GEOMETRIC MODELING: A First Course

1. Introduction

1.1 Preamble

Dazzling cinematic special effects... Realistic simulations of machines in motion... Analyses of the stresses in aerospace structures during flight... Programs that automatically drive robots along collision-free paths in cluttered environments... Rapid prototyping machines that behave like three-dimensional printers... Underlying these and many other applications are *geometric models* of the objects under study. Geometric models are computational (symbol) structures that capture the spatial aspects of the objects of interest for an application. This course is primarily concerned with geometric models for three-dimensional objects, and with the associated computer algorithms for constructing and querying the models.

We are interested in modeling both real (i.e., physical) and virtual objects. Virtual objects often correspond to physical objects that are being designed but have not yet been built. But they may also correspond to objects that do not obey the laws of Physics, and therefore are purely imaginary. In addition, some of the objects we encounter in the applications are themselves mathematical abstractions—for example, the set of accessible directions along which a touch probe may approach a given surface.

Geometric information is pervasive in many engineering and scientific fields, such as (i) VLSI layout, (ii) geographic information systems, (iii) electronic packaging, (iv) computer graphics and visualization, (v) computer vision, (vi) architectural and structural design, and (vii) design and manufacture of electromechanical products, to name a few. The first two examples just cited are primarily two dimensional (2-D), whereas the last two are intrinsically 3-D; examples (iii) through (v) may be either 2-D or 3-D. This course emphasizes modeling of 3-D objects, and attempts to strike a balance between applications in two areas: graphics and multimedia, and robotics and automation.

3-D computer graphics is becoming ubiquitous. Most desktop computers systems are expected to bundle support for 3-D applications in the near future, and the use of 3-D in multimedia and the World Wide Web is burgeoning. Display techniques are relatively well-developed and are implemented in hardware accelerators and in software browsers. Construction of the geometric models of the objects to be displayed, however, is becoming a bottleneck for 3-D graphics. In this text we attempt to complement, rather than compete with, Computer Graphics textbooks. We focus on modeling, and de-emphasize display and visualization, because they are well covered in the graphics texts—e.g. [Foley *et al.* 1990]. Models for free-form curves and surfaces are also treated briefly, because they constitute a large subject on their own, and are discussed in several recent texts—e.g. [Bartels *et al*.1987, Farin 1997, Rockwood & Chambers 1996]. Much of the course deals primarily with polyhedra and objects bounded by surfaces of low degree, which suffice to illustrate the main combinatorial ideas of the field and its numerical difficulties.

The intended audience are undergraduates in Computer Science or Engineering disciplines at the junior or senior level, and practicing engineers. The material corresponds to a typical semester course, and has been class-tested at the University of Southern California. Mathematical pre-requisites are the usual calculus courses, including analytic geometry. Knowledge of linear algebra is helpful, but we use little of it beyond elementary concepts such as matrix multiplication.

The sections titled Mathematical Underpinnings, which typically appear towards the ends of chapters, are written tersely and require a higher level of mathematical maturity than the remainder of the text. They are intended to provide supplementary mathematical material, and entries into the mathematical literature. They should be ignored in standard undergraduate courses. End-of-chapter sections titled Further Explorations provide a glimpse of geometric modeling material not covered in the text, with references to the literature. These sections should also be ignored in undergraduate courses. Further Explorations are not meant to be exhaustive. The geometric modeling literature has exploded over the last few years, and it is not possible to cover it completely in a text of reasonable size.

A graduate course can be designed around the material presented here by expanding the Mathematical Underpinnings sections, and adding a few advanced topics from the current literature referenced in the Further Explorations sections, or at the discretion of the instructor. Graduate versions of this course have been taught by the author at USC and earlier at the University of Rochester since the mid 1970s. The material has now become sufficiently well codified to be taught to undergraduates.

Readers are assumed to have a certain computational maturity. They should be familiar with basic programming notions such as data structures and recursion, and be proficient in C or related object-oriented programming languages such as C++ or Java. Students with good C skills can learn C++ or Java in parallel with the course material.

Learning about computation without *doing* computation is not very sensible. But interesting exercises in geometric modeling require a fair amount of software, which cannot be written within the time constraints of typical university or self-study courses. Early versions of this course were taught in Pascal, and then in C, with custom packages for vector and matrix manipulation, and for display. In the late 1990s the language used was C++ and a Geometric Tool Kit (GTK) was provided to deal with much of the low-level computation and graphical display involved in typical geometric modeling problems. Display was done by using a VRML browser. The current version uses Java and the Java 3D Application Programming Interface (API). By building upon available low-level facilities students can tackle significant problems, and construct prototype geometric modeling systems.

Programming assignments can include small graphic user interfaces to the core geometric computations that are the main subject of the course. With modern interface design tools, such as the Java Swing API, it is possible to implement attractive interfaces with a reasonable effort.

The remainder of this text is organized as follows. Sections 1.2 and 1.3 discuss the role of geometric information in two specific application domains. Section 1.4 proposes a systematic approach to the study of geometric modeling. The final section of this introductory chapter presents a historical summary of the field. Chapter 2 presents basic concepts from Euclidean and projective geometry and uses them to model motions and projections. Chapter 3 deals with fundamental issues in the computer representation of physical objects and presents the main approaches for representing object geometry. Chapter 4 addresses representations for curves and surfaces, and Chapter 5 representations

for solids. The next two chapters are devoted to algorithms. Chapter 6 presents some of the fundamental algorithms that serve as building blocks for the application algorithms discussed in Chapter 7. Finally, Chapter 8 adresses geometric modeling systems issues.

1.2 The Role of Geometry in CAD/CAM

In this section we focus on a specific application area, mechanical and electromechanical Computer-Aided Design and Manufacture (CAD/CAM), and discuss briefly the role that 3-D geometry plays in it.

Figure 1.2.1 shows the traditional organization of activities in the life-cycle of a product. Functional requirements, constraints, and optimization criteria are derived from market considerations, and serve as input to the design process. Designers generate detailed specifications for the parts and assemblies to be produced. These specifications consist primarily of geometrical information about the objects, augmented with non-geometric data such as material, hardness, and so forth. Traditionally, the geometry was defined through engineering drawings, but these have major drawbacks, as we shall discuss later, and are being replaced by the computer-based models studied in this course.

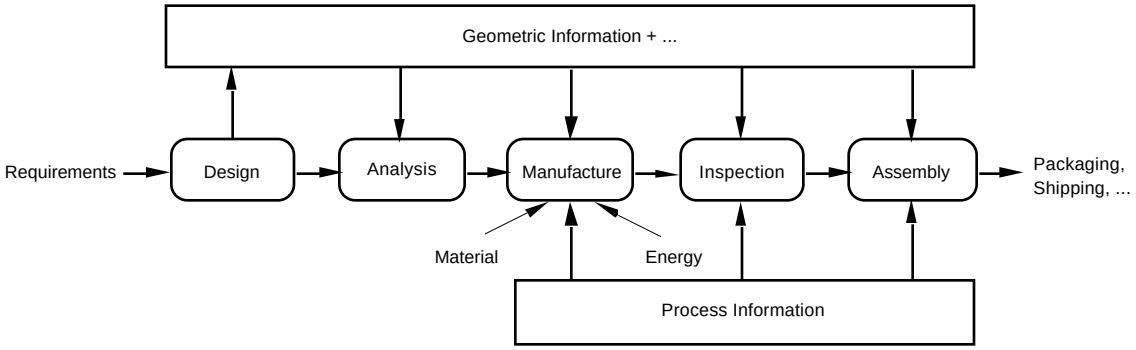

Figure 1.2.1 - Product development life cycle

The proposed designs are analyzed for compliance with the specifications, and modified if necessary. This may be done by trial and error, or by using computer-aided optimization techniques. Part specifications are passed on to manufacturing, where decisions are made about the processes to be used. These decisions take into account not only the product specifications, but also knowledge about the capabilities, costs and availability of equipment and manufacturing processes (e.g., milling, drilling, injection molding, layered fabrication). The results are plans and programs at several levels, plus *schedules* that provide timing information. High-level plans are usually called *process plans*, and consist of totally or partially-ordered sets of process descriptions. Their lower-level counterparts are sometimes called *operation plans*, and may extend all the way down to specific instructions to machine tools and robots. All the activities discussed thus far are information processing tasks. In contrast, manufacturing-plan execution is a physical process that uses energy and materials to perform the specified operations.

Manufactured parts are inspected, assembled, packaged, and shipped. Assembly and inspection are similar to manufacturing, in that process and operation plans must be generated, and these plans depend critically on part geometries and process capabilities. A

complete life cycle would also include service, maintenance, disposal, and perhaps other activities. Finally, the entire process must interface smoothly with the business and management activities of the enterprise, which are not shown in Figure 1.2.1.

The traditional design and manufacturing process just described is sequential and has severe drawbacks. For example, a designer may unnecessarily specify geometric features that are costly to manufacture. Modern manufacturing engineering espouses the principles of concurrent engineering, which have been shown to lead to better quality, lower cost, and faster time to market. In essence, all of the activities shown in Figure 1.2.1 are still performed, but in parallel, rather than sequentially. This ensures that designers have timely feedback on the downstream consequences of their design decisions. Figure 1.2.2 depicts what we call an Engineering Environment, by analogy with the Programming Environments that are commonplace in Software Engineering. Engineering Environments are well suited for supporting the concurrent engineering paradigm.

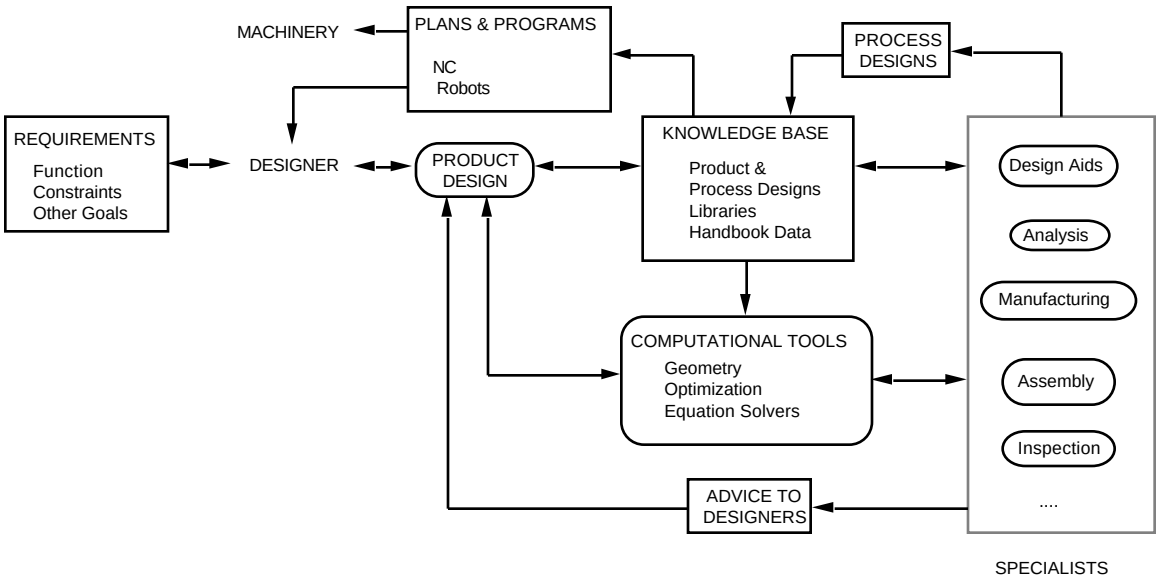

Figure 1.2.2 - Engineering Environments

A designer interacts with the product requirements, sometimes negotiating them with marketing and management. He or she also interacts with a product design subsystem through a suitable human-computer interface. Computer models for evolving designs are stored in a data base. Several specialists, which may be humans or programs, reason about the product design and produce two kinds of information: feedback to designers, and process plans and programs for driving the actual manufacturing machinery. Plans are also stored in the data base. The computer-based specialists and other computational processes shown in the figure invoke a set of fundamental computational tools of generic applicability. These include the geometric modeling tools studied in this course, plus optimization procedures, algebraic routines, constraint-maintenance subsystems, and so forth. Engineering Environments that support collaborative work, distributed in time and space, raise interesting design and research questions, and are still embryonic. System architectures based on intelligent agents offer potential solutions.

1.3 The Role of Geometry in 3-D Graphics

Computer Graphics in its early years was 2-D, and objects were defined by sequences of drawing commands. But it soon became clear that it was important to distinguish between the *model* of the object to be displayed, and the display primitives themselves. This distinction is especially important in 3-D applications. Today, a typical graphic application provides a Graphic User Interface (GUI) with which a (human) user defines *graphic models* of the objects to be displayed. The objects are then *rendered* by *image synthesis* software and hardware, producing the desired output, as shown in Figure 1.3.1. Rendering techniques and systems have evolved significantly over the past two decades, and are now capable of photo-realistic displays with texture and reflections, Virtual Reality (VR) immersive displays, and so on.

Figure 1.3.1 – Graphics modeling and rendering

Graphic models contain the shape or geometry of the objects, *i.e.*, their geometric models, but they often require additional information such as color, texture, and so on. For applications that involve animation, the temporal behavior of the objects must also be described. Our brief discussion of graphics and CAD/CAM applications (in the previous section), shows that geometric models normally must be augmented with application-specific data to be practically useful.

The geometric models needed in 3-D graphics are not always built directly by a user, as shown in Figure 1.3.1. Sometimes they are constructed by Computer Vision techniques from existing physical objects, or are acquired by medical imaging equipment such as CAT (Computer Aided Tomography) scanners, or are the result of scientific computations that simulate physical phenomena. Some rendering techniques do not even require a 3-D model, and have reasonable success in synthesizing images of 3-D scenes from a sequence of 2-D images of existing objects. This is a relatively new area of graphics, called image-based rendering.

Geometric models used in CAD/CAM must be sufficiently accurate and faithful to permit manufacture of the desired objects within tolerances, to determine if there are collisions between objects, and so on. In graphics, however, models do not necessarily have to be realistic, provided that the *images* generated from them *look* realistic to human observers. Thus, Computer Graphics can use certain modeling techniques that are not adequate for CAD/CAM. For example, natural objects such as trees, waves or clouds, are modeled by graphics-specific techniques [Foley *et al.* 1990]. In this text we focus on models that are suitable for both graphics and CAD/CAM. Discussions of graphics-specific modeling techniques can be found in graphics textbooks and in the graphics research literature, for example, in the proceedings of the annual SIGGRAPH conferences. As graphics moves towards more realistic simulations of objects in motion, it has an increasing need for collision detection and other capabilities normally associated with the CAD/CAM domain. Therefore, we are likely to see an increased overlap in modeling methodologies for these two domains in the future.

Sophisticated Virtual Environments are contributing to a more intimate connection between the Graphics and the Robotics and Automation fields through sharing of spatial reasoning tools. These are used in Robotics and Automation to accomplish tasks without a need for human intervention to plan and program the required actions. For example, to assemble two objects one must determine which approach directions can be used. This knowledge can then be used to drive a robot that assembles the physical objects. Suppose now that we want to display in a Virtual Environment a human joining two objects together. We can program the operation manually, by supplying directly to the Virtual Environment all the low-level motion information. But this is tedious, error-prone and time-consuming. High-level programming of actions in the Virtual Environment is possible if we deploy planning tools that use geometric reasoning and are similar to those developed in the Robotics and Automation field.

1.4 Models, Representations, Algorithms and Systems

Mathematics and Computer Science do not deal directly with physical objects and phenomena. They deal with *models* that capture the relevant aspects of the entities under study. We distinguish between mathematical and computational models, and usually refer to the latter as *computational representations* (or simply *representations*). The following non-geometric example illustrates the distinction. The decimal and Roman strings '115' and 'CXV' are computational representations that can easily be stored and manipulated using standard programming language constructs. They clearly represent the "same thing", but what is it that they represent? We could say that they represent physical entities such as collections of pebbles, but then we would have to explain that the color of the pebbles did not matter, nor did their material, nor did the fact that they were pebbles, and so forth. It is more reasonable to say that the strings represent *natural numbers*, which are abstract mathematical entities that capture the aspects of reality relevant to counting. Mathematics provides us with a rich theory of natural numbers that we can use to study the properties of decimal and Roman string representations and associated computations.

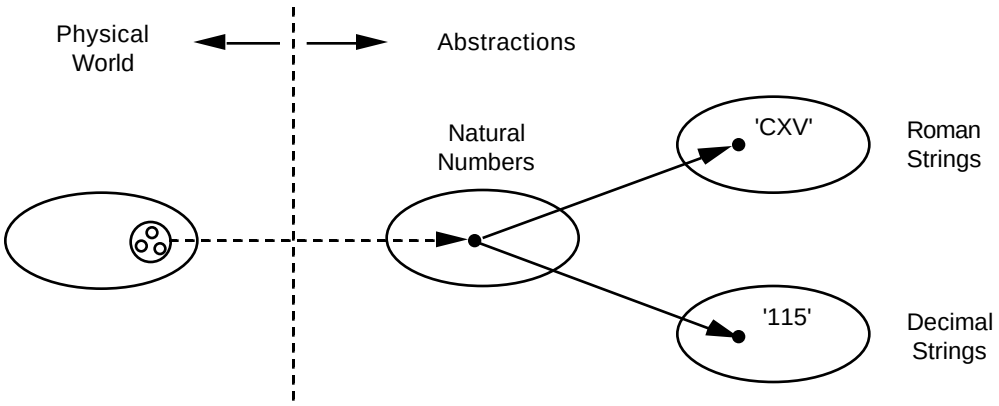

Figure 1.4.1 - Physical entities, mathematical models and representations

The ultimate validity of a model of a physical entity must be ascertained experimentally. The models must be able to predict the behavior of the corresponding physical entities. The results of measurements performed in the real word—the answers A in Figure 1.4.2—must agree with the values predicted by the model—the answers A_m in the figure. Centuries of experimentation have shown that natural numbers are good mathematical models. Similarly, the 3-D Euclidean space (E^3) of analytic geometry has proven to be an excellent

model for the spatial aspects of reality, provided that we do not study phenomena at the galactic spatial scale (which requires the curved spaces of general relativity) or involving velocities comparable to the speed of light (which fall under the purview of special relativity).

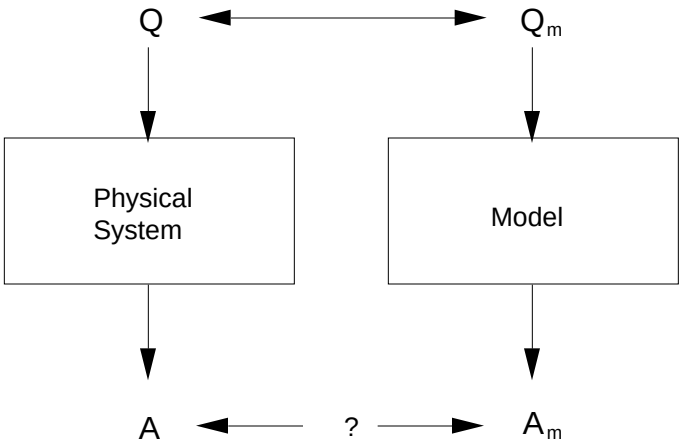

Figure 1.4.2 - Models and reality

Let us turn now to a simple geometric example. Suppose that we press the tip of a sharp pencil on the top of a desk and make a mark. We can easily conceive a "perfectly sharp" pencil whose tip has zero diameter, and which produces a mark also of zero diameter. This situation may be modeled mathematically as follows. The desktop is modeled by a subset of 2-D Euclidean space (E^2), and the tip by a point in that space. If we move the tip on the desktop we can make another mark, which corresponds to another point. Mathematics lets us pose well-defined questions such as "what is the distance between the two points?". The mathematical theory of Euclidean spaces has been developed with great rigor and we can rely upon its theorems and methods.

If we want to compute the distance between two points we need means to designate the points unambiguously. This is the role played by representations. A point in E^2 may be represented by two real numbers, its x and y coordinates measured with respect to an agreed set of axes, corresponding, for example, to two orthogonal sides of the desk top. Computationally we can use a pair (a 2-element array) of floating point numbers as the representation for the point. (Note, however, that floating point numbers must have finite exponents and mantissas and therefore cannot represent *all* the real numbers; this is the source of many numerical robustness problems in geometric modeling.) With this representation, the distance can be computed by a very simple algorithm that produces a non-negative floating point number by evaluating the familiar expression from analytic geometry involving the square root of a sum of squares.

The methodology embodied in this desktop example may be summarized and generalized with the help of Figure 1.4.3. There are objects of interest in the physical world, such as a desktop and pencil marks. One often needs to answer questions about properties of these objects—for example: "how far apart are two pencil marks?". To do this we first replace the objects by abstractions which we call *mathematical models*. In our example, the desktop becomes a subset of the Euclidean plane, and the pencil marks are modeled by points. Observe that modeling involves abstraction, and ignores many aspects of reality, which are judged irrelevant for the issues to be studied. For example, the Euclidean model of the desktop captures its geometric or spatial aspects, but says nothing about many macroscopic

properties of the desk such as its material and color. In addition, the atomic nature of the desk is entirely ignored. This implies that each model has a specific domain of applicability. For example, the E^2 model for the desktop is appropriate for studying the stability of solid objects lying on a table, but is clearly unsuitable for dealing with robotic operations at the nanometer scale that move individual atoms or molecules at the solid/gas interface between table and air.

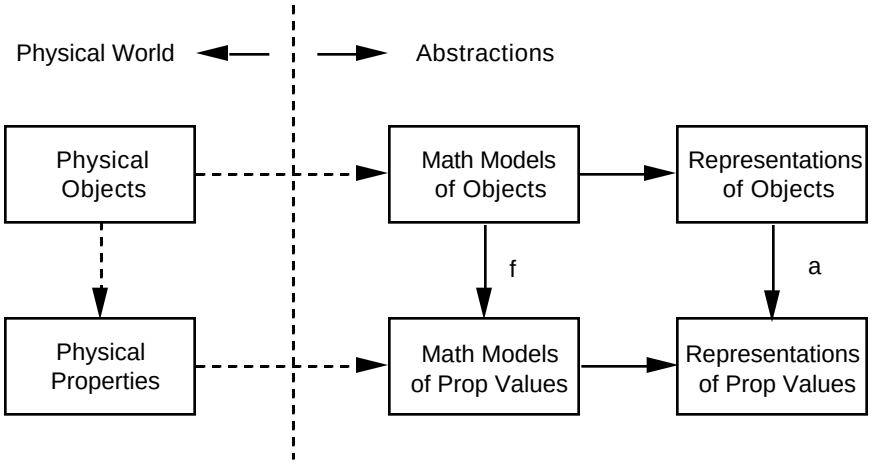

Figure 1.4.3 - A systematic view of modeling

Object properties are modeled by mathematical functions, denoted by *f* in Figure 1.4.3, which map (mathematical models of) objects into the values of their properties. Thus, physical "nearness" is measured by the distance between points, which is a mathematical function that maps two points onto a non-negative real number.

Computation requires a further step. Mathematical models must be associated with corresponding symbol structures, i.e. *representations*, that can be constructed within computers. Thus, we need representations for objects, and for the values of their properties. In our example, points on the plane were represented by pairs of floating point numbers, and distance values were represented by single positive floating point numbers. We also need *algorithms* for doing the actual computation, i.e., for converting the input representations into their output counterparts. In our example, the distance computation evaluates a simple expression containing the coordinates of the input points. Algorithms are sets of computer-intelligible instructions, usually arranged sequentially. Therefore algorithms are *not* mathematical functions. We often say that they *implement* functions, because correct algorithms produce the (representations of the) values of the corresponding mathematical functions.

Our discussions of representations and algorithms, and of Engineering Environments, suggest the following high-level architecture of a generic system for geometric computation—see Figure 1.4.4. The essential components are (i) geometric models, i.e., representations for geometric objects, (ii) algorithmic processes that use such representations to answer geometric queries, such as "what is the distance between two points?", (iii) input facilities for creating and editing object representations, and for invoking processes, and (iv) output facilities and representations for results. The subsystem that provides facilities for entering, storing, and modifying object representations is usually called a *geometric modeler* or *geometric modeling system*.

Sometimes a geometric modeler is more broadly defined to include some of the facilities, e.g. graphic rendering, that are needed in most applications.

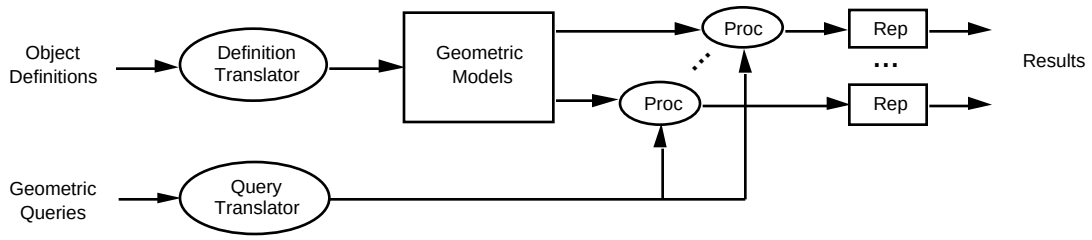

Figure 1.4.4 - A generic system for geometric computation

This course covers all the components shown in Figure 1.4.4, with a special focus on representations and algorithms. We distinguish between fundamental and application-specific algorithms. Fundamental algorithms are useful for constructing and maintaining representations of geometric objects, and for a range of applications. They include intersection computations, distances, and so on. Application algorithms are relevant primarily to specific applications such as graphics or the computation of volumes. They often use fundamental algorithms to perform lower-level calculations.

The mathematics that underlies much of the material in this course is discussed very briefly, mainly in the sections titled “Mathematical Underpinnings”. A more throrough discussion is appropriate only at the graduate level, because it involves a variety of non-trivial subjects, ranging from general and algebraic topology to differential and algebraic geometry.

1.5 Historical Summary

This section provides a brief, non-exhaustive summary of the historical development of geometric modeling. Articles by many of the pioneers of the field appear in [Piegl 1993], and contain references to the work cited below. A series of survey articles also contain many references and trace the history of the field [Requicha & Voelcker 1982, 1983; Requicha 1988; Requicha & Rossignac 1992].

The beginnings of geometric modeling can be traced to the 1950s, when several related technologies were launched. Computer graphics saw the development of Sketchpad in Sutherland’s Ph.D. thesis at MIT, and of the DAC-1 system at General Motors. Sculptured, or free-form curves and surfaces were introduced at MIT by Coons and in France by Bezier and de Casteljau. The APT programming language for numerically-controlled (NC) machining was initially developed by Ross’ group at MIT. Work on modeling of solid objects included Robert’s thesis on modeling for vision at MIT, Gutterman proposals for a solid modeling system at Sandia, and the research of Luh and Krolak at IBM.

From this spate of initial activity emerged four main streams of work that evolved largely independently for some two or three decades. The *computer graphics* stream focused on rendering and interaction. Initially, graphics was 2-D and the notion of a model was not acknowledged explicitly. The models were essentially the display lists used to drive the hardware. Later on, computer graphics’ models became object approximations through

unstructured nets of polygons. Today, polygonal models are still prevalent, and associated rendering algorithms and hardware are well developed and capable of producing highly realistic results.

The *wireframe* stream led to the commercial CAD systems of the 1970s and 80s. Initially these systems were simple drafting aids that represented objects by 2-D views composed of lines and arcs. This had unpleasant consequences. For example, when a view was modified the changes were not reflected in the others. Later, wireframe systems adopted representations consisting of unstructured collections of curve segments (i.e. *edges*) in 3-D. As we shall see later, wireframes do not designate solid objects unambiguously, and cannot guarantee the validity of their data.

The *free-form curve and surface* stream found important applications in computer-aided design and manufacture of car bodies, aircraft fuselages and in other tasks in the automotive and aerospace industries. The technology evolved towards B-spline representations (discussed later in this course), which were pioneered by the University of Utah group led by Riesenfeld and Barnhill. (Barnhill moved later to Arizona State University.) NURBS (Non-Uniform Rational B-Splines) are becoming a de-facto standard for free-form surface representation. Free-form surfaces are increasingly being used in animation and other applications in the entertainment industry. Curve and surface modeling is sometimes called Computer-Aided Geometric Design, and has strong ties with numerical analysis and differential geometry.

Solid modeling is distinguished by the use of unambiguous representations for solids. The initial forays cited above were ambitious but met with limited success, and were abandoned. The field resumed progress in the early 1970s when a flurry of activity took place in most of the industrialized countries. Braid's Ph.D. thesis at the University of Cambridge in England introduced the BUILD system. Braid and co-workers became a major force in the commercial solid modeling arena, and were responsible for the ROMULUS, Parasolid and ACIS modelers, the last two of which are still actively being developed. In Germany, work proceeded in the Compac system at the University of Berlin, and in Proren at the University of the Ruhr. Brun's Euclid system was launched in France and Engeli's Euklid in Switzerland. In Japan, Okino developed TIPS-1 at Hokkaido University, and Hosaka developed GeoMap at the University of Tokyo. In the U.S. two systems were built outside academia: the Shapes system at the Draper Labs, and the commercial Synthavision system, which had been developed initially for studying ballistic and nuclear effects and was adapted for CAD applications. In American universities Eastman's group at Carnegie-Mellon produced the GLIDE system with emphasis on architectural applications and data base issues. (Eastman moved to Georgia Tech in the 90s.) At Stanford, Baumgart's system, which was intended primarily for Computer Vision, introduced data structures that were influential in later developments. Finally, the Production Automation Project at the University of Rochester established much of the theoretical foundation for the field, and developed the PADL systems. These were disseminated widely, and several commercial systems were built upon them.

A related, fifth stream emerged in the early 1970s with Shamos' Ph. D. thesis. This stream focuses on the theoretical aspects of design and analysis of geometric algorithms, and has become known as Computational Geometry. (The term had been coined earlier, in the 1960s, by Forrest in England, and also by Minsky at MIT, with a broader meaning.) Computational Geometry has dealt primarily with theoretical problems involving polygonal and polyhedral objects, often in 2-D, but is moving towards a broader domain, with an increased emphasis on applications.

Finally, spatial reasoning and other geometric aspects of Robotics may be considered a sixth related stream, which also has evolved independently, although it has strong intellectual ties with solid modeling and CAD/CAM.

In the late 1990s the computer graphics, sculptured surfaces, solid modeling, computational geometry, and robotics research communities were still largely distinct, published in separate journals and attended different conferences. However, a convergence of all these different aspects of Geometric Computation is becoming evident, and new systems use ideas from all of these subfields.

Geometric modeling technology has been slow to gain acceptance, because it requires substantial computational resources, which have not been available at low cost until very recently, and because it often has significant impact on how enterprises are organized and managed. 3-D graphics exploded into the marketplace in the second half of the 1990s, much like color 2-D graphics and windowing systems spread earlier in the decade. Many of the ideas in Sketchpad, e.g. constraint-based design, are coming into widespread use only now. Spline technology is now blossoming in many graphic packages for the personal computer market. Solid modeling is gaining acceptance steadily, but there are still many companies that use simpler wireframe systems or even manual drafting.