THE IMPACT OF ROBOTICS ON COMPUTER SCIENCE

The rapid development of robotics and the resulting need for computer scientists to be better trained in traditional mathematics necessitate changes in computer science curricula.

JOHN E. HOPCROFT

Computer science began as a discipline about 1965. Its first 20 years were introspective, as it developed areas internal to the science itself. The typical curriculum at a university included data structures, algorithms, the design of programming languages, environments, operating systems, theory of computation, and the like. Having established itself as a science, it is now blossoming and reaching out into application areas including graphics, animation, electronic prototyping, simulation, and robotics. Computer science is destined to play a major role in numerous application areas. It is important to examine the consequences of this changing nature, particularly in regard to curriculum in universities.

Since Marvin Denicoff was an early supporter of robotics, it seems fitting to select robotics as a vehicle for exploring the likely impact of applications on computer science. Research in robotics today consists primarily of work in instrumentation, sensors, control theory, and robot design. These are important engineering problems, and research on them will lead to improvements in the cost of manufacturing and in the quality of finished products. However, the improvements, although significant, will not result in a marked change in our way of life.

There is a more general view of robotics as the study of representing, manipulating, and reasoning about physical objects in a computer [3]. This view is concerned with issues of languages and representation, and it is precisely in these areas, which traditionally come under the auspices of computer science, that significant breakthroughs will occur that will have as fundamental an effect on society as did the industrial revolution. Not only will computer science play a major role in these areas of robotics, but robotics and other application areas will have a major impact on computer science. If indeed computer science departments branch out into such areas, then it is important to examine what the effect of this will be on computer science.

Perhaps the most significant change will be in the prerequisite mathematics required of students, and it is in this area that I will focus. During the past 20 years, computer scientists have urged mathematics departments to teach more discrete mathematics. The needs of computer science have been primarily in logic, graph theory, combinatorics, and number theory—areas dependent on discrete mathematics. The robotics field, however, suggests that future needs will include much more traditional branches of mathematics. In this article, I will illustrate what I view as the computer science side of robotics and the mathematics that will need to be incorporated into the computer science curriculum in order to support research in this area.

Computer science has traditionally dealt with abstractions. The ability to represent abstract concepts symbolically promises to substantially modify previously established ways of thinking about design and assemblies. Traditionally, the engineer has thought of objects as being determined by their size and shape. However, intrinsic to an object being designed is its functionality and not its size or shape. Consequently, the potential to represent objects by

This article is based in part on research supported by the National Science Foundation under grants ECS 83-12096 and DCR 8502568.

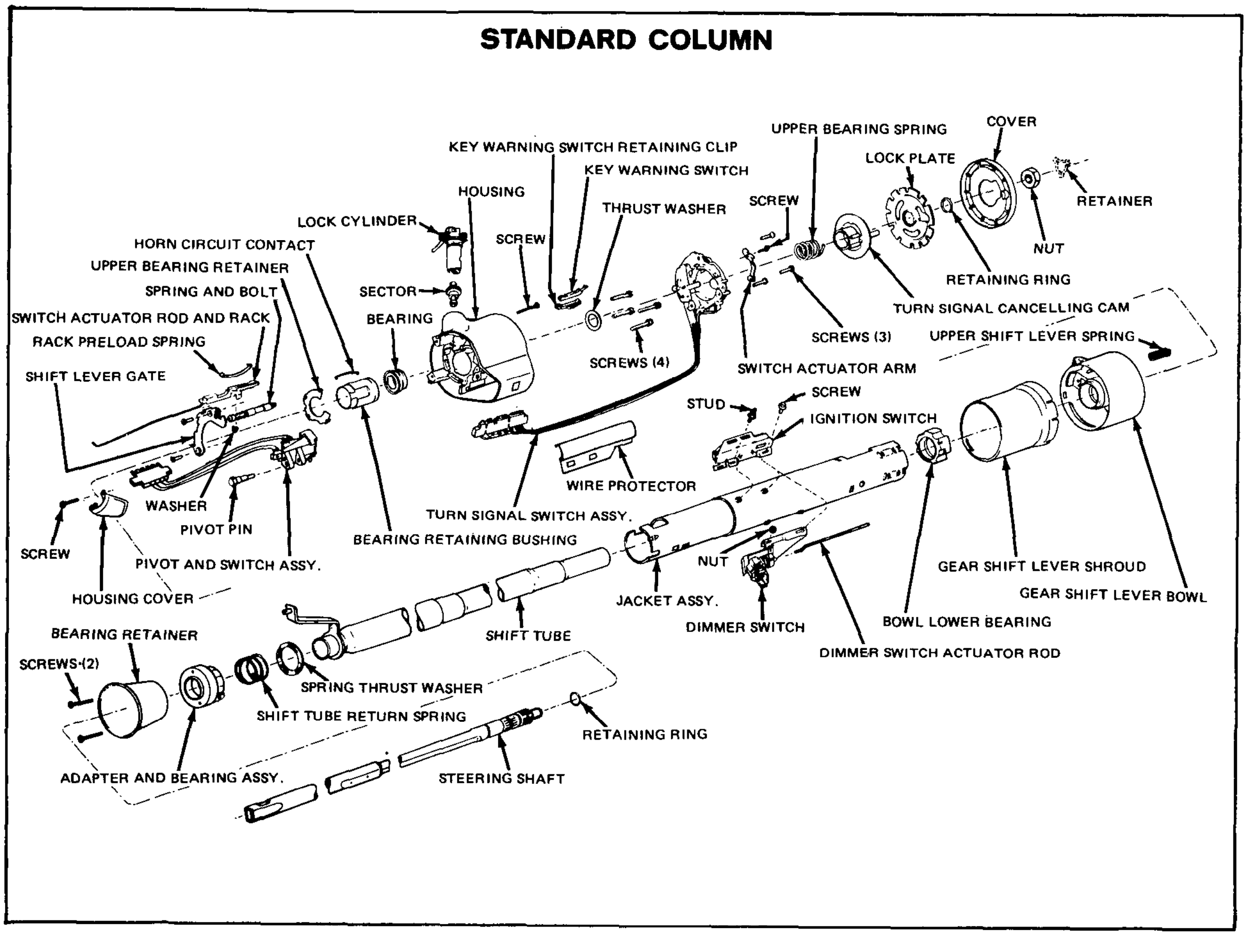
their functionality leads to entirely new design methodologies. Computers have brought to the design process the capacity to design objects in three dimensions. With the sole exception of the sculptor, we are now able, for the first time in history, with the aid of solid modeling, to utilize all three dimensions.

To explore these topics, consider a typical assembly. Figure 1 is an exploded view of the steering column for a General Motors Citation. Imagine trying to write the computer program for a robot to automatically assemble such an intricate steering column. At today's level of programming, one would use either a teach pendant and direct the robot through the desired sequence of moves, or attempt to code the necessary steps in a language such as VAL®. In either case, programming the robot would be a tedious task. This prompts one to look at the benefits of model-driven programming. Envision a system that would compile an exploded diagram from a database of parts descriptions and simple assembly tasks, as in Figure 1, into the code needed to drive the robot. This feat is within today's know-how and in fact exists in much simpler domains. Restricting the universe to the toy world of Lincoln Log® cabins makes the task quite easy, and there is in existence today a program that can construct a Lincoln Log cabin from such an exploded diagram. The exploded diagram is constructed on a graphics terminal from a database of predefined logs, and then the physical logs are placed in a parts feeder from which the robot grasps the logs and assembles the cabin. This establishes that, in principle, computer interfaces can be as user friendly as the exploded diagram description of an assembly.

One of the important capabilities of any such sys-

VAL is a registered trademark of Unimation, Inc.

Lincoln Log is a trademark of PLAYSCHOOL, Inc., a Milton Bradley Company.



Courtesy: Chevrolet Motor Division

FIGURE 1. Exploded Diagram of an Assembly

tem is the ability to model individual parts and physical assemblies. This opens up the whole new world of electronic prototyping. A manufacturer designing a computer constructs a physical prototype before committing to production, simply to verify certain aspects of the design—for example, no two parts occupy the same physical location. The risks of an error in trying to verify such aspects from a two-dimensional design are sufficiently high that manufacturers go to the expense of building physical prototypes. Using model-driven programming, it should be possible to replace physical prototypes with electronic prototypes and thus reap the benefits. With the computer model, one can not only verify the physical locations of parts, but one can also carry out other engineering analyses such as heat flow or stress analysis. In addition, electronic prototyping permits changes much later in the design cycle. A change in design only requires rerunning the design verification programs and avoids going back to rebuild the physical prototype. One can even imagine prototyping objects for which physical models are impossible, such as a satellite antenna that will not support its own weight under gravity. If the satellite is launched in space and the antenna does not deploy properly, one can test various hypotheses in parallel by using multiple copies of the computer model, whereas a physical prototype would necessitate sequential testing and the avoidance of any situation that could destroy the model.

The complexity of the objects we wish to model raises the issue of economics. Consider a model of an internal combustion engine. In order to justify the time and effort involved in constructing such a model, it would be necessary for the model to support the entire range of engineering activities involved in the design, analysis, and assembly of the engine. This means that one must have the capability to easily lift the cylinder and a piston from the database, possibly by pointing at them with a light pen, and then construct a finite-element mesh for the cylinder in order to study the combustion properties. From this example alone, it is clear that a wide range of activities must be supported in any such engineering decision.

The level of modeling capability just described would open up vast new opportunities in a number of different fields. Outside of the mechanical parts area, for example, simulating the dynamics of mechanical systems could be used to test synthesis procedures for organic molecules. If a promising procedure were discovered, it could then be tested in the laboratory to see if it would indeed work. In this way, thousands of synthesis procedures could be examined and reduced to a set small enough for laboratory testing.

In the medical field, the design of an artificial knee could be tested by modeling a patient in some reasonable range of human activities like sitting in a chair, walking up stairs, or stepping off a curb, to verify that the knee would function properly for that specific person.

In designing VLSI circuitry, the effect of misregistration between the different levels of materials could be simulated by building up the actual geometric structures of the chip, and so the increased delay time of the chip could be calculated. By relating the increase in delay time due to misregistration, the necessary accuracy in registration of the various levels could be determined, thereby reducing the cost of chip foundries by not overdesigning for tolerance.

Robotics is only the tip of a much larger field. The more general field is concerned with representing, manipulating, and reasoning about physical objects. For lack of a better name, I call this field *stereo phenomenology—stereos* being the Greek word for solid, and *phainomenon* and *logos* being the Greek words for representation and theory—that is, theory of representation. Via stereo phenomenology, robotics will have a significant impact on computer science.

Now that we have seen examples of how this new technology could be used, let us examine what effect such applications will have on computer science and what kind of education will be necessary to support these activities. In a research-oriented university, a delay time is involved in the transfer of ideas from university research laboratories to industrial practice. The major transfer of information from universities to industry does not occur through journal articles and publications; rather it comes about through students who get degrees and then take jobs in industry, carrying with them a certain amount of information. Given that students beginning their studies today will not get degrees for another four years and will not be in a position to affect the ways in which things are done in industry for possibly another 10–15 years, the time delay is on the order of 20 years. We must therefore look ahead as far as possible and try to educate students now for the rapidly changing world of 20 years hence.

To see just what is involved, consider the construction of the mechanical valve computer model (Figure 2) shown in Figure 4. This exercise in modeling an already designed part makes two things clear. First, there is a large amount of dimensioning information that makes entering the design into the computer a time-consuming chore. Second, there are a

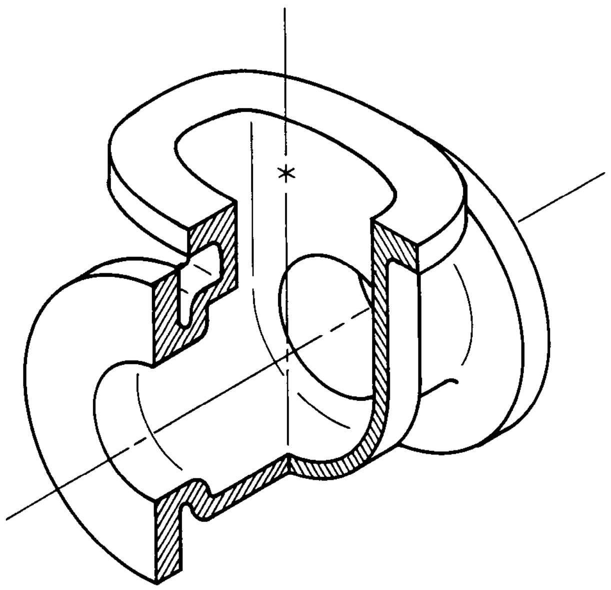


FIGURE 2. Model of a Valve Body

number of blending surfaces that have little functional role other than to smoothly connect two other surfaces. Frequently two different manufacturers given the design would produce two different products that vary in the precise shape of blending surfaces. Unfortunately, these surfaces are more mathematically complex than the surfaces they join. I estimate that 95 percent of the effort in entering the design was devoted to these surfaces. An inordinate amount of time was spent on details of the construction with which we were not very facile. To simplify model construction, we had to automate these aspects.

Our first step was to create a textual language for describing parts. Approximately one-third of the description of the valve is given in Figure 3. Looking at the description, one notices two things: Only the first four lines of the definition of the pipe at the top of the figure are needed to specify the pipe, which is defined as the difference of two cylinders truncated by two half planes. The remaining lines of the definition name the features of the pipe so that we can

```
pipe(radius, thickness, length) := (cyl1 − cyl2) ∩ H1 ∩ H2 where
  begin
    cyl1 := ycylinder(radius + thickness);
    cyl2 := ycylinder(radius);
    H1 := {y ≥ 0};
    H2 := {y ≤ length};
    top := (cyl1 − cyl2) ∩ {y = length};
    bottom := (cyl1 − cyl2) ∩ {y = 0};
    outside := cyl1.surface ∩ H1 ∩ H2;
    top.in_edge := top ∩ cyl2;
    bottom.in_edge := bottom ∩ cyl2
  end;

flanged_pipe(radius, thickness, length) :=
    smooth_attach(nipple.top.in_edge, sflange.bottom.in_edge, 1/8) where
  begin
    ft := 3 × thickness + hole_diameter;
    flange := pipe(radius, ft, fl);
    sflange := smooth(flange.outside, flange.bottom, 1/16);
    nipple := pipe(radius, thickness, length);
    bottom := nipple.bottom;
    top := sflange.top
  end;

ovoid(r1, r2, r3) := smooth(cyl1 ∩ cyl2 ∩ cyl3, r3) where
  begin
    cyl1 := ymove(−(4 + 9/16), zcylinder(r2));
    cyl2 := ymove(4 + 9/16, zcylinder(r2));
    cyl3 := zcylinder(r1);
  end;
```

The text is approximately one-third of the definition of the valve shown in Figure 4.

FIGURE 3. Textual Language Description of the Valve

refer to them symbolically. This simplifies combining the pieces. To attach Part *A* to Part *B*, simply indicate the features to be aligned, and let the computer deal with the various coordinate systems. This dispenses with much of the difficulty in dealing with numerical information.

The second step was to develop combining operations such as smooth attach, for example, in the definition of the flanged pipe in Figure 3. Rather than construct the blending surfaces by hand, a mathematical theory of blending surfaces was developed that allowed them to be generated automatically. The result is shown in Figure 4, which was generated from the computer model of the valve constructed from the description in Figure 3. The blue surfaces are parts of the valve that would be supplied automatically. Using the computer to construct these surfaces substantially reduces the time and effort it takes to build the model.

Underlying this design automation is a theory for blending the intersection of any two quadric surfaces. Figure 5 shows a number of pairs of surfaces and the blends obtained. Part (a) is two circular cylinders intersecting at right angles. The cylinders are shown in red, and the blend is shown in blue. Part (b) is two ellipses, one cut from the interior of the other. Part (c) shows two circular cylinders whose surfaces share a common line. Part (d) shows the blend of a sphere and a hyperbolic paraboloid. In this case, the primary surfaces actually do not even intersect.

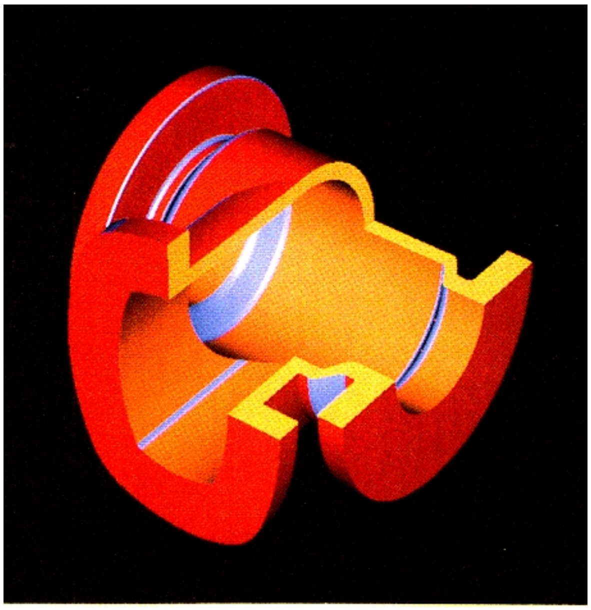


FIGURE 4. Computer-Generated Image of the Valve

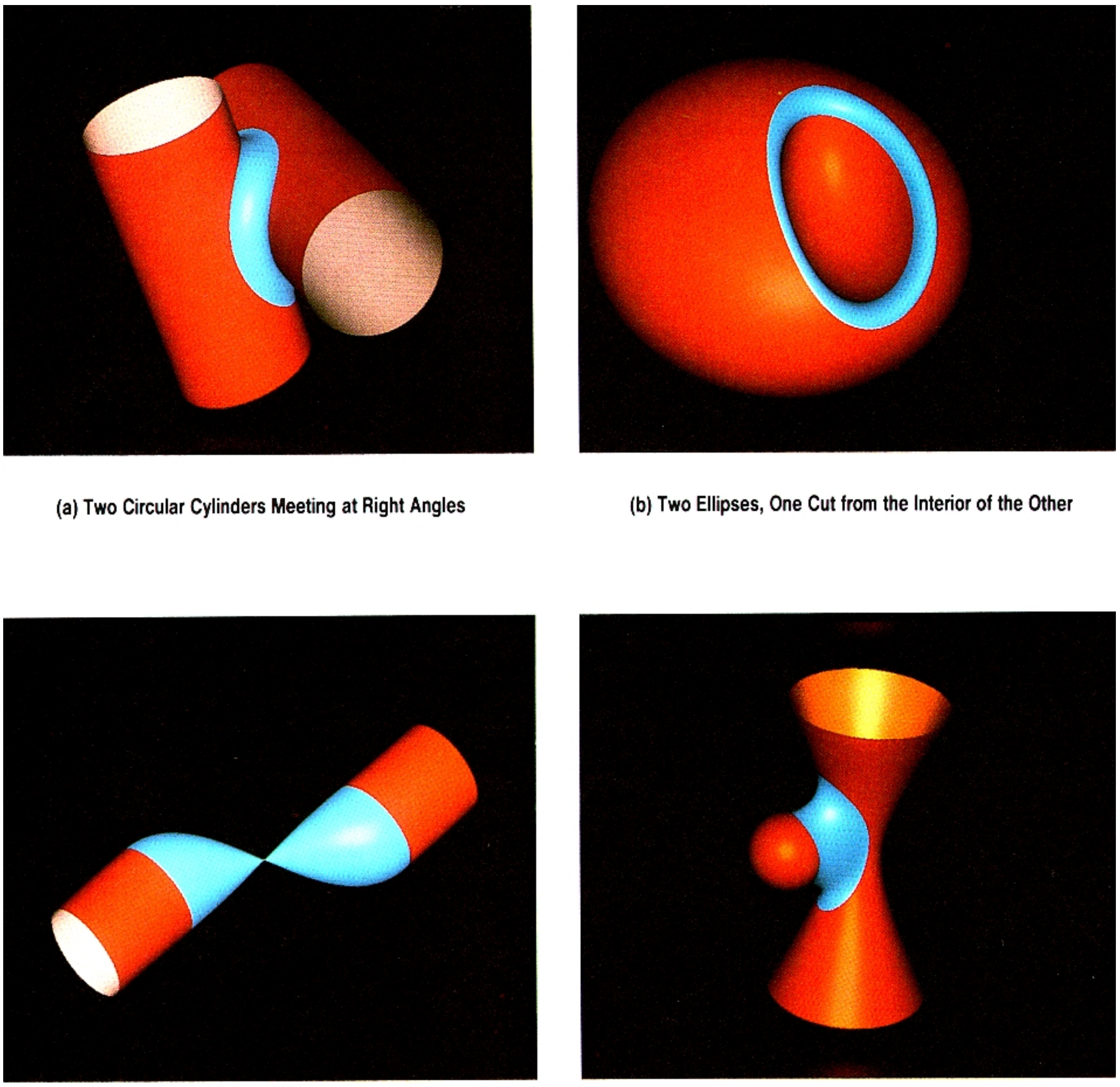
Although the mathematical theory is valid for any pair of intersecting quadric surfaces, it may not give the blend that is expected. If the pipe in Figure 5c were intended for fluid flow, then obviously the blend would not be satisfactory. One of the major difficulties in design automation is how to capture the intended use in the mathematical formulation of the problem.

How are the blending surfaces obtained? Consider the two intersecting cylinders shown in Figure 6, page 492, and think of expanding the radius of the vertical cylinder. The expanded vertical cylinder intersects the horizontal cylinder in some degree-four curve. Now reduce the radius of the vertical cylinder to its original size while simultaneously increasing the radius of the horizontal cylinder. The intersection of the two cylinders is always a space curve. As the radii vary, the space curve sweeps out a surface. Appropriately relating the rate at which the radii vary yields a quartic surface tangent to each of the cylinders. This is called the potential method [1, 2] and requires only freshman calculus.

There is, however, a need for a theory to answer questions such as, Do other blending surfaces exist? From the theory of ideals, we know that, under certain technical conditions, the ideal generated by the equations of two surfaces, *G* and *H*, consists of all equations of surfaces that contain the intersection of *G* and *H*. From an understanding of this particular theory, we are able to determine the class of all surfaces tangent to the two surfaces in given curves of tangency, and to characterize all possible curves of tangency. In the case of blending two quadrics with a fourth-degree surface, the potential method is completely general, in that the class of surfaces that it produces contains all possible blending surfaces. Furthermore, the theory of ideals unifies a substantial body of existing work and shows that methods thought to be different actually give rise to the same class of surfaces. The theory also provides certain blends not previously known to exist.

Although we have discussed automated design in terms of supplying blending surfaces, the area is much broader. Imagine designing a crankshaft that must satisfy certain global properties such as balance about various axes. Portions of the crankshaft whose shapes are not functionally determined can be automatically designed to achieve the desired balance. Implementing systems of this nature requires substantial knowledge of rings and ideals. But today's curriculum provides the computer scientist with very little exposure to these topics in classical algebra.

One design principle is never to start from scratch. In designing a mechanical part, always go back and



(a) Two Circular Cylinders Meeting at Right Angles

(b) Two Ellipses, One Cut from the Interior of the Other

(c) Two Circular Cylinders Sharing a Common Line

(d) A Sphere and a Hyperbolic Paraboloid

FIGURE 5. Blending Surfaces

find a previous design and modify it to get the desired design. We do this with computer programs by taking a previously written program and editing it. For computer-aided design to be successful, we must have editors that allow easy modification of three-dimensional objects. One difficulty in constructing such editors is shown in Figure 7, page 492. Suppose a vertex of (a) is moved. What would happen? One possibility would be the elongation shown in (b) or the deformation shown in (c). What actually happens depends on the internal structure supplied by representations of the object.

Figure 8, page 493, illustrates two definitions of the cube. The first definition defines the cube as the intersection of three slices. A vertex v of the cube is defined as the intersection of the front, right, and top faces. Given this definition, moving a vertex requires the program to readjust the front, right, and top faces so that their intersection is the new vertex location. This is done by modifying the width of the

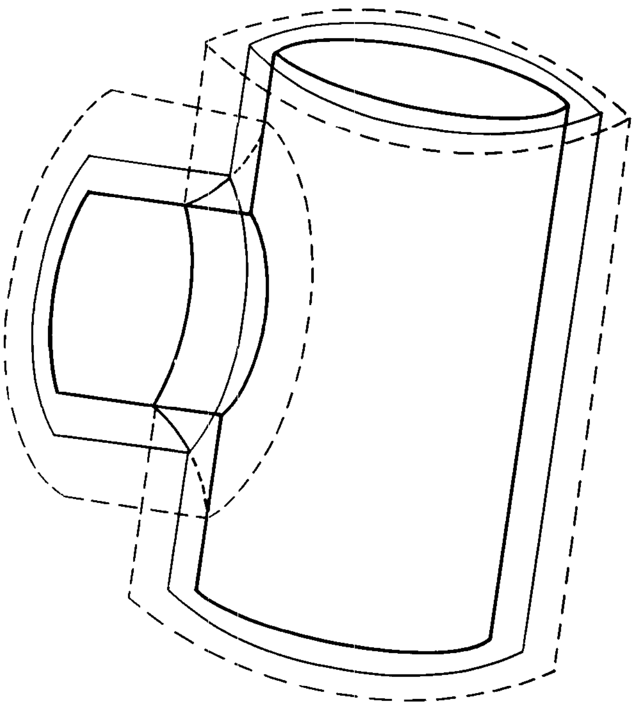


FIGURE 6. Illustration of a Potential Method for Constructing a Blending Surface

three slices, producing the object in Figure 7b. If, on the other hand, the cube was defined in terms of vertices, edges defined as lines connecting vertices, and faces defined as patches with the edges as boundaries, then moving the vertex would modify the edges and hence the faces, producing the object in Figure 7c. Returning to the valve example, one would like to edit the three-dimensional model as conveniently as one edits programs or text, always being assured that the result will be a legitimate physical object.

Another aspect of modeling is the selection of internal representations. There are several possible representations, for example, of an edge formed by the intersection of two surfaces of an object. One way would be to represent the edge as the intersection of two surfaces. In systems where this is done, numerical methods are usually used to crawl along edges in the various algorithms for manipulating or displaying objects. Another method, which has very desirable features, is to use a parametric representation for the edges. Figure 9, page 494, shows a semicircle and a parametric representation in terms of rational functions. This representation raises the question as to which curves in three space have rational parametric representations. It turns out that the class is very limited. Even the mechanical parts area, with surfaces limited to quadrics, gives rise to curves that do not possess rational parameterizations.

Consider the intersection of two quadric surfaces as shown in Figure 10, page 494. In general, the curve will be an irreducible fourth-degree space curve, in which case there is no rational parameterization. Thus, there is a need to know the types of parameterizations that exist for various classes of space curves. To understand the type of algebraic geometry [9] needed to answer these questions, it is constructive to look at the proof that there is no rational parameterization for the intersection of two quadrics when the intersection is an irreducible degree-four curve. Consider a curve in the plane and

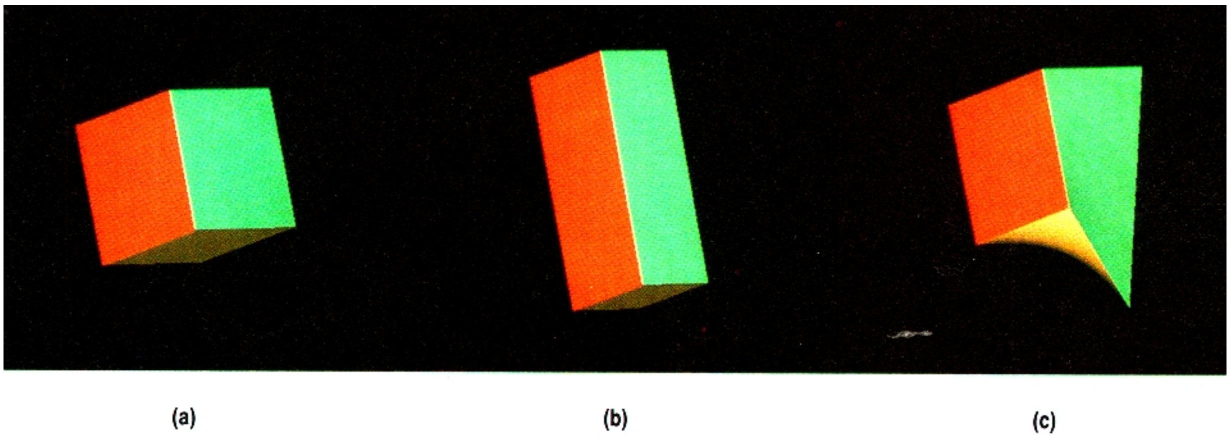


FIGURE 7. Editing a Cube by Moving a Vertex

Definition 1

```
x-slice(width) := H1 ∩ H2 where
  begin
    H1 := {x ≥ 0};
    H2 := {x ≤ width};
    left := {x = 0};
    right := {x = width}
  end;

y-slice(length) := H1 ∩ H2 where
  begin
    H1 := {y ≥ 0};
    H2 := {y ≤ length};
    front := H2;
    back := H1
  end;

z-slice(height) := H1 ∩ H2 where
  begin
    H1 := {z ≥ 0};
    H2 := {z ≤ height};
    top := H1;
    bottom := H2
  end;

cuboid(length, height, width) :=
    x-slice(width) ∩ y-slice(length) ∩ z-slice(height) where
  begin
    front.right_edge := front ∩ top;
    front.right.top_vertex := front ∩ right ∩ top
  end;
```

Definition 2

```
begin
  vertex1 := (x0, y0, z0)
  vertex2 := (x1, y1, z1)

   ...
  edge1 := (vertex1, vertex2);
  edge2 := (vertex1, vertex3);

   ...
  face1 := patch(v1, v2, v3, v4);

   ...
end;
```

Definition 1 corresponds to the cube in Figure 7b. Definition 2 corresponds to the cube in Figure 7a.

FIGURE 8. Two Definitions of a Cube

a set of points on the curve. To each point in the set, assign a positive integer. Consider the set of all rational functions that intersect the curve only at these points, and for which the order of the intersection at each point is less than or equal to the integer assigned there. The Riemann–Roch theorem states that the set of all such functions, called a linear series, is a vector space, and relates the dimension of the vector space to the sum of the integers assigned to the points, the genus of the curve, and the index of specialty of the points (see Figure 11, page 495). It is this particular theorem that provides a proof that the irreducible degree-four curves of intersection of quadrics have no rational parameterizations.

Project the curve of intersection of the two quadric surfaces from a point on the curve onto a plane. The result is a cubic curve. If a line intersected the projected curve in *four* points, then a plane contain-

$$x = \frac{2t}{1+t^2}$$

$$y = \frac{1-t^2}{1+t^2}$$

$$x^2 + y^2 = \frac{4t^2}{(1+t^2)^2} + \frac{1-2t^2+t^4}{(1+t^2)^2} = 1$$

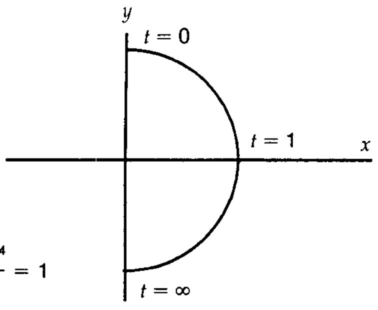


FIGURE 9. Parametric Representation for a Semicircular Edge

ing that line and the projection point would intersect the curve of intersection in *five* points, which is impossible by Bezout's theorem. This theorem's geometric interpretation is that three surfaces expressible by algebraic equations cannot meet each other in more points than there are units in the product of the degrees of the equations. Furthermore, not only is the projection a cubic curve, but it has no double points. To see this, suppose a line l contains three points of the intersection of the two surfaces. Then the line l would intersect each surface in three points. By Bezout's theorem, each surface would contain the line, and hence the line would be contained in the intersection. This would imply that the curve of intersection is reducible. Thus, the projected curve is a nonsingular cubic.

A nonsingular cubic curve is of genus one. The Riemann–Roch theorem states that the linear series for a simple point on a nonsingular cubic is of dimension one. Thus, there can be no rational functions other than the constant functions that have at most a single simple singularity on a nonsingular cubic. The constant functions form a vector space of dimension one. A nonconstant function would mean that the vector space was at least dimension two, a contradiction. If the curve had a rational parameterization, however, then the function $x = 1/(\alpha - t)$ would be a nonconstant function with a simple pole at α.

Finally, if the curve of intersection had a rational parameterization, then so would its projection. But the projection, a nonsingular cubic, has no rational parameterization. Thus, we conclude that there is no rational parameterization for the curve of intersection of two quadric surfaces when the intersection is an irreducible degree-four space curve.

Notice that a generalization of the Riemann–Roch theorem is needed in the area of blending surfaces. Instead of a plane curve, consider a surface S in three-space. Rather than selecting points on a curve, select algebraic curves on the surface, and with each curve, associate a positive integer. Then consider the class of surfaces that intersect the given surface S only at the prescribed curves, and whose degree of intersection is at most the associated integer. For example, for a surface to be tangent to the original surface in a given curve, associate integer two with the curve. Such a generalization of the Riemann–Roch theorem would provide a theory of blending surfaces. Note that students in traditional computer science departments do not often hear about these kinds of theorems.

In computer-aided design, one needs interactive three-dimensional graphics and hence efficient algorithms for rapid display of objects on engineering workstations. A simple way to display an object is to project all edges of the object and its silhouettes, apply a hidden line removal algorithm to determine the visible faces, and then shade. However, objects tend to have a large number of edges. Thus, a more efficient method might be to project only the silhouette edges and apply the hidden line removal algorithm to silhouette edges only. The computation is then substantially less than working with all edges. The silhouette edges divide the object into sheets that are either facing toward the viewer or facing away. Knowing the visible sheets allows one to add the visible edges and then display the object.

Now, an interesting question is, If the object were

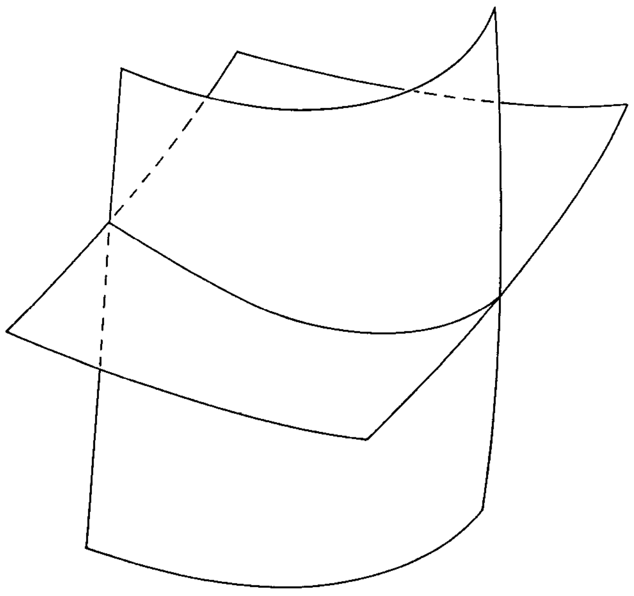
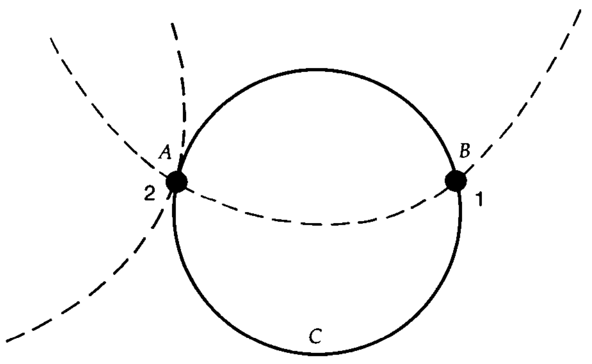


FIGURE 10. A Space Curve Defined by the Intersection of Two Surfaces



A curve C in the plane is represented by a solid circle. The points A and B have integers 2 and 1 associated with them. The set of all rational curves intersecting C at A or B or tangent to C at A forms a three-dimensional vector space. Two such rational curves are illustrated in dotted lines.

FIGURE 11. Illustration of the Riemann-Roch Theorem

changed slightly, either by editing some feature or by slight rotation, how much work could be saved? Must one go back and redo the entire computation, or can one determine what happens to the silhouette and save much of the computation? This is an important question in simulating models with moving parts, in deformation of nonrigid objects, and in animation.

Determining the effect of motion or deformation of an object involves differential geometry, which also comes into play in the calculation of envelopes. A different technique for computing a blending surface for two intersecting cylinders is to calculate the envelope of a spherical ball rolling around the smaller cylinder, maintaining contact with both cylinders. Clipping a suitable portion of the envelope yields the desired blending surface (see Figure 12).

To illustrate the mathematics of computing the envelope, consider the two-dimensional problem of computing the envelope of a circle whose center is constrained to a circular locus. A point on the surface of the small circle satisfies

$$(x-a)^2 + (y-b)^2 - 1 = 0$$

while the center of the small circle satisfies

$$a^2 - b^2 - 9 = 0.$$

Moving the small circle an infinitesimal amount and intersecting with the original circle yield a point on the envelope. This gives a third equation

$$b(x-a) + a(y-b) = 0.$$

The theory of resultants allows us to eliminate a and b from these equations and gives the equation of the envelope:

$$(x^2 + y^2 - 4)(x^2 + y^2 - 16) = 0.$$

Each of the problems mentioned makes use of some branch of mathematics that today's computer science student gets little exposure to: classical algebra, algebraic geometry, differential geometry, and elimination theory. As a final example, we consider some problems in motion planning that make use of algebraic topology. Motion planning has been intensively studied by many researchers. Some of the salient work is that of Lozano-Perez and Wesley [5] who formulated the configuration-space approach to the problem. Schwartz and Sharir [7] showed that motion planning can be reduced to a problem in the theory of reals, and O'Dunlaing, Sharir, and Yap [6] developed retraction techniques. These researchers have developed algorithmic solutions for motion planning that, in a formal sense, completely solved the problem. The techniques, however, are computationally prohibitive for interesting size problems.

An alternative approach is to use a heuristic that is effective most of the time. In planning motion for a robot doing automated assembly, a heuristic that efficiently determines mundane motions but occasionally asks for help in planning intricate maneuvers is perfectly acceptable.

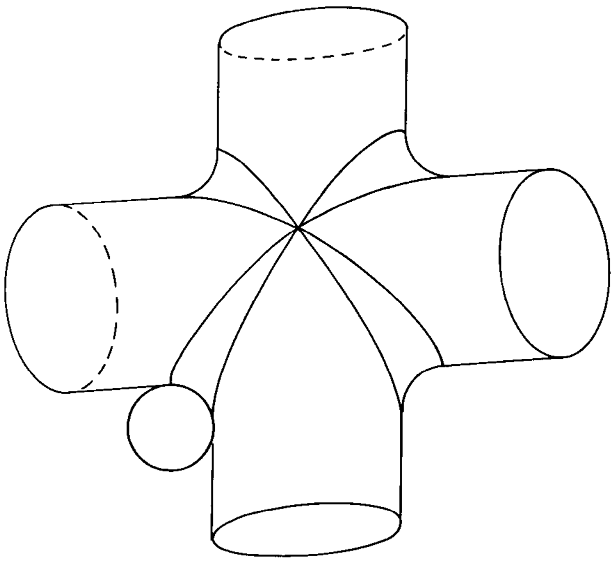
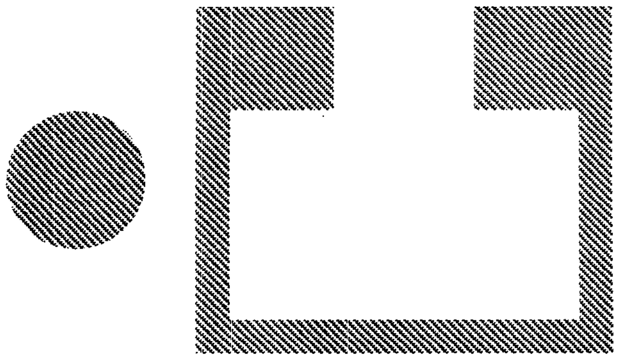


FIGURE 12. Constructing a Smooth Blend between Two Cylinders by Calculating the Envelope of a Ball Moving in Contact with the Two Cylinders


(a) A Circular Disk and an Obstacle with a Cavity

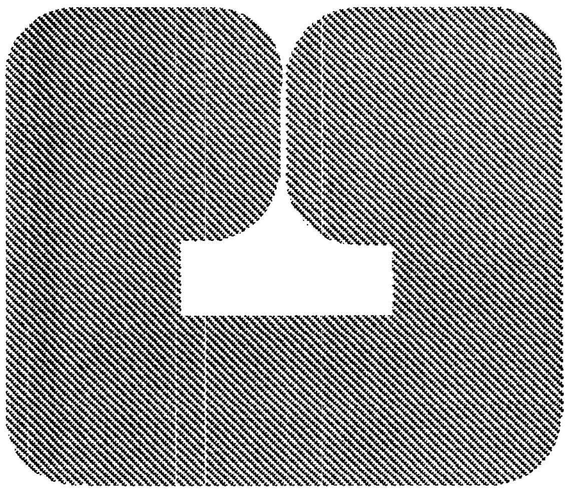

(b) Configuration Space

FIGURE 13. Configuration Space for a Circular Object and Obstacle with a Cavity

In doing robot motion planning, the intent is not to actually follow a trajectory in which the robot scrapes the wall. The idea is to first find any path, even if it involves scraping a wall. Given a path, a second procedure is applied that moves the trajectory a safe distance from the wall. The computational difficulty occurs in the first step of simply finding a trajectory. What I would like to do is justify the heuristic, but first I must discuss the concept of configuration space.

To represent the configuration of a robot arm in a work cell, attach a coordinate system to each moving part and specify the position and orientation of each coordinate system in terms of an earlier one. This provides a set of parameters that determine the configuration of the arm. In a given configuration, the set of parameters will have a set of values that can be viewed as a point in a high-dimensional space called configuration space. Certain points in this configuration space are illegal in that they correspond to a configuration of the arm where the arm overlaps some object in the work cell. The set of illegal points is called a *configuration-space obstacle.* This idea is illustrated in Figure 13a with a circular object and an obstacle with a cavity. The configuration of the circle and the obstacle is represented by two parameters specifying the location of the center of the circle. Thus, in this example, configuration space is two dimensional. The configuration-space obstacle is shown in Figure 13b.

Configuration-space objects can have a very complex structure. Figure 14 illustrates two types of holes that can exist in a configuration-space obstacle. Hole *A* is like a hole in a doughnut: It is a hole through the object, but it does not partition the surface into two components. Hole *B*, on the other hand, is like a yeast hole. If in baking the doughnut

One such heuristic is illustrated by motion planning for a maze. To plan a path from an initial position to a final position, move the initial and final positions along straight lines until encountering walls; then plan motion along the walls. The advantage of this method is that the problem can be reduced from a two-dimensional search to a one-dimensional search by just moving along the wall from the modified initial position to see if the modified final position is encountered. In order to justify this heuristic, one must establish that, whenever it is possible to get from the initial position to the final position, there is some way of doing it by moving solely along the walls.

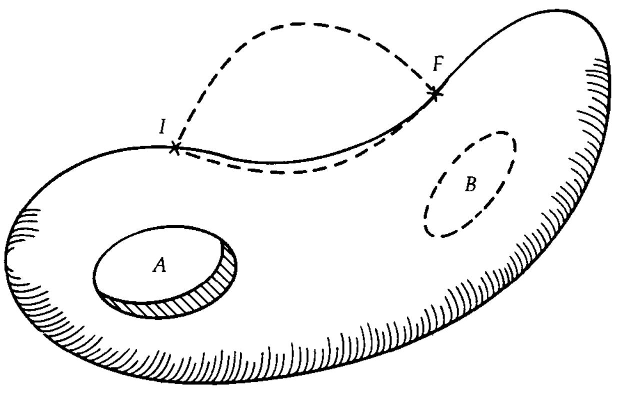


FIGURE 14. Configuration-Space Obstacle with Holes

there was some yeast, one would get a hole in the interior of the doughnut where the surface of the hole was not connected to the remainder of the surface of the object. The surface has two connected components.

In order to justify the search heuristic mentioned earlier, we need to prove that the existence of a motion of two objects from an initial position in which they are in contact to a final position in which they are in contact implies the existence of a motion that always maintains contact of the objects [4].

The meaning of the theorem is illustrated in Figure 14. In configuration space, a motion in which an object stays in contact with an obstacle is a path on the surface of the configuration-space obstacle. A legal path from I to F is illustrated along with the claimed path on the surface of the configuration-space obstacle. If F were on the surface of the hole B, then there would be no path on the surface of the configuration-space obstacle since I and F are in different components of the surface. In this case, however, the theorem does not claim such a path, since the hypothesis of the theorem is false. There is no legal path between I and F.

The proof of the theorem makes use of algebraic topology [8]. Algebraic topology reasons about properties of a space by mapping the space onto an algebraic structure, usually a group, in a manner such that topological properties are mapped to algebraic properties. These groups are called homology groups and are denoted H_0, H_1, etc.

The Zeroth Homology Group, H_0, provides information about the connectivity of the space. If the space is path connected, then H_0 is isomorphic to the integers under addition. If there are k connected components in the space, then H_0 is isomorphic to k copies of the integers.

The First Homology Group, H_1, provides another kind of information about the space. It tells us how many holes there are in the space. If the space is topologically equivalent to a disk in that it is contractible to a point, then H_1 is the trivial group consisting of only the identity element. H_1 for a sphere with k handles is isomorphic to k copies of the integers under addition. Thus H_1 provides us with information concerning the number of holes in the space.

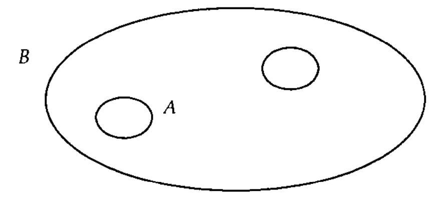
The theorem justifying the motion planning heuristic follows from the Mayer–Vietoris theorem that states that the sequence of homomorphisms connecting the homology groups

$$H_1(A \cup B) \xrightarrow{h_1} H_0(A \cap B) \xrightarrow{h_2} H_0(A) H_0(B) \xrightarrow{h_3} H_0(A \cup B) \xrightarrow{h_4} \varnothing$$

is an exact sequence. An exact sequence is one in which the image of one homomorphism is the kernel of the next.

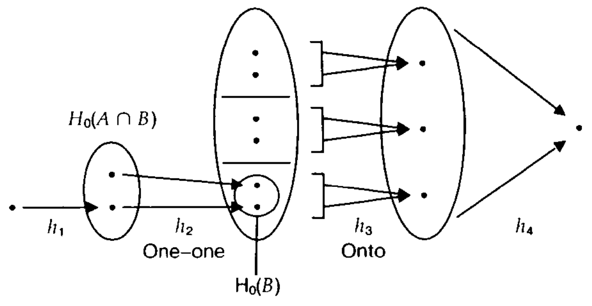
Let A be the configuration-space object, and let B be the closure of the complement of A. Thus $A \cup B$ is the entire configuration space, and $A \cap B$ is the boundary of the configuration-space obstacle. Assuming that the entire space is contractible to a point, $H_1(A \cup B)$ is the trivial group consisting of only the identity element. Assuming that the space is connected, $H_0(A \cup B)$ is Z, the group of integers under addition. It is an easy exercise to show that the configuration-space object is connected and hence that $H_0(A)$ is also Z. $A \cap B$ is the boundary, and thus $H_0(A \cap B)$ gives the number of components in the boundary. B is the set of valid configurations. Thus $H_0(B)$ is the number of components of the free-space region. $H_0(A \cap B)$ and $H_0(B)$ are the only two groups that are not known. The Mayer–Vietoris theorem can be used to relate them.

The argument used to relate $H_0(A \cap B)$ and $H_0(B)$ is illustrated in Figure 15. Since H_1(A ∪ B) is the



$$\underbrace{H_1(A \cup B)}_{\varnothing} \xrightarrow{h_1} \underbrace{H_0(A \cap B)}_{\text{Boundary}} \xrightarrow{h_2} \underbrace{H_0(A)}_{Z} \oplus \underbrace{H_0(B)}_{\text{Valid configurations}} \xrightarrow{h_3} \underbrace{H_0(A \cup B)}_{Z} \xrightarrow{h_4} \varnothing$$

(a) The Exact Sequence



(b) Artist's Conception of the Sequence

FIGURE 15. Exact Sequence of Homomorphisms

trivial group consisting of a single element, the identity element, the image of h_1 must be a single element of $H_0(A \cap B)$. The Mayer–Vietoris theorem claims that the sequence of homomorphisms is exact. Thus the image of h_1 is the kernel of h_2. Since this kernel is a single element, h_2 makes no identification and hence is one-one. Similarly, since the range of h_4 is the identity, the entire group, $H_0(A) \cup H_0(B)$, is in the kernel of h_4. Thus the exact sequence property implies that h_3 is onto. Simple arguments of this nature relate the group $H_0(A \cap B)$ of the boundary to the group $H_0(B)$ of free space and show that each connected region of free space has a connected boundary. This establishes the theorem. The technique is sufficiently general so that one can define motion to be ordinary translational, or rotational motion, or to be any continuous deformation.

Suppose for a minute that the object and the obstacle are polyhedrons. Moving the object until it contacts the obstacle corresponds to a path in configuration space that terminates on the surface of the configuration-space obstacle. Suppose that the initial contact of the object and the obstacle is between a vertex of the object and a face of the obstacle. Subsequent motion may consist of the object rotating until an edge and then a face come into contact with the face of the obstacle. In configuration space, this corresponds to moving along a face of the configuration-space obstacle until the face intersects another in a lower dimensional face. By moving along lower and lower dimensional faces, eventually a vertex is reached. If one could prove that not only can the search be restricted to the surface of the configuration-space obstable, but that it can be restricted to edges, then the motion planning problem would be reduced from searching a space to graph traversal, a much simpler problem. Unfortunately, there are situations in which this is not sufficient. Algebraic topology provides answers to such questions.

Returning to our original theme, applications are bringing to computer science a need for more classical mathematics. Over the past 20 years, computer science has gradually deleted much of the traditional mathematics from its curriculum—each time a course was added in discrete structures or discrete mathematics, some other course was dropped. If I am correct that the nature of computer science is changing from a science concerned with its internal development to one very concerned with applying its knowledge and techniques to applications, then I suggest we go back and reconsider the curriculum we have developed and ask if it is proper for the next 20 years.

Acknowledgments. The author is indebted to Dean Krafft and Christoph Hoffmann for discussions that helped crystallize many of the ideas in this article, for support in implementing the graphics software used in producing the accompanying photographs, and for the actual production of the photographs themselves.

REFERENCES

1. Hoffmann, C., and Hopcroft, J. Automatic surface generation in computer aided design. *Visual Comput. 1*, 2 (Oct. 1985), 92–100.
2. Hoffmann, C., and Hopcroft, J. Quadratic blending surfaces. TR 85-674, Dept. of Computer Science, Cornell Univ., Ithaca, N.Y., Sept. 1985.
3. Hopcroft, J., and Krafft, D. The challenge of robotics for computer science. In *Advances in Robotics.* Vol. 1. *Algorithmic and Geometric Aspects of Robotics*, C. Yap and J. Schwartz, Eds. Lawrence Erlbaum Associates, Hillsdale, N.J. To be published.
4. Hopcroft, J., and Wilfong, G. On the motion of objects in contact. *Int. J. Robotics Res.* To be published.
5. Lozano-Perez, T., and Wesley, M. An algorithm for planning collision-free paths among polyhedral objects. *Commun. ACM 22*, 10 (Oct. 1979), 560–570.
6. O'Dunlaing, C.O., Sharir, M., and Yap, C. Retraction: A new approach to motion-planning. In *The 15th ACM Symposium on Theory of Computing* (Boston, Mass., Apr. 25–27). ACM, New York, 1983, pp. 207–220.
7. Schwartz, J.T., and Sharir, M. On the piano mover's problem II: General techniques for computing topological properties of real algebraic manifolds. TR 41, Computer Science Dept., New York Univ., 1982.
8. Vick, J.W. *Homology Theory: An Introduction to Algebraic Topology.* Academic Press, New York, 1973.
9. Walker, R.J. *Algebraic Curves.* Springer-Verlag, New York, 1978.

CR Categories and Subject Descriptors: I.2.9 [**Artificial Intelligence**]: Robotics; I.3.3 [**Computer Graphics**]: Picture/Image Generation—*display algorithms*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*visible line/surface algorithms*; I.6.0 [**Simulation and Modeling**]: General; J.6 [**Computer-Aided Engineering**]: *computer-aided design (CAD)*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education, curriculum*

General Terms: Algorithms, Design

Additional Key Words and Phrases: electric prototyping, solid modeling

Author's Present Address: John Hopcroft, Dept. of Computer Science, Cornell University, Ithaca, NY 14853.