Electronic Prototyping

J. Hopcroft
Cornell University
Ithaca, NY 14853

Abstract

The potential benefits of automation in space are significant. The science base needed to support this automation not only will help control costs and reduce lead-time in the earth-based design and construction of space stations, but also will advance the nation's capability for computer design, simulation, testing, and debugging of sophisticated objects electronically.

Progress in automation will require the ability to electronically represent, reason about, and manipulate objects. This paper discusses the development of representations, languages, editors, and model-driven simulation systems to support electronic prototyping. In particular, it identifies areas where basic research is needed before further progress can be made.

Introduction

An important aspect of automation is the ability to represent, reason about, and manipulate physical objects. This is true in the manipulation of objects in space as well as in the cost-effective design and manufacture of sophisticated parts. A science base is needed to facilitate these activities. This science base must support the modeling and editing of three dimensional objects, electronic prototypes, model-driven simulations, and automated designs. In this paper, the necessary components of the science base are discussed, and a number of examples are presented which illustrate the benefits that would accrue from such research.

Electronic Prototyping

Electronic prototyping is the process of constructing models of physical objects in a computer to support activities such as computer-aided design, engineering analysis, design verification, and automated manufacturing. Electronic prototyping will play an important role in the design, manufacture and operation of space stations as well as having great commercial value.

When compared to the current practice of constructing physical prototypes, the development and use of electronic prototypes (computer models of objects) offer substantial advantages, especially with regard to the design of more complex systems. Take the situation of a satellite with an antenna that deploys in space: it may be the case that the antenna will not support its own weight under gravity and, thus, it would be difficult to construct a prototype from a physical model. An electronic prototype overcomes this difficulty and allows the integration of design and manufacture. A further advantage occurs should the antenna fail to deploy properly. One would be reluctant to experiment with a procedure that might destroy the prototype if only a physical prototype is available for testing hypothesized dejamming procedures. However, an electronic prototype eliminates such worries. Furthermore, one can test hypotheses in parallel on duplicate copies of the antenna model.

A major component of an electronic modeling system is a model-driven simulation. The system must be able to automatically construct the equations of motion from the geometric model. The difference between such a system and current dynamic simulation systems is that during a model-driven simulation a collision detection algorithm is run. Whenever a collision

occurs, the dynamics of the system are edited to account for the new point of contact, and the simulation continues. Such a system could be used to simulate gripping and approach strategies for robots, testing designs of multifinger hands, designing algorithms for rotation and manipulation of objects, and studying walking strategies. The ability to redesign a multifingered grip in a few hours and then simulate the new design provides an enormous advantage over building multifingered grippers in hardware. Of course, promising designs must be tested in hardware since simulations often miss pertinent factors.

The versatility of an easily programmed system would allow simulation of a person performing jobs in a weightless environment in space to evaluate various procedures or it could be used to simulate a person on earth allowing us to understand better how tasks are performed. A spinoff of such research would have an impact on many disciplines. For example, when designing artificial joints such as a knee, it is important to understand the forces on the joint as it goes through a range of human activities such as sitting down in a chair, walking up stairs, or stepping off a curb. Knowing the forces would allow verification that the surface of the artificial knee would stay in contact with the surface of the bone for a specific individual with a specific height, weight, and body structure.

One of the first requirements of a modeling system is the ability to construct 3-dimensional computer representations of rigid objects. Although much is known about solid models and boundary representations, very little is known about how to effectively construct or edit a model of a solid. Constructing a model of an object such as an automobile crankshaft from an already existing design may take on the order of a man-month of effort. Clearly, for it to be economically feasible to construct such models, we must produce the tools necessary to significantly reduce this effort. These tools are software tools whose development will require basic research into languages and man-machine interfaces. In order for the cost of constructing the model to be justifiable, the constructed model must support the entire range of engineering activities. These activities include calculations of mass and moments of inertia as well as stress and vibration analyses requiring finite element techniques.

In computer automated design, one observes that a part such as a crankshaft has certain surfaces whose shape is precisely determined by the function of the surface. The shape of other surfaces is not critical and they can be arbitrarily selected provided that they conform to some simple criteria. The crankshaft also has certain global constraints on its design. For example, it must be balanced about certain axes. Thus, one step towards automated design would be to specify the surfaces that need precise shape along with a class of surfaces from which to select the remainder and have the computer complete the design. The crankshaft design would be completed by selecting the remaining surfaces from the class of allowable surfaces so as to satisfy the goal design criteria. An example of the potential savings in time and energy can be seen from [1] where an estimated savings factor of 20 can be achieved in constructing models by automating the blending surfaces.

The required science base for supporting automation is extensive and we can only present a few examples to illustrate the foundational work that must be done. One of the basic areas needing further development is the area of representations. Today there are over fifty commercial solid modelers. Most use boundary representations based on polygonal approximations to the objects, although a number use quadratic surfaces and some use parametric patches. Very little is understood in terms of the trade-offs between the various approaches. For example, using algebraic surfaces requires more computations per face, but using polygonal approximations requires many more faces to represent an object. Whether the increase in the number of faces for the polyhedral approach offsets the savings of the simpler computations is an important question. To resolve it will most likely require the knowledge gained from both the theoretical studies of the type currently taking place in the computational geometry discipline and the practical experience obtained from studies using actual modelers.

Another important avenue that needs to be explored is the development of reliable intersection algorithms for low degree surfaces. To the best of this author's knowledge, no completely reliable algorithm exists for intersecting a quadratic surface with a quartic surface such as a torus. One difficulty in the modeling domain is that degeneracies are the rule rather than the exception. Thus, while many applied mathematicians dismiss ill-conditioned problems with the statement that the problem should be reformulated, the geometric modeler must find efficient and numerically reliable techniques for solving ill-conditioned problems. If two lines l_1 and l_2 intersect a third line l_3 at points sufficiently close together, one can assume that the intersection points either coincide or do not coincide. However, once having made an arbitrary decision, one must insure that at some future point an inconsistent decision is not made. Developing a theory to determine which decisions are independent would be a major accomplishment. Numerous similar questions arise that illustrate the importance of developing a science base to answer such questions in geometric modeling.

Intersection algorithms tend to have running times that are quadratic in terms of the number of faces, since every face must be intersected with every other face. A number of modelers have overcome this by boxing the faces and determining subsets of faces that need not be intersected with other subsets. Empirically, this seems to reduce the execution time of the algorithm from n^2 to $n^{3/2}$. A more promising approach is to find a point on the intersection and trace the curve of intersection to determine the pairs of faces that need to be intersected. The difficulty is that, at the present state of knowledge, the problem of locating two faces that intersect is as hard to solve as the intersection problem itself. In particular, if two objects whose intersection is to be calculated do not intersect, how does the algorithm establish this fact? One promising approach is to triangulate the space exterior to the two objects. This process will either show no intersection or determine a point of intersection. Results in computational geometry suggest that we may soon have techniques to perform triangulations in time order of $n\log n$, a substantial savings over the current n^2 algorithms. More important the $n\log n$ bound is a worst case that is not usually encountered whereas the current n^2 algorithms use time n^2 independent of the intricacies of the problem. Considering that even simple object domains may involve on the order of 10,000 faces, an increase in computing time of a hundredfold is highly likely. Intersection of objects is just one example of an operation on objects. What is needed is the development of the underlying theory to support efficient and reliable algorithms for calculating Boolean operations, swept volumes, offsets, envelops, triangulations, etc.

Another major area that needs investigation is that of editing objects. Today there are good editors for text and good editors for programs because we understand the structure of text and the structure of programs. Text consists of paragraphs that consist of sentences, sentences are made up of words, and so on. A good editor makes use of this structure. In programming, good programmers do not start from scratch each time they construct a new program. Rather, they select a previous program in one window and use bits and pieces to construct a new program in another window. It is essential that in modeling components we develop the ability to reuse pieces of old models. So far this has not happened. A better understanding of the internal structure of physical objects needs to be achieved. For example, consider a cube defined as the intersection of three slices, an x-slice, a y-slice, and a z-slice. Let the planes defining the x-slice be called left and right, the planes defining the y-slice be called front and back, and the planes defining the z-slice be called the top and bottom. Let the front_right_top vertex be the intersection of the front, right, and top planes. Now consider graphically editing the cube by moving the front_right_top vertex. Once the vertex is moved the three planes must be moved to maintain the constraint that the front_right_top vertex is the intersection of the front, right and top planes. In this case the cuboid changes in dimensions but is still a cuboid. On the other hand if the cuboid had been defined with the vertices as basic elements, the edges as lines connecting certain pairs of vertices, and the faces as being patches, then moving the

front_right_top vertex has a quite different effect. In particular, the coordinates of the vertex are modified, three edges incident at the vertex change their orientation, and three faces change from being planar to hyperbolic paraboloids. The cuboid ceases to be a polyhedral figure and has curved faces.

This example illustrates that geometry alone does not capture the nature of a physical object. In fact, it may well be that for editing purposes an abstraction of the object devoid of geometry is essential. The editing and reuse of the design is done at the level of the abstraction, and the geometric properties are then derived from the abstraction. Ultimately, as models are created, it is clear that some geometric properties will have to be symbolically recorded. For example, the threads on a screw do not need to be geometrically represented for most applications. However, they must somehow be represented. As important as the reuse of previous designs is, surprisingly little research has been devoted to this area; a science base is almost completely lacking. This is likely to seriously impede the nation's efforts to automate the design and manufacturing process and will critically affect areas of high technology such as space exploration.

User interfaces will play an important role in increasing productivity. To illustrate the effectiveness of well designed user interfaces, consider the example of initializing a simulation of a man diving off a block. To describe to another person how to place the diver on the block, one would simply say "place the diver on the diving block". The other person would automatically understand that the diver should be placed with feet on the block, standing in an upright position, and facing forward with hands at sides. The fact that humans have internal models of the world allows them to communicate complex situations to others using relatively short messages. The fact that one human knows by and large how another human will interpret information allows him or her to structure communications so that the correct interpretation will be achieved. Computer interfaces are needed that allow the same ease of communication.

With our current software interfaces, describing the initial position of the diver would be a tedious task. Looking at a very simplistic model, assume the diver has one hundred degrees of freedom. In this instance, the user would need to specify one hundred parameters. Although well designed systems have default specifications, it is highly unlikely that default specifications would greatly reduce the number of parameters needed for the diver. However, a simple algorithm that takes advantage of partially supplied knowledge to fill in defaults might make human-to-computer communications almost as effective as communication between humans. For example, if points A and B on an object were specified to be placed at A' and B' in space, the algorithm might fix the remaining degrees of freedom by translating A to A' and then performing a minimum rotation to get B to B', i.e., a rotation in the plane determined by the points A, B and B'. There would be no extraneous rotation about the AB axis. The reason a set of simple heuristics such as the above is powerful is that the user understands how defaults are supplied and quickly learns how to initialize objects with minimal information. A simulation system with an easy interface for describing models, tasks, and initial configurations would be a powerful tool for developing such things as a robot arm capable of manipulating and repairing satellites in space. The design of such an arm could be greatly enhanced if one could easily edit a design to try out various ideas and easily specify procedures for using the arm.

A very promising avenue of research is symbolic specification. Objects can be assembled and manipulated symbolically by developing automatic naming conventions and inheritance methods. A human has considerable difficulty with coordinate systems in 3-space. Thus, rather than trying to specify location directly, locations are specified by constraining a feature of one object to mesh with a feature of another object. This usually requires the computer to maintain and solve systems of constraints. Research is needed in this area to eliminate the need to solve arbitrarily complex systems of constraints and to rapidly detect inconsistent

systems of constraints.

So far we have talked almost exclusively of objects that are physical entities. In addition to thinking about physical objects, we must also think of things such as tasks, trajectories, etc. and understand how to represent and edit them. In using a robot to perform several similar tasks, it would be preferable to take the code that was developed for one task, edit it, and reuse it for other tasks. This may be tedious to do at the code level since minute differences in the tasks may cause considerable differences throughout the code. Representations of the tasks at some level of abstraction from which code could be automatically produced to drive the robot would allow simple editing and allow the conversion from one task to another. The idea is a simple extension of the concept of high level programming; editing the source in the high level language is enormously easier than editing the machine language object code.

In the above we have tried to illustrate the need for a much better understanding of the software representations of objects and tasks. In addition, numerous aspects such as motion planning and configuration spaces, constraint systems, and symbolic computations involving ideals and the Grobner basis need a better understanding. Although robotics and automation deal with physical objects and are often thought of in terms of control, sensing, and instrumentation, the real nature of the subject has to do with representations, languages, abstraction, and reasoning. While these are generally computer science concepts, the researchers in robotics tend to have backgrounds in electrical and mechanical engineering. There is a need to integrate computer science into these fields. The newness of these ideas and the lack of sufficient researchers with training in computer science has contributed to the slow development of these areas.

It is crucial that the nation build the science base to support automation. The greatest challenge will be the development of the foundations in representations, languages, and user interfaces for the computing systems involved. A well thought out approach in this area is strongly needed.

Reference 1. Hoffmann, C.M., and Hopcroft, J.E. Automatic Surface Generation in Computer Aided Design, *The Visual Computer 1:2*, 1985, 92-100, Springer-Verlag.