Interference Detection Among Solids and Surfaces

John W. Boyse
General Motors Research Laboratories

In many industrial environments it is necessary to determine whether there is interference among components. There are many potential interference problems in products made up of assemblies of components and in product manufacturing and testing. Typically, drawings are used in an attempt to detect such unwanted interferences, but the two-dimensional, static drafting medium does not always show interferences among three-dimensional, moving parts. This paper presents a computer representation for solids and surfaces and algorithms which carry out interference checking among objects so represented. Objects are represented as polyhedra or as piecewise planar surfaces. Two types of interference checking are discussed: detection of intersections among objects in fixed positions and detection of collisions among objects moving along specified trajectories.

Key Words and Phrases: interference checking, intersection detection, collision detection, solid representation, polyhedral representation, graphics, polygons, surfaces

CR Categories: 3.2, 8.2

Introduction

In many industrial environments it is necessary to ascertain whether or not there is interference among components. In products made up of assemblies of components, and in product manufacturing and testing facilities, there are many potential interference problems. In current practice an attempt is made to detect such unwanted interferences before the fact by using drafting methods. Engineering drawings of the product assembly or test facility are prepared to show the various components in a number of views and possible positions. If interferences are detected on these drawings, modifications are made and new drawings are prepared. Unfortunately, the two-dimensional, static drafting medium does not always show interferences among three-dimensional objects, especially when these objects can move relative to one another. When drawings fail, the interference problems appear in the prototype stage or when facilities are set up for production. Correcting these problems at this point is expensive and time consuming. What is needed to avoid these unwanted interference problems is first, a true three-dimensional representation of the objects to be checked for interference, and second, ways of using this representation to tell when such interferences occur.

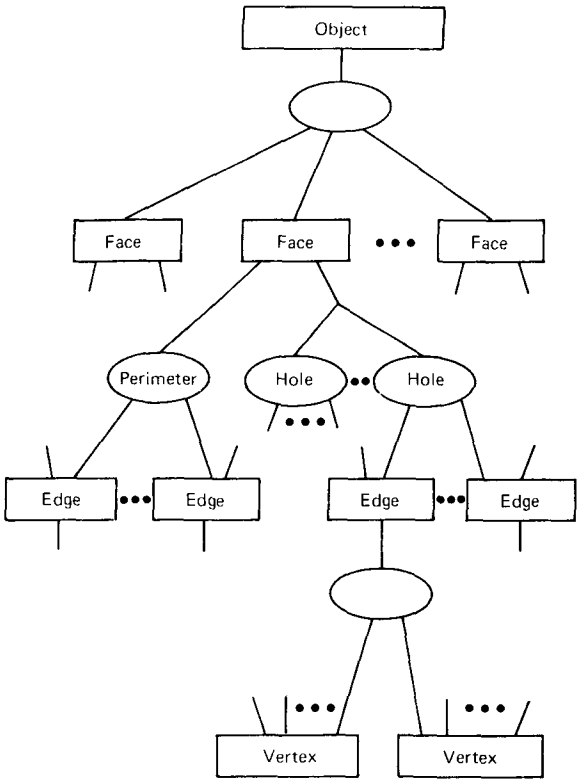
Computer representations of sculptured (free form) surfaces are highly developed and have been used successfully for computer aided design in the automotive, aircraft, and shipbuilding industries [8]. This work has been concentrated on elegant mathematical representations for sculptured surfaces but has not addressed the problem of representing solids. Most mechanical components are characterized by the relationships among the fairly simple surface types (e.g. planar and cylindrical) that enclose them and not by the complexity of their surfaces. For computer aided interference checking among solids, a representation is needed that takes advantage of the simple nature of the surface patches and, in addition, represents the complex boundaries between these surface areas which combine to enclose a solid object.

A few researchers have realized the paucity of results in the area of geometric modeling of solids with "simple" surfaces and are looking into various aspects of this problem. Geometric modeling work at the Universities of Rochester and Cambridge has been aimed toward computer aided design and manufacture of simple mechanical parts [1, 5, 6, 7, 15, 16]. At Carnegie-Mellon the work has been oriented toward computer aided architecture [9], while at Stanford the objective has been to provide internal descriptions of objects for a computer vision system [2, 3, 4]. All these systems limit object boundaries to planar or planar and cylindrical surfaces. The systems also provide operators which make it possible to build a representation for a complex object by taking set unions, intersections, and differences of simpler objects (primitives). These operators are subject to certain restrictions on the primitives, e.g. relative locations, orthogonality, or types of primitives allowed in an operation. The restrictions vary from one system to another.

The work at the University of Rochester is especially interesting because it rests on a sound mathematical foundation. Solids are rigorously defined as regular sets of points in Euclidean 3-space [13]. (A set is regular if it

Author's address: Computer Science Department, General Motors Research Laboratories, Warren MI 48090.

Fig. 1. Basic data structure for object.



equals the closure of its interior.) Regularized union, intersection, and difference operators are defined which preserve regularity and thereby prevent the creation of nonphysical solids. Thus, for example, the regularized intersection of a stack of two cubes is null rather than being a (nonsolid) planar patch.

The following three sections present a data structure for representing 3-D objects and algorithms for two kinds of interference checking: detection of intersections among objects in fixed positions (static interference checking) and detection of collisions among fixed objects and objects moving along specified trajectories (dynamic interference checking). An interactive graphic interface to these algorithms provides users with the ability to create, display, and manipulate objects in space and to check them for interference; this is discussed in a section on implementation.

Very little has been published which is aimed specifically at the interference checking problem. Forrest [10] mentions the problem and suggests some possible approaches. Pieper [12], Widdoes [18], and Udupa [14] attack the problem heuristically with the specific goal of determining collision-free trajectories for a robot arm. The geometric modeling systems alluded to earlier [1–7, 9, 15, 16] solve the static interference checking problem with their intersection operation because non-null intersection of two solids implies interference. (As noted above, the system at the University of Rochester uses regularized set operations [13] and thus distinguishes surface contact from interpenetration.) To the author's knowledge, the only published work aimed entirely at interference checking was done by Maruyama [11]. He developed an algorithm which differs from that developed here for static interference checking among solid objects but did not discuss dynamic interference checking.

Representation of Solids and Surfaces

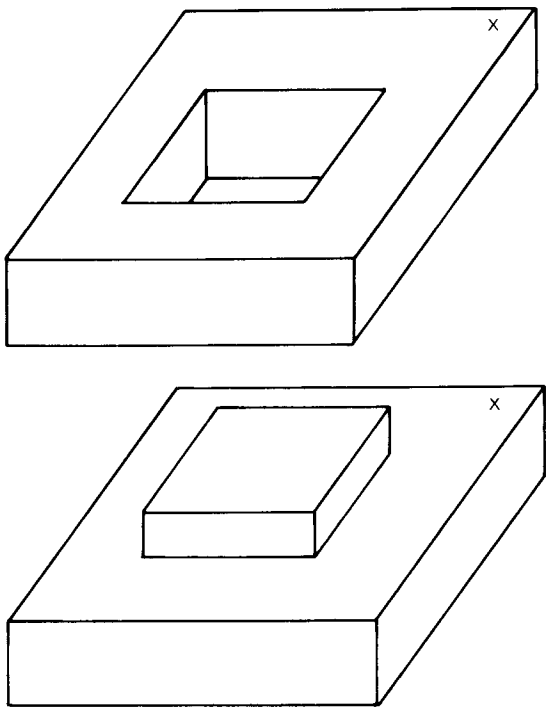
Many computer representations are possible for physical objects. This section explains the data structure used to describe objects and discusses some of the reasons for using it. Three basic types of objects are represented: solids, surfaces, and containers. Solids and containers are both depicted by polyhedra that may have an arbitrary number of faces and holes; the difference is that containers are void on their interior. Surfaces are represented by polygonal patches joined at their edges and may also have holes in them.

Although a "thin" polyhedron could characterize a surface, a separate surface representation is provided because in many applications only the surface geometry is of interest. For example, the thickness of a table top may be of no concern to a robot arm; the table top is simply a surface that supports objects and limits the arm's motion.

We limit our computer representations to polyhedra because this greatly simplifies interference checking computations and because it is easy to generate a wire-frame graphic display from such a representation. For many interference checking applications, piecewise planar approximations to curved surfaces are adequate because tolerances are broad enough that an exact representation is unnecessary. Of course, any surface can be approximated as closely as desired by polygonal patches. To get a very close approximation to a curved surface, however, requires a large number of facets, which in turn implies a large amount of storage and long processing times for interference checking.

The basic data structure used to describe a polyhedron corresponds to its topology as shown in Figure 1, where the object is bounded by faces which are bounded by edges delimited by vertices. A rectangle represents an item which may contain data and which may own entities (ellipses) which contain a set of items. The *object* item holds data which define the object type (solid/container/surface) and which define a circumscribed sphere and box for the object. A *face* is defined as a connected plane surface bounded by edges. The homogeneous coordinate representation for the plane is stored with an outward normal in the face item, and the face item owns sets which define its perimeter and any holes. These are holes in the face, not in the object, and so in Figure 2 the data structure for face "X" is identical for the two solids. An *edge* is a straight line segment defined by the two vertices it owns. Finally, each *vertex* item contains its Cartesian coordinates.

Fig. 2. Face "X" has identical structure in the two objects.



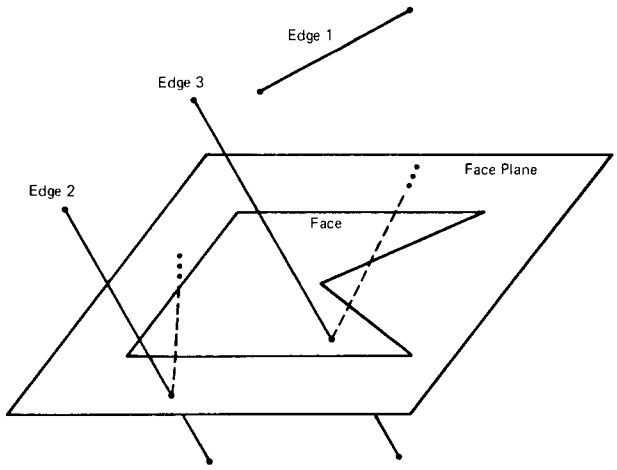
Only the basic structure is shown in Figure 1. To simplify algorithms and increase efficiency other sets exist as well, e.g. the object owns a set containing a list of all vertices. Additional sets and items are created on a temporary basis for purposes of checking collisions along given trajectories; e.g. an item in which the coefficient matrix of the quadric generated when an edge is rotated around a given axis is stored. Finally, this lowest level object may be owned by higher level objects so that complex solids or surfaces can be built up from simpler ones.

Intersection Checking

Given the data structure outlined above for representation of solids, containers and surfaces, the problem is to determine which (if any) pairs of objects intersect. In the following paragraphs we develop tests to determine when such intersections occur. The idea is that when a graphic console user moves objects on the screen, the system should audit the placement of objects to prevent the user from placing them in positions where they interfere with one another. This is called static interference checking because the concern is with stationary objects and their spatial relationships. The next section discusses collision detection; there the problem is determining whether a moving object strikes other objects as it moves along a given trajectory.

More formally, define a solid object as that set of points interior to and on the surface of the solid and let a surface comprise the set of points on the surface. Then

Fig. 3. Edge/face intersection.



two such solids/surfaces do not interfere if and only if the intersection of their defining point sets is null. Similarly, a container comprises the set of points in its interior and there is no interference between a solid/surface and a container if and only if the solid or surface is a subset of the container. Note that this definition means that interference exists when a pair of objects interpenetrates or when the faces of adjacent objects coincide. Computationally it is not difficult to insist on some small ϵ between the surfaces of noninterfering objects.

For deciding whether two objects interfere define the following four mutually exclusive relations between a pair of objects represented by point sets A and B:

(1) $A \cap B = \varnothing$.

(2) $A \cap B = A \quad (A \subseteq B)$.

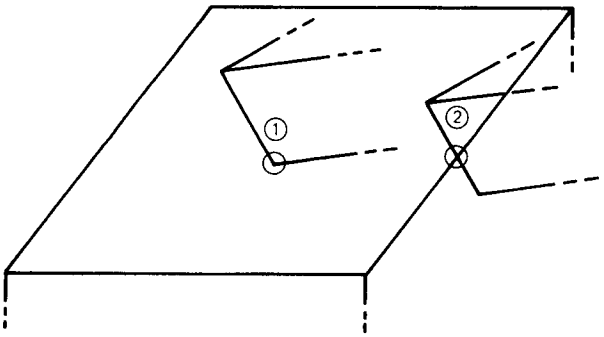
(3) $A \cap B = B \quad (B \subseteq A)$.

(4) $A \cap B = C; \quad C \neq \varnothing, \quad C \neq A, \quad C \neq B$.

As noted above, if (1) holds and both objects are solids/surfaces, there is no interference. If (2) holds with A a solid/surface and B a container, there is no interference (where it is assumed that interference exists unless A lies wholly within B). Interference checking proceeds by first deciding whether relation (4) holds and if it does not, testing for relations (1)–(3).

Surfaces of objects intersect only in case (4) and testing for this constitutes the bulk of the interference checking computation. The key to surface intersection checking is to note that the surface intersection of two plane-faced objects occurs if and only if an edge of one intersects a face of the other or vice-versa. (A face includes its bounding edges.) Some possible relations between an edge and face are shown in Figure 3. The edge cannot intersect the face if both endpoints lie on the same side of the plane containing the face and this is easily checked. If the edge intersects the face plane, we cast a ray from the intersection point to infinity in the face plane to decide whether the edge intersects the face. The edge intersects the face if and only if the ray crosses

Fig. 4. Face/edge collision.



edges of the face an odd number of times. For completeness, note that the edge can lie in the face plane. Intersection exists if the given edge crosses any edge of the face or if a ray cast from any point on the given edge has an odd number of face edge crossings.

If a test of all face/edge pairs on the two objects shows no surface intersection, then we must decide which of the relations (1)–(3) holds to complete the interference checking. With the elimination of (4), and assuming A is a solid, testing for (3) can be done by picking any vertex of B. This vertex (and therefore B itself) is inside A if the first face of A crossed by a ray cast from the vertex takes us from the inside to the outside of A. Similar tests can be used for relations (1) and (2).

It is worth noting here the sorts of difficulties that can occur with tests such as these. To test for a point interior to a solid we might have used a test analogous to that used to determine whether a point is interior to a face, i.e. draw a ray from the point to infinity and count the number of face crossings for the ray; if this number is odd, assume the point lies inside the object. This works in principle, but if the ray passes through an edge of the solid, there is a danger that the crossing will be counted twice or not at all when it should be counted once. Accounting for all such special cases in order to assure that interference checking is complete can cause major difficulties.

Checking for interference using the methods outlined above requires considerable computation for objects with many faces. To speed this up for objects that are far apart we carry in each object item the radius and center coordinates for a circumscribed sphere, and the minimum and maximum coordinate values for the object. This allows quick sphere and box intersection tests to be carried out for a pair of objects, and only if these show possible interference is detailed checking necessary.

Collision Detection

The previous section discussed static interference checking; this section discusses tests for collision between a moving and stationary object. It will be assumed in all cases that a pair of objects to be tested for collision do not intersect in their initial positions. Recall that a *face* on either a polyhedron or a surface is a connected plane surface bounded by straight line segments and includes its boundary. *Collision* between two objects occurs when the surface of one object comes into contact with the surface of a second object. For surface objects comprising polygonal patches and for polyhedra, collision occurs when a face of one object contacts a face of another.

It is not possible for the faces of two plane-faced objects to contact without the edge of one of the objects contacting a face of the other. Thus to detect a collision between two such objects, it is sufficient to detect a collison of an edge on one object with a face of the other or vice-versa. Because a face consists of its interior and a boundary, collision of a face and edge occurs in one of two ways; the edge comes into contact either with the interior of the face or with the boundary of the face (one of the edges of the face). The two cases are shown in Figure 4. The collision detection algorithm considers each of these two possible situations as follows:

1. Edge contacts face interior. Because edges are straight line segments and faces are planar, contact must occur at an endpoint of the edge. Assuming an edge moving relative to a stationary face, collision can be detected by determining the locus of each endpoint of the moving edge and examining these loci (space curves) to see whether either one intersects the face. (Note that such an endpoint is the endpoint of at least three edges and so the examination of a single endpoint services at least three edges.)

2. Edge contacts face boundary. Again assume an edge moving relative to a stationary face and note that the locus of this moving edge generates a surface in space. Collision is detected by examining the boundary of the face to see if it intersects the surface generated by the moving edge. (Note that the face is bounded by straight line segments and so the check consists of looking for intersections of line segments with the generated surface.)

These are the tests that need to be carried out for collision detection, although nothing has been said about how we might perform them. The tests are perfectly general and will work for arbitrary trajectories. Solutions have been obtained for two very important special cases: translation and rotation. Below we outline the algorithm for detecting collision between a rotating edge of one polyhedral object and a stationary face of another. Rotation is around an arbitrary axis which need not pass through the object. Similar algorithms can be used for other trajectories. The algorithm consists of two parts which correspond to the two situations discussed above and shown in Figure 4. Below we consider separately each of these two possible types of collision. Without loss of generality assume rotation around the z axis through angle $\theta > 0$.

Fig. 5. Collision of edge and face interior in the $Z = Z_{e1}$ plane.

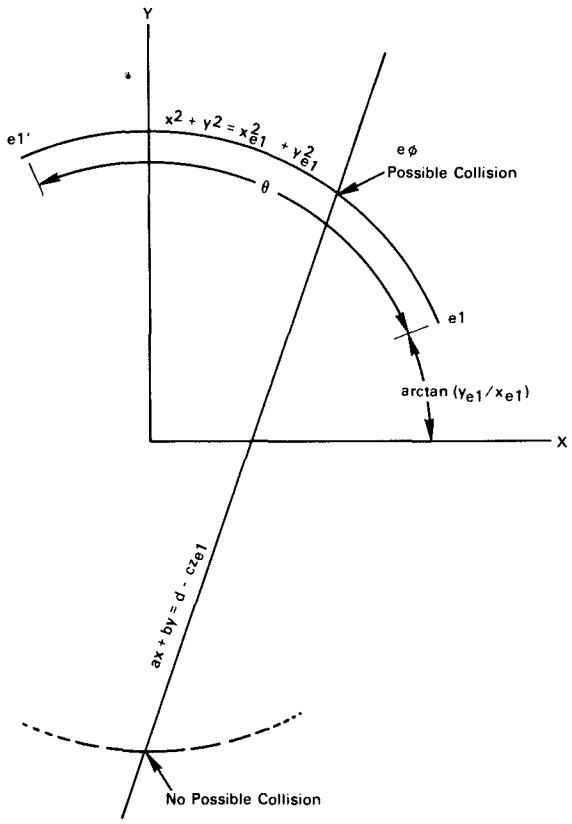
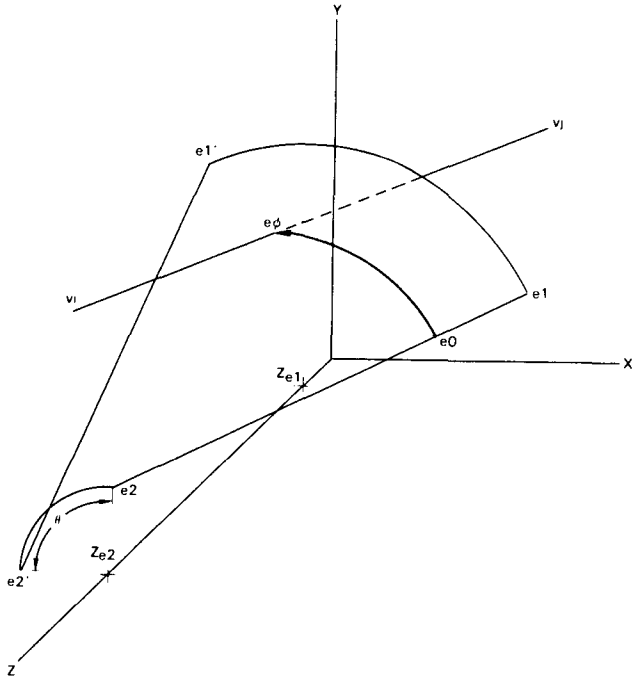


Fig. 6. Collision of edge and face boundary.



The moving edge is defined by the location of its endpoints in the initial position:

$e1 = (x_{e1}, y_{e1}, z_{e1})$,

$e2 = (x_{e2}, y_{e2}, z_{e2})$.

The face is defined by the equation for the plane in which it lies and by the locations of the (n) vertices in order around the boundary of the face:

$ax + by + cz - d = 0$,

$v1 = (x_{v1}, y_{v1}, z_{v1})$,

$\vdots$

$vn = (x_{vn}, y_{vn}, z_{vn})$.

1. Collision of Edge and Face Interior

Here we must determine if an endpoint, say $e1$, of the rotating edge intersects the face. The locus of this point lies in the plane $z = z_{e1}$ so we can work in this plane as shown in Figure 5. The moving point starts at arctan (y_{e1}/x_{e1}) and ends at arctan (y_{e1}/x_{e1}) + θ. (To simplify the discussion we will ignore the fact that arctan is multiple-valued and will also ignore special cases which occur for certain orientations of the edge and face.)

As shown in Figure 5, the locus of the endpoint lies on $x^2 + y^2 = x_{e1}^2 + y_{e1}^2$ in the $z = z_{e1}$ plane, and we wish to find intersections of this locus with the plane $ax + by + cz = d$. Intersections occur at real roots of these equations. If, for example, such an intersection occurs at point $e\phi$, two further conditions must be met for collision to take place. First, arctan (y_{e1}/x_{e1}) < arctan ($y_{e\phi}/x_{e\phi}$) < arctan (y_{e1}/x_{e1}) + θ; and second the intersection point must lie within the boundary of the face. This last condition can be determined using ray casting in the face plane, as discussed in the previous section (Figure 3).

2. Collision of Edge and Face Boundary

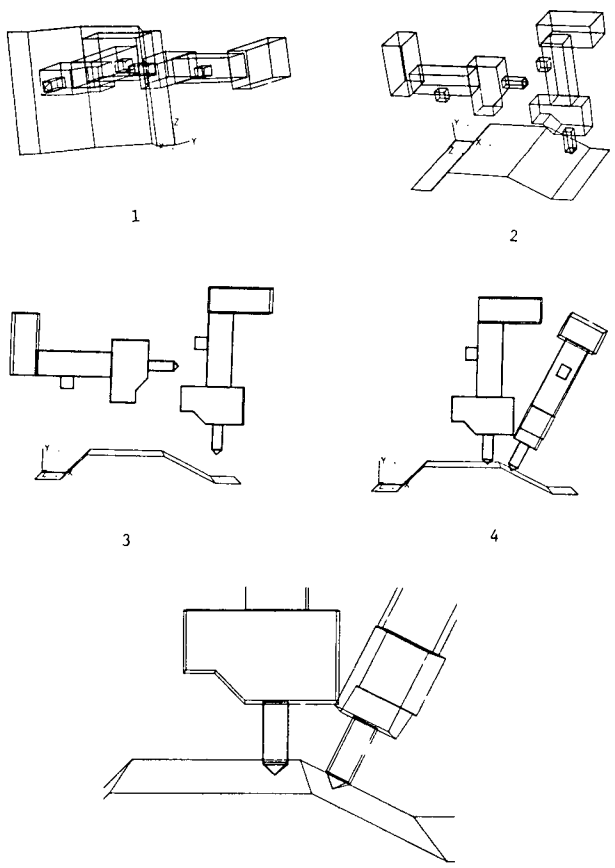
Collision occurs if an edge of the face (vi/vj in Figure 6) intersects the surface swept by the rotating edge. This surface patch is bounded by the solid lines in Figure 6 and is generated when edge $e1/e2$ rotates through angle θ to $e1'/e2'$.

The locus of any point on the rotating edge is an arc of the circle represented by $x^2 + y^2 = k_1^2$ and $z = k_2$ for some k_1 and k_2. One point on this locus must satisfy the line equations

$$\begin{aligned} x &= (x_{e2} - x_{e1})t + x_{e1}, \\ y &= (y_{e2} - y_{e1})t + y_{e1}, \\ z &= (z_{e2} - z_{e1})t + z_{e1}, \end{aligned} \tag{1}$$

and these determine k_1 and k_2 and give us the equations for the surface: (The "e" subscript has been dropped to reduce clutter.) $x^2 + y^2 = [(x_2 - x_1)t + x_1]^2 + [(y_2 - y_1)t + y_1]^2$, $z = (z_2 - z_1)t + z_1$. Eliminating t we get an explicit equation for the surface which is a hyperboloid of revolution with axis the z-axis:

Fig. 7. Sequence of displays.



$$x^2 + y^2 = [(x_2 - x_1)(z - z_1)/(z_2 - z_1) + x_1]^2 + [(y_2 - y_1)(z - z_1)/(z_2 - z_1) + y_1]^2. \quad (2)$$

This surface is unbounded and would be obtained by extending edge $e1/e2$ to infinity and rotating 360°.

Next we ask if any face edge intersects this quadric. Consider vi/vj, defined by equations

$$x = (x_{vj} - x_{vi})t + x_{vi},$$
$$y = (y_{vj} - y_{vi})t + y_{vi}, \quad (3)$$
$$z = (z_{vj} - z_{vi})t + z_{vi}; \quad 0 \le t \le 1.$$

Substitute these into eq. (2) and solve the quadratic in t. Any real root, τ, such that $0 \le \tau \le 1$ means the face edge intersects the quadric at coordinates given by setting $t = \tau$ in eqs. (3).

If the face edge intersects the unbounded surface at $e\phi = (x_{e\phi}, y_{e\phi}, z_{e\phi})$, collision occurs providing this intersection point lies within the quadric patch swept by the rotating edge. The point is inside the patch if the following two conditions hold. First, rotation is around the z-axis and so the intersection must be between planes $z = z_{e1}$ and $z = z_{e2}$, i.e. condition $z_{e1} < z_{e\phi} < z_{e2}$ must hold (assuming $z_{e1} < z_{e2}$). Second, the rotation angle must be less than θ at the intersection point, i.e. in Figure 6, arctan $(y_{e\phi}/x_{e\phi})$ − arctan $(y_{e0}/x_{e0}) < \theta$. $x_{e\phi}$ and $y_{e\phi}$ are known. x_{e0} and y_{e0} are the corresponding points on the rotating edge in its initial position and are found from eqs. (1) and the fact that $z_{e0} = z_{e\phi}$.

This completes the algorithm for finding collisions between faces and edges. The goal, however, is to detect a collision between two objects and furthermore to determine just where this collision occurs. This is done by recording all face/edge collisions and the points on the moving object's trajectory where these collisions occur. The collision between objects is then the earliest face/edge collision along the trajectory. The tests apply to any combination of solids, surfaces, or containers.

There are capabilities in addition to that of collision detection which are useful for some applications. The collision detection algorithms determine the distance translated or angle rotated before collision occurs and also the coordinates of the collision point. Thus, for example, one can use the algorithms to determine the distance separating two objects in a given direction.

As with static interference checking, there are quick collision tests which can be carried out to eliminate the necessity of detailed checking when objects are far apart. Basically, for a translating object, a circumscribed cylinder is generated which envelops the moving object throughout its trajectory. Intersection of this cylinder with the circumscribed sphere of the stationary object is then tested and if there is none, no object collision can occur. A similar sort of quick test is used for a rotating object except that here the envelope is toroidal.

Implementation

An interactive gráphic system which uses the algorithms outlined above for interference checking has commands which allow the user to translate objects for any distance in any direction or to rotate any angle around an arbitrary axis. If the user requests collision detection, the system stops the moving object at any point along the specified trajectory where a collision occurs and tells the user what object was hit, the distance or angle moved when collision occurred and the coordinates of the collision point. The user can also request that an object be moved to a new position and that a static interference check be carried out. In this case the system lists all objects that intersect the moved object in its new position. When asking for collision detection or static interference checking, the user has the option of requesting that all objects in the database be tested against the moving object or that only a user specified subset of the objects be tested.

Perspective displays of the objects being manipulated are available on the graphic console. A sequence of such displays is shown in Figure 7. The first display shows the drawback of attempting visual interference checking. It is not clear how many objects are displayed, much less whether or not they interfere. This particular simple group of objects may be viewed from angles which show clearly there is no interference (Display 3). In more

complex scenes, however, it is often impossible to find viewing angles which allow such visual verification. Display 4 shows the result of commands to move the polyhedra until they collide with the surface; in this case collision detection was used to place the polyhedra on the surface in given positions. Visually, it is not clear from this display that the polyhedra do not interfere. Blowing up the display as in 5 does not help. The interference checking algorithms, however, can tell us not only that the solids do not interfere, but also that they are separated a distance of 0.79 cm in the horizontal (X) direction.

A virtual memory data management system [17] supports the data structure outlined earlier. This system makes it easy to create, modify, and access complex and flexible network structures of the sort needed for interactive graphic applications [19] and representation of complex geometric entities.

Concluding Remarks

This paper has presented a structure for representing three-dimensional objects and algorithms for detecting intersections and collisions among these objects. This final section discusses the limitations of this system and some possible extensions.

One major obstacle to the use of any computer program that depends on a digital description of an object is that the digital description may not exist and may be fairly expensive to generate. This is really the chicken and egg problem: if only one or a few programs exist which can use the digital data, it is not worth generating; if the digital data exist for only a few components, development of programs which use the data is not worthwhile. On the other hand, once data do exist, they are usually put to many more uses than originally intended. It was noted in the introduction that other researchers in the geometric modeling area have realized the importance and usefulness of being able to build digital representations of complex solids quickly and easily and have developed ways to do this by providing primitive objects, and union, intersection, and difference operators which can be used to combine these primitives to form more complex objects. Our current system has rudimentary capabilities of this type through its object tree structure, but more extensive capabilities would greatly extend the usefulness of the system.

The limitation of the present system to polyhedra and to interference checking for translational and rotational trajectories only may be too restrictive. Many components include simple curved surfaces, e.g. cylindrical and conical surfaces. These might be represented directly as curved surfaces in the data structure; for interference checking a facet model of the surface could be generated dynamically to a specified tolerance.

Received August 1977; revised May 1978

References

1. An Introduction to PADL. Production Automation Project, Rep. TM-22, U. of Rochester, Dec. 1974.
2. Baumgart, B.G. GEOMED-a geometric editor. Rep. No. CS-414, Comptr. Sci. Dept., Stanford U., May 1974.
3. Baumgart, B.G. Geometric modeling for computer vision. Ph.D. Th., Rep. No. CS-463, Comptr. Sci. Dept., Stanford U., Oct. 1974.
4. Baumgart, B.G. A polyhedron representation for computer vision. Nat. Comptr. Conf., 1975, pp. 589–596.
5. Braid, I.C. *Designing with Volumes*, 2nd ed. Cantab Press, Cambridge, England, 1974.
6. Braid, I.C. The synthesis of solids bounded by many faces. *Comm. ACM 18*, 4 (April 1975), 209–216.
7. Braid, I.C., and Lang, C.A. Computer-aided design of mechanical components with volume building bricks. Proc. 2nd IFIP/IFAC PROLAMAT Conf., North-Holland Pub. Co., Amsterdam, 1973, pp. 173–184.
8. *Computer Aided Geometric Design*. R.E. Barnhill and R.F. Riesenfeld, Eds. Academic Press, 1974.
9. Eastman, C., Lividini, J., and Stoker, D. A database for designing large physical systems. Nat. Comptr. Conf. Proc., 1975, pp. 603–611.
10. Forrest, A.R. Computational geometry—achievements and problems. In *Computer Aided Geometric Design*, Academic Press, 1974.
11. Maruyama, K. A procedure to determine intersections between polyhedral objects. *Int. J. of Comptr. and Inform. Sci. 1*, 3 (1972), 255–266.
12. Pieper, D.L. The kinematics of manipulators under computer control. Ph.D. Th., CS-116, Comptr. Sci. Dept., Stanford U., October, 1968.
13. Requicha, A.A.G., and Voelcker, H.B. Constructive solid geometry. Production Automation Project, Report TM-25, U. of Rochester, November 1977.
14. Udupa, S. Collision detection and avoidance in computer controlled manipulators. Proc. 5th Int. Joint Conf. Artif. Intel., Cambridge, Mass., 1977, pp. 737–748.
15. Voelcker, H.B., et al. Discrete part manufacturing: theory and practice. Production Automation Project, Rep. TR-1-I, U. of Rochester, 1974.
16. Voelcker, H.B., and Requicha, A.A.G. Geometric modeling of mechanical parts and processes. *Computer 10*, 12 (Dec. 1977), 48–57.
17. Warn, D.R. VDAM—a virtual data access manager for computer aided design. Proc. Workshop on Data Bases for Interactive Design, Waterloo, 1975, pp. 104–111.
18. Widdoes, C. A heuristic collision avoider for the Stanford robot arm. Stanford AI Lab, June 1974, unpublished.
19. Williams, R. A survey of data structures for computer graphics systems. *Comput. Surveys 3*, 1 (March 1971), 1–21.