminate differences between the topological form of $\mathcal{M}^\nu \cap P_c$, and that of $S \cap P_c$, the second phase of the algorithm first tries to eliminate extraneous intersections of the triangular surface patch which have the form of loops, as in Figure 6.10.

If the vertices of the triangular surface patch were known to lie on the same side of P_c, then a bounding-box approach, in conjunction with repeated subdivision, might be sufficient to confirm that there are no extraneous loop intersections. This does not work, however, if the vertices are on opposite sides of the plane, since a bounding-box approach will simply indicate that there is at least one intersection. Also, again, we would like to remove extraneous intersections without subdivision, if this is possible.

To exclude the possibility of this kind of extraneous intersection (loops), we can make use of bounds on the Gauss map, which is defined as the map that takes a point on the surface to the corresponding unit surface normal.

Proposition 3. *To exclude loop intersections in the triangular surface patch, it is sufficient to check that neither n_c nor $-n_c$ is contained in N_1, the convex hull of the image of the Gauss map of the triangular surface patch.*

Proof. The statement follows directly from [Hoh91, Theorem 1] (and the notation N_1 has been chosen to be consistent with that of the cited theorem). Indeed, if the stated condition is satisfied, then there exists a plane $\{x \in \mathbb{R}^3 : P_2 \cdot x = \sigma\}$ defined by $P_2 \in \mathbb{R}^3$ such that $P_2 \cdot n_c > \sigma$ and $P_2 \cdot n > \sigma$ for $n \in N_1$. (Note that here we have used the condition on $-n_c$.) Further, there is a plane defined by some $P_1 \in \mathbb{R}^3$, separating the image of the Gauss map from n_c, so that $P_1 \cdot n_c < \sigma$ but $P_1 \cdot n > \sigma$ if $n \in N_1$. It then follows from Theorem 1 in [Hoh91] that any intersection of the triangular surface patch with P_c is a curve, a point, or a set of curves and points; and, further, that all isolated point intersections are at the boundaries of the surface patches, the curves do not contain any singularities, and no intersection curve forms a loop. □

Note that the stated condition is not a necessary condition. As described in Section 6.4.5, if at a certain level of subdivision it is not possible to exclude the possibility of a loop intersection, the algorithm will subdivide further, and test again.

A fail-safe estimate of N_1 can be obtained using the method of [GS01, Secs. 3.1-3.4], which is based on the method of [Sta98a] for the direct parametric evaluation of Loop subdivision surfaces. Many details relevant to the implementation are described in [Sta98a, Sec. 4]. The central idea of the method is to decompose the triangular parametric domain of the triangular surface patch into a sequence of subdomains for which the patch is regular ($n = 6$); and, to change coordinates so that the subdivision can be expressed in terms of the eigenbasis of the subdivision matrix, so that subdivision becomes a simple scaling.

6.4.4 Multiple intersections on surface-patch boundary

Suppose now that the test of the previous subsection has confirmed that there is no loop intersection of the triangular surface patch and the plane. We now wish to verify that there is *exactly one intersection along a certain boundary of a triangular surface patch.* The algorithm of Section 6.4.5 excludes the possibility of zero or multiple intersections by using a certain property of Bézier curves, and proceeding as in the case of loop intersections if the test fails, i.e., by subdividing and repeating the test. The result is a modified set C composed only of triangles generating simple intersections. The property mentioned can be described as follows.

A triangular surface patch in Loop subdivision, in the regular case (all vertices have valence $n = 6$) can be expressed as a quartic Bézier surface, defined by the twelve one-neighbours of the mesh triangle (see [Sch96, p. 21] and [Lai92])). Further, an edge of a Bézier patch is a Bézier curve, and the control points of the curve are just the five control points along the border of the triangular array of Bézier control points [APS98]. If we join these five points by line segments, to produce a control polygon for the curve, it follows from the Variation Diminishing Principle for Bézier curves that the curve intersects P_c no more frequently than the Bézier control polygon.

Proposition 4. [PT95, p. 12]. *If the control polygon does not intersect P_c, then the Bézier curve does not intersect P_c. Also, if the control polygon intersects P_c exactly once, then the curve intersects P_c at most once.*

Since a Bézier curve interpolates its endpoints [PT95], the cases of zero and one intersection can easily be distinguished by checking whether the endpoints are on opposite sides of the plane. All of these tests can be verified in a fail-safe manner using interval arithmetic. Again, however, these tests provide only a *sufficient condition.* If the sufficient condition is not satisfied then a subdivision step must be carried out.

6.4.5 Algorithm giving patches with simple intersections

The algorithm is described in the pseudocode given below. We suppose that $\mathcal{M}^{\nu_0}$ is non-empty. If the list C is initially empty, then the planar-cut algorithm should return either $\mathcal{M}^{\nu_0}$ or $\emptyset$, depending on whether $\mathcal{M}^{\nu_0}$ is entirely above, or entirely below, the plane P_c.

Green triangles are treated as pairs, with tests applied to the red mother triangle.

If C is not empty, the algorithm below either

- certifies a triangle in C as generating only simple intersections; or
- replaces the triangle in C by child triangles eligible for certification, or deletes such child triangles because their corresponding triangular surface patch cannot intersect P_c; or
- throws an exception in order to obtain
  - certification; or
  - the information that the triangle can be deleted from C.

Certification of a triangle includes labelling of the edges, to indicate a single intersection on each of a pair of edges. This unfortunately destroys the monotonicity property of certification. The property of having no loop intersections is inherited by any child triangles created by subdivision. On the other hand, to satisfy the single-intersection test on a given edge, it is necessary to do several things:

- the sufficient condition provided by the Variation Diminishing Principle must be satisfied;
- the triangle vertices must have the same above-below status, with respect to the plane P_c, as the corresponding vertices of the associated triangular patch;
- the end points of the edge must be regular ($n = 6$).

The first two of these conditions are verified using interval arithmetic. Although the property that an edge does not have multiple intersections is inherited by children created by subdivision, the ability to confirm all conditions using interval arithmetic is not. This means that in principle it is possible that a child of a certified triangle cannot be certified (see pseudocode, below). Convergence of the algorithm is guaranteed, however, by the geometric reduction in triangle size due to subdivision. Similarly, whether or not the various tests eventually pass, the fact that triangle size is eventually reduced to the size of data uncertainty means that if the algorithm throws an exception, it is because there exists an additional perturbation of S of that size which could change the topological form.

Algorithm 1 Pseudocode that gives patches with simple intersections

```
while there exist uncertified triangles in C do
  if size of triangle is smaller than uncertainty level then
    if only the above-below status of one or two endpoints of an edge of a
    triangle is incorrect then
      Perturb the triangle endpoints in a direction parallel to n_c, to correct the
      situation.
    else
      Throw an exception, asking the higher-level process to specify the topol-
      ogy of the intersection curves on the triangle.
    end if
  else if Wu-Peters guarantees there is no intersection then
    Delete the triangle from C.
  else if either the loop test or single-intersection test fails then
    Apply RG refinement. Mark the child triangles as uncertified, and put them
    in C in place of the mother triangle.
  else
    Label the edge containing the single edge intersection, and certify the trian-
    gle.
  end if
end while
```

Finally, the algorithm scans the vertices of the mesh. If the limit position of a vertex cannot be isolated from P_c using interval arithmetic, the vertex is perturbed in a direction parallel to n_c to correct the situation.

The algorithm can be made more efficient by making use of the fact that the loop test does not have to be repeated for triangles that are the children of a triangle that has already passed the test. (This monotonicity property for the loop test was mentioned above, within this subsection.)

In the case when an exception is thrown, specification of the topology may involve an indication that there is no intersection, so that the triangle can be deleted from C, or it may involve specification of multiple edge intersections, as in the case of a saddle point. This is discussed briefly in Section 6.5.

6.4.6 Tracing the graph defining the topology of $S \cap P_c$

If the higher-level process is permitted only to specify triangles that have single edge intersections, the tracing procedure is straightforward. The intersection of the edge curve is known to exist, and we can compute an approximation to the intersection between the triangle and P_c. Following the triangle adjacency links within the mesh permits us to continue the process in the adjacent triangle. Since we are assuming that each edge of the triangular surface patch has zero or one (non-composite) intersections with the plane, the intersection path forms a closed loop.

In the more general case more elaborate procedures must be implemented. This question is also discussed briefly in the next section.

6.5 Discussion and future work

In previous sections we have described our general approach to the problem described in Section 6.3. There remain, however, many interesting research questions, and several major issues to be resolved in the light of more experimentation. In this section we first give a brief discussion of our prototype implementation, followed by a summary of the major issues just mentioned.

6.5.1 The prototype implementation

The major components of the algorithm described above have been implemented, but not always in full generality. In particular, the fail-safe calculation of N_1, mentioned at the end of Section 6.4.3, is replaced by an estimate. Study of the modifications necessary [GS01, Sec. 3.5] to obtain tight bounds for N_1 constitutes an interesting research topic in itself, but it is a topic that is largely independent of the one discussed here.

Our prototype implementation is written in C++ using standard tools, such as *OpenGL* and *QT*. We also make use of the *CGAL* library [CGA12] for basic subdivision, and for other tasks, such as checking the condition of Proposition 3. Similarly, we used the *MPFI* package [MPF12] for interval arithmetic.

The prototype implementation does not achieve interactive execution rates. To illustrate, for the model of Figure 6.9, which has (after a preliminary subdivision to isolate extraordinary vertices) 304 faces, 456 edges, and 154 vertices, the solution required about 540 *milliseconds* to compute the planar cut, when no traps occurred. Although these times could quite likely be improved with a GPU implementation, it is important to emphasize that efficiency is not our main concern. Our point of view is different here: rather than requiring interactive solution rates, we have made correct topological form a tight constraint. We then try to minimize solution times subject to this constraint.

It is unlikely that the approach described in this chapter would be of interest in the context of applications such as gaming, even if interactive rates could be achieved. It is in traditional CAD applications that correct topological form can be crucial [HS05, Far99, ASZ07].

6.5.2 Future work

The main questions left open are related to the design of the interface for exception handling, and the elimination of cases where the algorithm traps even though there exists no small perturbation of the data which could lead to a change in the

topological form. Also, it may be possible to replace the backward error approach of Section 6.4 by one based entirely on a direct forward error analysis, which is discussed briefly in Chapter 7.

One aspect of the interface design is the level of generality to be treated. For example, in the specification of the topology, we might permit the higher-level process to specify only closed-loop topologies; alternatively, we might also permit the specification of saddle points, as mentioned at the end of Section 6.4.5. The former choice would limit the generality of the cases that could be handled, while the latter would require the implementation of a more elaborate tracing procedure, as mentioned at the end of Section 6.4.6. In the latter case it would be possible for the higher-level process to specify triangle deletion, or a saddle-point intersection involving four incident path segments. Even the generality of the latter case would rarely be needed in practice, but in principle, the specification of even more general types of topology might be permitted. An example is the possibility of permitting specification of planar intersections of the mesh and P_c, as mentioned at the end of Section 6.4.2.

As stated at the end of Section 6.3, our goal is to establish weak conditions guaranteeing that the algorithm will trap and ask for additional information only in the case when a perturbation of the original data, approximately equal in size to the data uncertainty, could lead to a change in the topological form. But (again as stated at the end of Section 6.3) the fact that the algorithm traps only if there exists a perturbation of the surface S itself which would change the topological form does not necessarily *guarantee* that there exists a perturbation of the original data that has this effect. In fact, the latter condition is very likely in practice, but in future work, we would like to narrow the gap between "guarantee" and "very likely". This possibility is discussed in the remaining paragraphs of the chapter.

In this context, the test of Proposition 3 has as its intuitive basis the idea that if the surface (viewed as a function defined locally over P_c) is nearly horizontal, then because of undulations in the surface, a translation of the original control points will lead to a change in topological form. If the Wu-Peters bounding box has size

on the order of the uncertainty, this means that if the algorithm has thrown an exception, it has done so in a case when there is no other choice: a perturbation of the original control points, on the order of the uncertainty in the data, would have led to a change in topological form.

The test does not, however, provide perfectly tight necessary conditions. For example, it could fail because of a saddle point, or because the bound on the surface normal provided by N_1 is not tight, so that the surface normal cannot be equal to $\pm n_c$, even though we are unable to guarantee this.

The consequence of such gaps between the strength of conditions is not algorithm failure, but unnecessary trapping. This will happen very rarely, but nonetheless, it would be better to eliminate such behaviour to whatever extent possible. In fact, it should be possible to design stronger tests, for example by giving conditions which assure that there exists a local relative minimum [Lue73, Sec. 6.2], so that again a translation of the original control points will guarantee a change in topological form. On the other hand, the computational cost of such tests must be examined.

CHAPTER 7

DISCUSSION AND CONCLUSION

7.1 Forward analysis

The work in Chapter 6 uses the backward error analysis to take into account the error in the calculation of geometric values for vertices that have been subdivided adaptively. This approach assumes that the problem has been perturbed by an amount smaller than the data uncertainty present in the problem. This way, the proposed algorithm will give a topologically correct result for a perturbed problem, and the same result can serve as the result for the original problem since those perturbations are often much smaller than the data uncertainty in the original problem.

In this section we consider the same problem but from a different point of view—we adopt the forward error analysis here. Instead of dealing with the perturbed problem and using the fact that the perturbations are usually much smaller than data uncertainty in the given data that define the problem, we deal with the problem directly and solve it by computing and considering the bounding error that results from the calculation. As the relative error in the calculation of geometric values for vertices at level $(\lambda_0+\lambda)$ can always be computed using interval arithmetic or bounded using the standard model [DB07, Ch. 2] (the *runnning error analysis* [Hig02, p. 72] provides an idea of how rounding error can be bounded tightly at the same time of the execution of the computation that needs to be bounded), we can therefore propose an approach that directly computes the result with correct topology in well-conditioned cases, or throw an exception in other cases.

This approach is different from the one presented in Chapter 6 in that the computation of the geometric values of mesh vertices and limit vertices is done entirely using interval arithmetic. For example, the geometric values of a vertex p_ℓ, when subdivided adaptively to the level $(\lambda_0 + \lambda)$, where $\lambda_0 \geq 0$ and $\lambda \geq 1$,

can be computed using the MPFI package [MPF12], and described as a triple $(IA(x), IA(y), IA(z))$, where $IA(x)$, $IA(y)$, and $IA(z)$ are intervals representing the cuboid where the vertex $p_\ell(\lambda_0 + \lambda)$ may locate. Since geometric values of the meshes are all represented in interval arithmetic, the algorithm that gives patches with simple intersections (refer to Section 6.4.5) also uses intervals for the corresponding Wu-Peters test, loop test, and single-intersection test.

7.2 Summary and further work

In this thesis, we targeted the robustness of the problem of Boolean operations that are fundamental and important in the field of solid modelling. We considered the problem where the solids are represented using Loop subdivision surfaces, and in the context that ordinary IEEE arithmetic, the widely accepted standard for floating-point computation, and interval arithmetic, are used, and that an error criterion that requires correct topological form is used.

As uncertainty in the data is unavoidable, our point of view was to deal with the problem as one of managing uncertainty. We proposed an interactive approach for the robust implementation of geometric operations with the aim to obtain the correct topological form of the result. When this is not possible, i.e., when supplementary topological information is needed, the algorithm throws an exception, which will be caught by a higher-level process that is capable of providing such information when demanded.

One question, that has been left open, is the design of the interface for the communication between the higher-level process and the Boolean solver. There also exist cases where the algorithm that deals with the planar-cut problem traps unnecessarily. This may happen, for example, when there is an extraordinary vertex whose limit values are very near the plane. Another reason for unnecessary trapping is that some tests in the algorithm use sufficient conditions that are not necessary: for example, Proposition 3 in Section 6.4.3 is derived from Hohmeyer's conditions that are sufficient but not necessary.

Another open question is the generality of topology that is allowed for surfaces in the Boolean solver. Accordingly, our representation of subdivision surfaces may need to be generalized: by introducing piecewise smooth subdivision surfaces the solver will support free-form surfaces with creases and corners, and by extending subdivision rules [YZ01] nonmanifold surfaces will be allowed in the solver. And if more general surfaces are involved in the problem definition, our algorithm that gives patches with simple intersection and a more elaborate tracing procedure for the topology of the Boolean result will have to be reformulated to cover more general topology.

BIBLIOGRAPHY

[AD03] Bart Adams and Philip Dutré. Interactive Boolean operations on surfel-bounded solids. *ACM Trans. Graph.*, 22:651–656, July 2003.

[ADPS95] Lars-Erik Andersson, S. M. Dorney, T. J. Peters, and N. F. Stewart. Polyhedral perturbations that preserve topological form. *Comput. Aided Geom. Des.*, 12(9):785–799, December 1995.

[AFR02] Ashish Amresh, Gerald Farin, and Anshuman Razdan. Adaptive subdivision schemes for triangular meshes. In *Hierarchical and Geometric Methods in Scientific Visualization*, pages 319–327. Springer-Verlag, 2002.

[AMP00] Dago Agbodan, David Marcheix, and Guy Pierra. Persistent naming for parametric models. In *WSCG*, pages 17–38, 2000.

[APS98] Lars-Erik Andersson, Thomas J. Peters, and Neil F. Stewart. Selfintersection of composite curves and surfaces. *Computer Aided Geometric Design*, 15:507–527, 1998.

[APS00] Lars-Erik Andersson, Thomas J. Peters, and Neil F. Stewart. Equivalence of topological form for curvilinear geometric objects. *International J. of Computational Geometry and Applications*, 10:609–622, 2000.

[AS10] Lars-Erik Andersson and Neil F. Stewart. *Introduction to the Mathematics of Subdivision Surfaces.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2010.

[ASZ07] Lars-Erik Andersson, Neil F. Stewart, and Malika Zidani. Error analysis for operations in solid modeling in the presence of uncertainty. *SIAM J. Sci. Comput.*, 29:811–826, March 2007.

[BF10] Richard L. Burden and J. Douglas Faires. *Numerical Analysis.* Brooks Cole, Pacific Grove, California, United States, 9th edition, 2010.

[BK04] Stephan Bischoff and Leif Kobbelt. Teaching meshes, subdivision and multiresolution techniques. *Computer-aided Design*, 36:1483–1500, 2004.

[BKZ01] Henning Biermann, Daniel Kristjansson, and Denis Zorin. Approximate Boolean operations on free-form solids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 185–194, New York, NY, USA, 2001. ACM.

[Ble11] Blender. Blender, 2011. Accessed 2011. `http://www.blender.com`.

[BLZ00] Henning Biermann, Adi Levin, and Denis Zorin. Piecewise smooth subdivision surfaces with normal control. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 113–120, New York, NY, USA, 2000. ACM.

[BM99] S. B. Brunnermeier and S. A. Martin. Interoperability cost analysis of the U.S. automotive supply chain. Technical report, Research Triangle Institute, Center for Economics Research, 1999.

[BP00] M. Bern and P. Plassmann. Mesh generation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 6, pages 291–332. Elsevier science, 2000.

[BPK+07] Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, Bruno Lévy, Stephan Bischoff, and Christian Rössl. Geometric modeling based on polygonal meshes. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, New York, 2007. ACM.

[BS92] M. Boyer and N. F. Stewart. Imperfect form tolerancing on manifold objects: a metric approach. *Int. J. Rob. Res.*, 11:482–490, October 1992.

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

[CGA12] CGAL. Computational geometry algorithms library, 2012. Accessed 2012. `http://www.cgal.org`.

[CLL07] Zhongxian Chen, Xiaonan Luo, and Ruotian Ling. An adaptive subdivision method based on limit surface normal. In *Tenth IEEE International Conference on CAD and CG*, pages 65–70, 2007.

[CM99] G. Casciola and S. Morigi. The trimmed NURBS age. In Donato Trigiante, editor, *Advances in Computation: Theory and Practice; Recent Trends in Numerical Analysis*. Nova Science Publishers, Inc., 1999.

[CXS12] CXSC. CXSC, 2012. Accessed 2012. `http://www2.math.uni-wuppertal.de/~xsc/xsc/cxsc.html`.

[DB07] G. Dahlquist and A. Björck. *Numerical Methods in Scientific Computing, Volume 1*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2nd edition, 2007.

[dBHR93] Carl de Boor, K. Hollig, and D. Riemenschneider. *Box Splines*. Springer-Verlag Berlin, 1993.

[DKT98] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 85–94, 1998.

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

[EC95] Ioannis Z. Emiris and John F. Canny. A general approach to removing degeneracies. *SIAM J. Comput.*, 24:650–664, June 1995.

[EM90] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph*, 9:66–104, 1990.

[Far99] Rida T. Farouki. Closing the gap between CAD model and downstream application, 1999. SIAM News.

[FHK02] G. Farin, J. Hoschek, and M.-S. Kim. *Handbook of Computer Aided Geometric Design*. Elsevier, Amsterdam, 2002.

[FM67] G. E. Forsythe and C. B. Moler. *Computer solution of linear algebraic systems*. Prentice-Hall, 1967.

[For89] S. Fortune. Stable maintenance of point set triangulations in two dimensions. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 494–499, Washington, DC, USA, 1989. IEEE Computer Society.

[For92] Steven Fortune. Numerical stability of algorithms for 2D Delaunay triangulations. In *Proceedings of the eighth annual symposium on Computational geometry*, SCG '92, pages 83–92, New York, NY, USA, 1992. ACM.

[GD03] Philippe Guigue and Olivier Devillers. Fast and robust triangle-triangle overlap test using orientation predicates. *Journal of Graphics, GPU, and Game Tools*, 8(1):25–42, 2003.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 171–180, New York, NY, USA, 1996. ACM.

[GO97] Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry*. CRC Press, Inc., Boca Raton, FL, USA, 1997.

[Gra00] Thomas A. Grandine. Applications of contouring. *SIAM Rev.*, 42(2):297–316, June 2000.

[GS01] Eitan Grinspun and Peter Schröder. Normal bounds for subdivision-surface interference detection. In *Proceedings of the conference on visualization '01*, VIS '01, pages 333–340, Washington, DC, USA, 2001. IEEE Computer Society.

[HDD+94] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 295–302, New York, NY, USA, 1994. ACM.

[Hel98] Martin Held. ERIT - a collection of efficient and reliable intersection tests. *Journal of Graphics Tools*, 2:25–44, 1998.

[Hig02] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2002.

[Hof01] C. M. Hoffmann. Robustness in geometric computations. *Computing and Information Science in Engineering*, 1:143–156, 2001.

[Hoh91] Michael E. Hohmeyer. A surface intersection algorithm based on loop detection. In *Proceedings of the first ACM symposium on solid modeling foundations and CAD/CAM applications*, SMA '91, pages 197–207, New York, NY, USA, 1991. ACM.

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

[IEE08] IEEE, New York. *IEEE Standard for Floating-Point Arithmetic*, 2008. IEEE Std 754-2008.

[ISO97] ISO. *STEP International Standard, ISO 10303-42*, 1997.

[Jia08] Di Jiang. *Reliable Computation for Geometric Models*. PhD thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal, 2008.

[KBF05] David J. Kasik, William Buxton, and David R. Ferguson. Ten CAD challenges. *IEEE Comput. Graph. Appl.*, 25:81–92, March 2005.

[KCVS98] Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 105–114, 1998.

[Kob96] Leif Kobbelt. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In *Computer Graphics Forum*, volume 15, pages 409–420, 1996.

[Kob98] Leif Kobbelt. Tight bounding volumes for subdivision surfaces. In *Proceedings of the 6th Pacific Conference on Computer Graphics and Applications*, Pacific Graphics '98, pages 17–. IEEE Computer Society, 1998.

[KS88] Sheldon Katz and Thomas W. Sederberg. Genus of the intersection curve of two rational surface patches. *Computer Aided Geometric Design*, 5:253–258, 1988.

[Lai92] Ming-Jun Lai. Fortran subroutines for b-nets of box splines on three- and four-directional meshes. *Numerical Algorithms*, 2:33–38, 1992.

[LC06] Shuhua Lai and Fuhua (Frank) Cheng. Voxelization of free-form solids using Catmull-Clark subdivision surfaces. In *Lecture Notes in Computer Science*, pages 595–601. Springer, 2006.

[LC07] S. Lai and F. Cheng. Robust and error controllable Boolean operations on free-form solids represented by Catmull-Clark subdivision surfaces. *Computer Aided Design and Applications*, 4:487–496, 2007.

[Lev99a] Adi Levin. Combined subdivision schemes for the design of surfaces satisfying boundary conditions. *Computer Aided Geometric Design*, 16:345–354, 1999.

[Lev99b] Adi Levin. Interpolating nets of curves by smooth subdivision surfaces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 57–64, New York, NY, USA, 1999. ACM.

[LLS01] Nathan Litke, Adi Levin, and Peter Schröder. Trimming for subdivision surfaces. *Computer Aided Geometric Design*, 18:463–481, 2001.

[Loo87] C. Loop. Smooth Subdivision Surfaces Based on Triangles. Master thesis, Department of Mathematics, University of Utah, Utah, USA, August 1987.

[Lue73] D. G. Luenberger, editor. *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, 1973.

[Man87] Martti Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, Inc., New York, NY, USA, 1987.

[Mar05] David Marcheix. A persistent naming of shells. In *Proceedings of the Ninth International Conference on Computer Aided Design and Computer Graphics*, CAD-CG '05, pages 259–268, Washington, DC, USA, 2005. IEEE Computer Society.

[MKC09] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

[Möl97] Tomas Möller. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2:25–30, 1997.

[MP02] David Marcheix and Guy Pierra. A survey of the persistent naming problem. In *Proceedings of the 7th ACM Symposium on Solid Modeling and Applications*, SMA '02, pages 13–22, New York, NY, USA, 2002. ACM.

[MPF12] MPFI. Multiprecision interval arithmetic library (MPFI), 2012. Accessed 2012. `http://perso.ens-lyon.fr/nathalie.revol/software.html`.

[MS02] Andreas Meister and Jens Struckmeier. *Hyperbolic Partial Differential Equations*. Springer Vieweg, 2002.

[Owe98] Steven J. Owen. A survey of unstructured mesh generation technology. In *International Meshing Roundtable*, pages 239–267, 1998.

[P1712] P1788. IEEE interval standard p1788, 2012. Accessed 2012. `http://grouper.ieee.org/groups/1788`.

[Pat02] Nicholas M. Patrikalakis. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[PP09] Enrico Puppo and Daniele Panozzo. RGB subdivision. *IEEE Transactions on Visualization and Computer Graphics*, 15:295–310, 2009.

[PT95] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer-Verlag, London, UK, 1995.

[Req80] Aristides A. G. Requicha. Representations for rigid solids: theory, methods, and systems. *ACM Computing Surveys*, 12:437–464, 1980.

[RO89] J. Rossignac and M. O'Connor. SGC: a dimension-independent model for pointsets with internal structures and incomplete boundaries. In M. Wosny, J. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 145–180. North-Holland, 1989.

[RR99] Jarek R. Rossignac and Aristides A. G. Requicha. Solid modeling. In John G. Webster, editor, *Encyclopedia of Electrical and Electronics Engineering*. John Wiley and Sons, Inc., 1999.

[RS97] Ari Rappoport and Steven Spitz. Interactive Boolean operations for conceptual design of 3-D solids. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 269–278, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[SAG84] Thomas W. Sederberg, D. C. Anderson, and Ronald N. Goldman. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics, and Image Processing*, 28:72–84, 1984.

[Sam90] Hanan Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[Sch96] J. E. Schweitzer. *Analysis and Application of Subdivision Surfaces*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1996.

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

[Sei94] Raimund Seidel. The nature and meaning of perturbations in geometric computing. In *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '94, pages 3–17, London, UK, 1994. Springer-Verlag.

[She00] Jonathan Richard Shewchuk. Mesh generation for domains with small angles. In *Proceedings of the 16th annual symposium on computational geometry*, SCG '00, pages 1–10, New York, NY, USA, 2000. ACM.

[Spa99] Spatial Technology. *The ACIS 3D Toolkit Guide*, 1999.

[SR04] Steven Spitz and Ari Rappoport. Integrated feature-based and geometric CAD data exchange. In *Proceedings of the ninth ACM symposium on Solid modeling and applications*, SM '04, pages 183–190, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[SS13] Peihui Shao and Neil F. Stewart. Managing uncertainty and discontinuous condition numbers in geometric computation. *Soft Computing*, 2013. To appear.

[Sta98a] Jos Stam. Evaluation of Loop subdivision surfaces. In *CD-ROM Proceedings of SIGGRAPH'98*. ACM, 1998.

[Sta98b] Jos Stam. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 395–404, 1998.

[Ste11] N. F. Stewart. *Boolean operations on subdivision-surface objects with interactive removal of ill condition*, 2011. Draft document.

[Str06] I. Stroud. *Boundary Representation Modelling Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[SV91] Vadim Shapiro and Donald L. Vossler. Construction and optimization of CSG representations. *Comput. Aided Des.*, 23:4–20, February 1991.

[Tre92] Lloyd N. Trefethen. The definition of numerical analysis. Technical report, Cornell University, September 1992.

[VR93] H. B. Voelcker and A. A. G. Requicha. Research in solid modeling at the University of Rochester: 1972-87. In L. Piegl, editor, *Fundamental Developments of Computer-Aided Geometric Modeling*, pages 203–254. Academic Press, New York, 1993.

[Wil65] J. H. Wilkinson, editor. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1965.

[WP04] Xiaobin Wu and Jorg Peters. Interference detection for subdivision surfaces. In *Proceedings of the Fifth International World Wide Web Conference*, pages 96–107, 2004.

[Wu05] Xiaobin Wu. An accurate error measure for adaptive subdivision surfaces. In *Proceedings of the international conference on shapes and solids*, pages 51–56, 2005.

[Yap90] Chee-Keng Yap. Symbolic treatment of geometric degeneracies. *Journal of Symbolic Computation*, 10:349–370, August 1990.

[Yap08] Chee Yap. Reliable implementation of real number algorithms: Theory and practice. In Peter Hertling, Christoph M. Hoffmann, Wolfram Luther, and Nathalie Revol, editors, *Reliable Implementation of Real Number Algorithms: Theory and Practice*, chapter Theory of Real Computation According to EGC, pages 193–237. Springer-Verlag, Berlin, Heidelberg, 2008.

[YZ01] Lexing Ying and Denis Zorin. Nonmanifold subdivision. In *Proceedings of the conference on visualization '01*, VIS '01, pages 325–332, Washington, DC, USA, 2001. IEEE Computer Society.

[Zor06] Denis Zorin. Modeling with multiresolution subdivision surfaces. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, pages 30–50, New York, NY, USA, 2006. ACM.

[ZSS97] Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *Proceedings of the 24th annual conference on computer graphics and interactive techniques*, SIGGRAPH '97, pages 259–268, New York, NY, USA, 1997. ACM.