Chapter 1

Introduction

Solid modeling is rapidly emerging as a central area of research and development in such diverse applications as engineering and product design, computer-aided manufacturing, electronic prototyping, off-line robot programming, and motion planning. All these applications require representing the shapes of solid physical objects, and such representations and basic operations on them can be provided by solid modeling.

As a field, solid modeling spans several disciplines, including mathematics, computer science, and engineering. In consequence, it is a broad subject that must accommodate a diversity of viewpoints and has to meet a diversity of goals. Sometimes, this diversity of goals can lead to conflicting demands. Current thinking on the subject views the proper resolution of these conflicts to be application-dependent. That is, it is no longer thought realistic to envision a comprehensive solid-modeling system that satisfies the needs of all potential users. Rather, as it is argued, we should concentrate on constructing a software environment in which many tools for geometric and solid computation are available and can be combined with ease as appropriate for the specific application under consideration.

Whether we seek to build a complete system or wish to accumulate a set of tools, we need to study and implement many geometric algorithms. In this book, we explain what is needed for this task. Of necessity, we must therefore cover techniques from computer science, numerical analysis, symbolic computation, and many other areas. The relevant facts from these

areas are brought together as they are needed. Not only do we develop them technically, we also explain their intuitive content as much as possible, for we have found that intuitive explanations accelerate absorbing the material. Moreover, an intuitive understanding of the ideas underlying a particular subject serves as a guide when special topics are pursued further by reading the pertinent literature, be it for research purposes or for specializing certain methods as demanded by some application.

Geometric or *surface modeling* traditionally identifies a body of techniques that can model certain classes of piecewise parametric surfaces, subject to particular conditions of shape and smoothness. It developed as a separate field in several industries, including automobile, aerospace, and shipbuilding, and has some of its intellectual roots in approximation theory. It is our view that the streams of geometric and solid modeling are converging. As solid modeling strives to extend the geometric coverage, there is an emerging need to research the use of surface forms and the techniques to interrogate them. Similarly, as geometric modeling contemplates building complete solid representations from surface patches, the usefulness of traditional solid-modeling techniques is more widely recognized. In anticipation of the growing importance of this convergence, a large part of this book is devoted to geometric investigations of implicit and parametric surfaces.

Although we will develop a lot of machinery to deal with surface geometries, we do not cover the traditional body of knowledge on parametric surfaces developed by classical geometric modeling. Instead, we concentrate on the fundamental issues that are at the focal point of the possible integration of geometric and solid modeling, and develop techniques that have the potential to bridge the current gaps between the two areas of activity. Thus, we understand geometric modeling in the more generic sense.

1.1 A Brief Historical Perspective of Solid Modeling

As a field, solid modeling is the outgrowth of several convergent developments. These include automatic drafting systems, free-form surface design, and graphics and animation.

Computer drafting systems replace manual engineering drawing. Some of the benefits are that electronic drawings can be modified and archived more easily, and that one may verify automatically the validity of a design by a program rather than by human inspection. Complicated engineering drawings may contain errors. Such errors in the electronic counterpart may, in principle, be corrected without the risk of introducing new errors elsewhere. However, programs to verify design validity are nontrivial, and research on this subject continues.

Early efforts in automated drawing resulted in wireframe modeling systems — that is, in systems in which only the edges and vertices of objects are represented. This would be a natural representation, assuming that the objective is to generate line drawings of the design, projected in certain

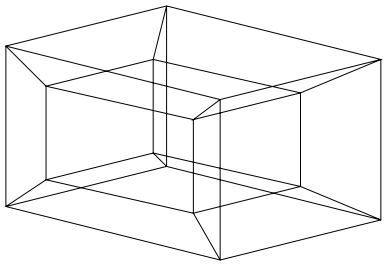

Figure 1.1 Ambiguous Wireframe Object

directions. Unfortunately, there may be ambiguities in interpreting the representation. A simple, well-known example of this phenomenon is shown in Figure 1.1. The figure shows a block with a beveled hole through its center. It is not possible to deduce the direction of the hole, since it could lie in any of the three principal directions. For this reason, wireframes are not the preferred object representation. On the other hand, wireframe objects have small storage requirements and can be accessed and displayed quickly. The resulting line drawings constitute an acceptable visualization aid in many situations, and can provide a quick feedback between the modeling system and the designer. For this reason, many modeling systems retain the capability to generate wireframe drawings.

A second contributor to solid modeling has been free-form surface design, using parametric surface patches and, in particular, various types of spline surfaces. Surface design with splines originated in the automobile industry, principally for car body design; in the shipbuilding industry, for the design of ship hulls; and in the aircraft industry, for the design of wings, fuselages, and so on. Free-form surface design in these areas has led to the field of computer-aided geometric design (CAGD), primarily focusing on methodologies for designing curved surfaces subject to aesthetic or functional constraints. Research in CAGD has discovered many useful classes of parametric surfaces and has developed a large repertoire of algorithms for their design, analysis, and manipulation. These surfaces would be most useful in solid modeling. As we stated, we cover basic concepts underlying the use of these surfaces in solid modeling. Many books are available that cover the more classical techniques of modifying the shapes of specific surfaces classes, and we cite some of them at the end of this chapter.

The difficulty of evaluating and representing the intersection of parametric surface patches has hindered the development of solid modelers that incorporate parametric surfaces. Roughly speaking, the topology of a surface patch becomes quite complicated when Boolean operations are performed. Finding a convenient representation for these topologies continues to be a major challenge.

It might seem peculiar to identify computer graphics as an area contributing to solid modeling. Indeed, the primary focus of computer graphics is to

render realistic images of objects, and this can be done from data structures that do not represent complete solids. However, there is a need in solid design to obtain visual feedback, so we want to render images from solid representations, and thus an understanding of graphics algorithms and the surface representations used by them can be useful. Conversely, constructing images from solid models is gaining in importance in animation and is also used by rendering algorithms that support image generation in scenes from changing points of view. Thus, computer graphics is beginning to deal with solid models as data and could make useful contributions to visualization techniques and to user-interface design.

1.2 Three Levels of Abstraction

Conceptually, a solid-modeling system spans three levels of abstraction:

1. The user of the modeling system is presented the highest level of abstraction, the *user interface*. He or she interacts with the system through a design language that may be textual, visual, or both. On this level, conceptual tools for constructing, modifying, archiving, and destroying designs are available. Also, there may be various tools for analyzing a design, perhaps even for reasoning about some of its properties.

2. Next, there is a lower level of abstraction comprising the *mathematical and algorithmic infrastructure*. The infrastructure implements the conceptual operations available in the user interface, as well as a wide range of auxiliary tools needed by these operations. Examples include algorithms for constructing the intersection of two objects, or tools for determining whether and how two curved surfaces intersect.

3. On the lowest level, there is the *substratum* of arithmetic and symbolic computations that are used as primitives by the algorithmic infrastructure. In the most basic sense, this substratum consists of the hardware capabilities for integer and floating-point arithmetic, and the logical operations the chosen programming language offers for expressing computations and viewing storage.

Computer science teaches that levels of abstraction should be kept logically separate, and that lower levels must not unduly influence higher levels. Yet, as we shall see, this ideal situation appears to be presently unattainable, except at the great expense incurred by exact arithmetic, so certain operations done at the highest conceptual level take their particular form essentially because of unreliabilities on the lowest level.

1.2.1 User Interfaces

Much of the interest in solid modeling is due to the perceived value of automating the design and analysis of solid objects. If the field is to reap the benefits that good solid-modeling systems could provide, then modeling must be made as widely accessible as possible, and must be developed into as flexible an instrument as possible. Much of this would be the result of having good user interfaces that successfully engage people with minimum training and increase the productivity of experienced designers.

It is widely accepted that user interfaces should have a strong visual component, but clearly a textual interaction is also required; for instance, for the sake of precision. Furthermore, if the modeling system is interfaced to an analysis system, a more machine-oriented interface must also be present.

A major task of the interface is to present the user with a set of operations for solid design and modification. Among these operations are the well-known Boolean operations, global modifications such as rounding and offsetting, and local modifications such as edge beveling or face extrusion. In many cases, the operations are not fully understood. For example, when sweeping an area along a space curve to define a volume, should we allow possible self-intersection?

In doing detail design, we would like to concentrate our effort on local areas of interest and to design them more or less without paying attention to the rest of the design. Having completed the design of such *features*, we could then specify their position and orientation in the larger design through *constraints*. Much current research explores these goals of interface design. It is not clear how best to define features, yet we know that the concept is needed. The notion of features is probably not a static concept; that is, the same geometric design of an object will have different features, depending on the view point. For instance, if we consider how to evaluate stress concentrations, then features such as sharp edges are of interest. If we consider machining operations, we might be interested in the geometry of holes and slots found on the object. The main difficulty, it seems, is to identify a catalog of forms defining a set of features of interest, and to understand how these forms may interact. For example, if a slot is placed at the edge of an object, as shown in Figure 1.2 to the right, should we still consider it a slot?

Current thinking also stresses the need for *conceptual design*; that is, for a coarse design laying out the overall structure without various details. It is argued that a more efficient approach to engineering design is an overall outlay of the rough shape, with rapid visual feedback, exploration of the suitability of the design through some suite of analysis computations, followed by a computationally intensive detailed design. This approach should not only alleviate some of the delays incurred by certain expensive detailed design operations, but also bring us closer to the long-term goal of *design by functionality*. In the abstract, we may not care about a particular shape, as long as it realizes a certain functionality. For example, in designing a

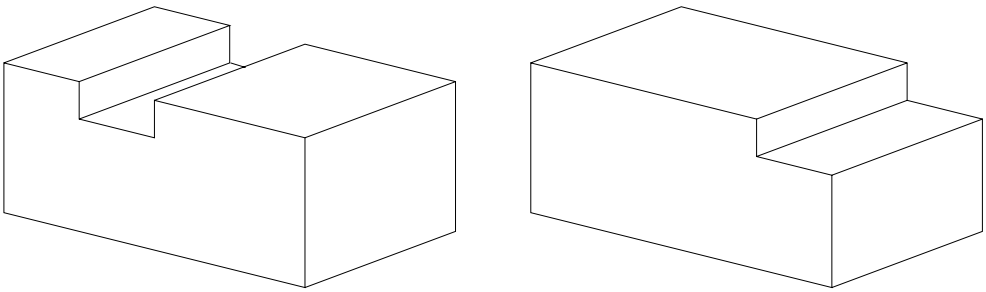

Figure 1.2 Two Slot Positions

piston engine like the one shown in Figure 1.3, our functional goals might include that the engine develop a certain power while not running too hot or being too heavy. As design parameters, we might wish to vary piston size, stroke length, wall thickness, and cooling-line placement. It seems certain that design by functionality is a possibility, but we have no systematic body of knowledge establishing this connection precisely in a range of applications.

1.2.2 Mathematical and Algorithmic Infrastructure

Infrastructure is traditionally the strongest and most prominent research subject in solid modeling. Among the many questions addressed is the development of efficient and robust algorithms and representations for solving

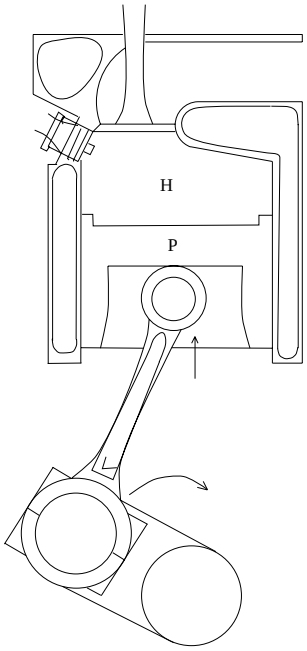

Figure 1.3 Piston Engine

the geometry problems that arise in solid modeling. We mention a few:

1. Given two solid models, test whether they interfere with each other. If so, determine the volume of their intersection.

2. Given two curved faces intersecting in a sharp edge, find a surface that smoothly connects the two curved faces, thereby blending the edge.

3. Given a curved face of a solid model and a point on the surface, determine whether this point lies within the face boundary.

4. Given a closed piecewise algebraic curve in the plane, sweep it along a space curve and determine the surface of the volume so swept.

Since efficient algorithms depend on suitable representations for solids and their constituting elements (i.e., for their faces, edges, and vertices), it is crucial to study different schemata for representing them. Moreover, since there are typically no uniform best choices, conversion algorithms between different representations must be designed. In all these algorithms, we must account for efficiency in space and time, as well as for numerical stability and accuracy.

1.2.3 The Substratum Problem

The distinction between substratum and infrastructure may appear arbitrary, and we should clarify why it is made here. When implementing many of the geometric algorithms found in the literature, one encounters a fundamental difficulty: Even though an algorithm may have been meticulously implemented, it need not be completely free from errors. But these errors seem to be due neither to carelessness in the implementation, nor to a mistake in the algorithm design. Rather, the error is due to an oversight — we take for granted that the arithmetic operations in the geometric computation are precise. Of course, we realize that floating-point arithmetic is approximate, but we might have assumed that the errors so incurred are insignificant. This is not always the case, and the distinction between substratum and infrastructure helps to conceptualize the nature of the difficulty.

A computer does two types of computation: symbol manipulation, which can be done exactly, and numerical computation. The latter is exact only for integer and rational arithmetic, and is subject to imprecision in floating-point arithmetic. Floating-point error is due to the limited precision to which the calculations are done and suffers from *roundoff* and *digit-cancellation* errors. The two problems are well known in numerical analysis and have given rise to extensive research into designing algorithms that exhibit greater accuracy and numerical stability, besides solving a problem efficiently.

The vast majority of geometric computations in solid modeling are performed in floating-point arithmetic. Since logical decisions are made based on these calculations, errors incurred by the arithmetic should be of great

concern. In particular, since the same logical decision may recur throughout a computation but may have been based on different calculations, there is the possibility of making inconsistent decisions. It is precisely this possible inconsistency that causes solid-modeling systems to fail on certain inputs.

The interplay of symbolic and arithmetic computation is a critical dimension in solid modeling and appears to be unparalleled. It raises fundamental problems of profound mathematical content, and there is a growing sense in the field that these problems need to be addressed urgently. At this time, it appears that there are three choices:

1. Create a substratum implementing exact arithmetic. Typically, this slows down all computations unacceptably, but in some situations a priori precision bounds exist, and then this approach may lead to acceptable speeds.

2. Use an inexact arithmetic substratum, and hope for the best. This is the traditional choice of system designers, and a great deal of effort is subsequently expended to tune the system such that the occasional catastrophic failures do not happen in typical applications.

3. Augment an inexact arithmetic substratum with specific algorithmic steps that avoid catastrophic failure and are capable of delivering valid results for all inputs. This alternative has been proposed only recently and is the subject of much current research.

Note that an exact arithmetic substratum does allow a clean separation of the various levels of abstraction. However, current technology does not have the necessary tools to make this approach attractive in the curved-surface domain, and we have to wait with this alternative until more progress has been made.

The second choice does not permit a clean separation of substratum, infrastructure, and user interface. Systems implemented in this way will fail occasionally, so the first phase of an implementation is usually followed by a second, time-consuming phase in which the system is fine-tuned to avoid failures on common inputs. Although this fine-tuning is often done by trial and error, it can be ameliorated by careful consideration of the geometric significance of each error as it is encountered. Such systems are difficult to maintain and changes to them are risky.

The third alternative, finally, is to redesign a substratum in which operations such as point/surface incidence are supplemented by processing steps that account for inaccuracies. At this time, it is hard to assess the impact this approach may have on the complexity of the algorithm, and whether the approach can achieve a strict separation of the levels of abstraction. Known complexity bounds seem to be overly pessimistic, and much further work will be needed before we can judge the ultimate utility of this approach. Research on this subject is, of course, fueled by the hope that in this way we can reach a middle ground somewhere between the expensive exact approach and the complicated and unsatisfactory traditional approach.

1.3 About This Book

This book deals primarily with the concepts and tools needed to design and implement solid-modeling systems, their infrastructures, and their substrata. Of necessity, this subject requires a considerable amount of mathematical fact and thinking. We have made every effort to make the material accessible even to the novice. The reader should be able to absorb the intuitive content without much difficulty. Going into the details may require some patience, perhaps, but should not be daunting.

Throughout the book, algorithms and the underlying theory needed to design them are in the foreground. Thus, designers, implementors, technical leaders of solid-modeling groups, and academic researchers constitute the primary audience of this book. Nevertheless, a prospective user of a solid-modeling system should read Chapters 1 and 2 to gain an appreciation of the field and of the basic concepts it exercises. Armed with these insights, he or she should then be able to assess the true capabilities of the systems under consideration. Chapter 4 provides an understanding of the finer points, and is useful when judging whether the system can provide the needed accuracy and robustness.

Chapter 2 explains the basic concepts. It discusses first the conceptual operations that one expects to find in a user interface, except for visualization and archiving operations. The presentation is kept conceptual, rather than technical, and is well suited to the casual reader who wishes to gain an overview. The chapter then presents the two dominant representation schemata used in solid modeling — namely, constructive solid geometry (CSG), and boundary representation (B-rep). Basic geometric operations in CSG are also sketched. The section on topological validity presents technical material needed for devising algorithms that test whether a given boundary representation is correct. There are other representation schemata, and they are briefly summarized at the end of Chapter 2.

In Chapter 3, we design an algorithm for Boolean operations on solids given in a boundary representation. This algorithm serves as a useful frame for developing an appreciation of the subtleties of representing and manipulating solids. Representing, analyzing, and manipulating solid models by computer is not a simple matter, and to implement competently modeling operations such as the determination of the intersection of two solids is a project of considerable complexity. For this reason, the intersection operation is discussed in depth. We restrict Chapter 3 to the intersection of polyhedra with nonmanifold boundaries. Except for the treatment of tangencies and singularities, polyhedral intersection requires dealing with virtually all aspects of this operation, so our restriction does not oversimplify the problem. The polyhedral-intersection algorithm can be used as the basis for an extension to curved solids. The added difficulties encountered relate to mathematical issues and accuracy questions that are discussed in Chapters 5 and 6.

Chapter 4 addresses the important substratum problem. There is at present much discussion as to the true origin of the problem; hence, the chap-

ter looks at different manifestations of accuracy and robustness problems and surveys a number of approaches proposed for solving them. It seems that the polyhedral case is already very difficult, and we have restricted our presentation to this case only. As already mentioned, research on this subject is relatively recent and has not yet matured, so the chapter concentrates more on making the problems intelligible rather than on giving recommendations for a "best" solution.

The balance of the book is devoted to the treatment of curved surfaces. Chapter 5 begins this topic by looking at the representational requirements. Surfaces can be represented parametrically or implicitly. Both representation styles have strengths and weaknesses. For example, for an implicitly represented surface $f(x, y, z) = 0$, the problem of testing whether a given point p lies on the surface is simple. If the same surface is represented parametrically, then this question is difficult. On the other hand, it is simple to generate points on a parametric surface. On an implicit surface, this would be much harder. Hence, we will examine methods for converting between parametric and implicit surface forms. Curve and surface singularities add specific subtleties to boundary representations of curved solids. If they are ignored, geometric ambiguities may arise that are due to the fact that, between two points in space, there might be different connecting curve segments, both belonging to the intersection of the same pair of curved surfaces. An example of this phenomenon is also discussed in Chapter 5, along with recommendations on how to avoid it.

When we are dealing with curved surfaces, the evaluation of their intersection is a fundamental operation. Chapter 6 looks at a number of techniques used to implement this operation. Specifically, by combining traditional numerical techniques with symbolic computations from algebraic geometry, it is possible to deal with complicated singularities. For plane curves, these algebraic techniques are simple and effective. Such curves could be trimming curves when intersecting parametric surfaces. For space curves presented as the intersection of two implicit surfaces, complications arise that continue to be research topics. Here, one expects that more sophisticated symbolic computations will be needed.

Since symbolic algebraic computations require a working knowledge of ideal theory, we have added a chapter on Gröbner bases techniques. These techniques include some very powerful algorithms for implicitization and inversion, and provide completely general and comprehensive methods for solving systems of algebraic equations. The great generality of the algorithms make them too slow to be of immediate routine use in production systems. However, they hold much potential for providing specializations that could play a major role in the manipulation and analysis of curved surfaces. We touch on some of these specializations in the section on basis conversion. This section also describes a strategy for implicitizing curves and surfaces that can handle problem sizes that could not be attacked successfully by other techniques known to us.

1.4 Notes and References

As A. Requicha pointed out, the term *solid modeling* is of relatively recent coinage: Early work used the term *geometric modeling* to refer to solid modeling, and reserved the term *surface modeling* to refer to work on parametric curves and surfaces.

There are several surveys dealing with solid modeling, including Requicha and Voelcker (1983). The survey by Requicha (1988) is a recent update. Voelcker, Requicha, and Conway (1988) is another survey of the area that focuses on how solid modeling could be integrated into the manufacturing process and on what the problems raised by this prospect are.

Geometric modeling in the current sense is the subject of the books by Bartels, Beatty, and Barsky (1987), Farin (1988), and Mortensen (1985). These books give good introductions to the rather large literature on the subject.

In his book on solid modeling, Mäntylä (1988) describes how to represent manifold polyhedral objects and how to implement Boolean operations on them. He also describes a facility for storing and undoing previous designs. Chiyokura (1988) describes the implementation of *Designbase*, a specific modeling system with some curved-surface capabilities. Briefly, curved solids can be designed and modified by local operations, such as altering the shape of certain edges and faces, but Boolean operations require that one of the intersecting objects be polyhedral. All objects will have manifold surfaces. Chiyokura's book also discusses several classes of parametric curves and surfaces.

The ambiguity of wireframes shown in Figure 1.1 was found by J. Shapiro at the University of Rochester.