Collision Detection Techniques for 3D Models

CPS 243 Term Paper

Stefan Gottschalk
University of North Carolina
Chapel Hill, N.C.
6/15/97

1 Introduction

This paper surveys the techniques used to perform collision detection operations on 3D models. Virtual prototyping, robotics path-planning, and animation applications demand compute intensive interference testing. As hardware becomes faster and algorithmic techniques improve, collision detection is becoming more widely used in commercial systems and applied to large models.

The purpose of this paper is to bring together the techniques used to perform collision detection operations. Most of the techniques employ various divide-and-conquer approaches, such as adaptive subdivision, bounding volume hierarchies (BVHs), or hierarchical spatial subdivisions (HSSs). A few techniques resemble sweep-and-prune approaches, and a few others are difficult to categorize.

The different types of model representations generally have their own followers and techniques developed for them. For instance, the paramtric surfaces are almost exclusively the domain of the computer aided geometric design community, which has little overlap with the computational geoemtry community, for example. But the operations are very similar: reporting intersections among a set of mathematical objects is a classic computational geoemetry problem. Computing the space curve which is the intersection of two parametric cubic surfaces is a classic problem in computor-aided geometric design. They are both intersection problems.

We will begin our survey first with a review of the types of representations available for 3D models.

Next we review the variations on collision detection problem.

Then we look briefly at four fields rom which the solutions to these problems are being drawn.

Then the bulk of the paper is devoted to reviewing the techniques employed by researchers and practitioners to solve the variations of the collision detection problem.

2 Types of Representations

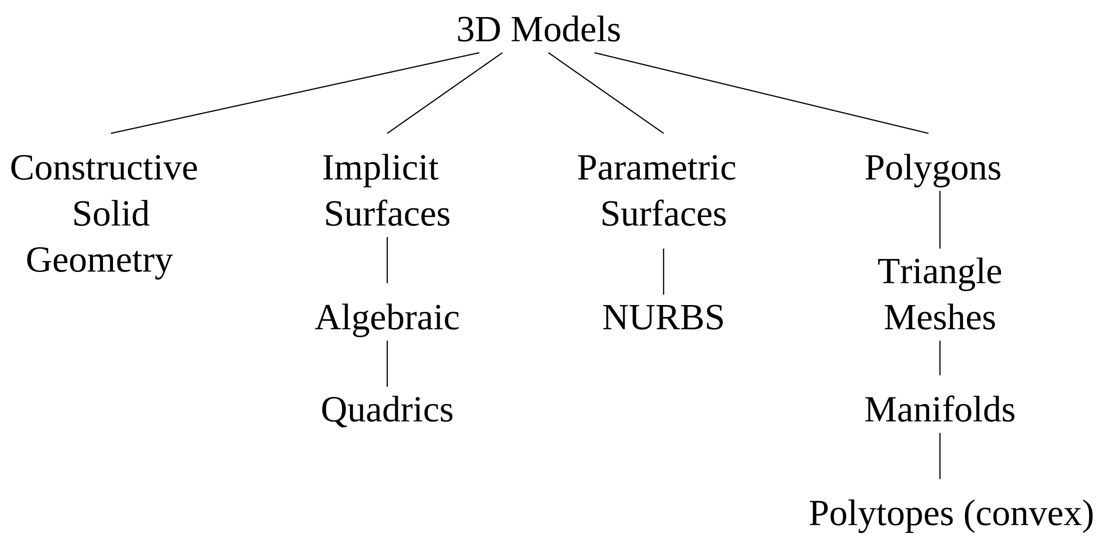

Fig 1. Taxonomy of 3D Model Representations

There are many types of model representations in use in CAD and 3D graphics. One possible taxonomy of is shown in figure 1. Four main categories of models are shown.

2.1 Constructive Solid Geometry

CSG stands for *Solid Constructive Geometry*. It is a framework in which objects are formed from primitives such as blocks, spheres, cylinders, cones, and tori, which are combined with set theoretic operations such as union, intersection, and set difference. The set theoretic operations can classify any point as being either inside, outside, or on the boundary of a composed object.

The strength of CSG is that it enables an intuitive design process of building shapes by means of cutting (intersection and set difference) and joining (union) simple shapes to form more complex ones. The operations are similar to what a machinist or mechanical engineer might execute to build a real solid object.

The difficulty with CSG is that certain operations, such as rounding an edge, or filleting a join, are difficult to describe with CSG operations. Furthermore, obtaining a true boundary representation, useful for rendering, for example, is a difficult and sometimes time-consuming task.

2.2 Implicit Surfaces

Implicit surfaces are sometimes used to describe objects, often as the primitives in CSG systems. They are defined with smooth mappings from space to the real numbers, $f : R^3 \mapsto R$, and the implicit surfaces are the loci of points where

$$f(x, y, z) = 0.$$

Such a function defines unambiguously what is inside the model, $f(x, y, z) > 0$, and what is outside, $(x, y, z) < 0$. Consequently, implicit surfaces are generically closed manifolds, which is often a desirable property of an object. If the function is polynomial in x, y, and z, then it is called *algebraic*.

A special case of algebraic surfaces are the *quadrics*, which are the second-degree polynomials in x, y, and z. These can represent slabs, cones, spheres, and cylinders in a unified framework.

2.3 Parametric Surfaces

Parametric surfaces are mappings from some subset of the plane to space, $f : R^2 \mapsto f : R^3$. Unlike implicit surfaces, parametric surfaces can generically self-intersect and do not in general describe closed manifolds. Hence, unlike CSG and implicit surfaces, they are not inherently a representation of the solid model, but rather a boundary representation.

Parametric surfaces are easier to render than implicit, and a special class called the Non-Uniform Rational B-Spline (NURBS) has gained in popularity in the CAD community. NURBS have some very nice properties which make them easier to operate on. It is worth noting that all algebraic implicit surfaces also have a NURBS representation, but the converse is not always the case.

2.4 Polygonal Models

Polygonal objects are the most commonly used model in computer graphics. They have simple representation; they are versatile; and there is hardware specifically designed to assist in the rendering of poloygons. Polygonal models are often used to approximate smooth and curved objects, such as spheres or cylinders. The more polygons which are used, the more closely the object is approximated. A polygon can be represented as a sequence of planar vertices in space. A polygonal model, in general, is just a collection of polygons, which need not necessarily be geometrically connected – these can be called "polygon soups."

For many applications, polygons can be tiled or form meshes, a suitable arrangement for representing terrain, cloth, and the surfaces of solid objects. Again, some graphics computers contain special hardware for accelerating the rendering of triangle meshes.

If the polygons form a closed manifold, then the model has a well-defined inside and outside – it is a proper solid. Some graphics algorithms rely on this structure.

If the object defined by the closed manifold is convex, then this additional structure can be exploited in collision detection algorithms.

3 Variations of the Collision Detection Problem

Collision Detection is a broad problem domain. There are many variations, which will be described below.

3.1 Query type: Contact, Intersection, Distance, ETA

In the simplest case, we are to determine only whether two models are in contact. Sometimes, we must compute which portions (if any) of two models are in contact, i.e. find their intersection. A related problem is distance computation: if two objects are not in contact, then what is the shortest line that joins them? Finally, if we know the placements of the objects and their motions, when will be the next collision between them? ETA refers to this, borrowing from the phrase, "estimated time of arrival".

Different applications have need of these different queries. For instance, a virtual prototyping application has a representation of a complex mechanical part which is to be services by a technician. They can verify accessibility by scripting the movements of a "virtual technician" to reach the appropriate parts of the engine without hitting the other parts. They need only know whether the technician is able to reach the required components. Eventually, such automated accessibility tests may save resources from being spent on unnecessary preliminary mock-ups in industrial design settings.

Distance computation is important for certain path-planning algorithms, which attempt to move an object from an initial position to a goal position, while otherwise maximizing the distance from the object and surrounding obstacles. This is a common task addressed by the robotics community. Another example is when a robot is to be moved to a specifc placement, but must observe a zone of safety in which it can approach no closer than a given distance to obstacles in its environment.

The actual intersection computation is important for physically-based modeling and animation systems which must know all contacts in order to compute an appropriate collision response in their simulations. Similarly, the ETA solution permits a simulation to control the time step in a simulation so as to minimize the number of unnecessary simulation steps.

3.2 Pair Processing vs. Nbody Processing

Sometimes the problem posed involves only a pair of models, such as the virtual technician and the assembly in a previous example, or the robot and its environment. This is sometimes called "pair processing."

Sometimes the problem involves many different parts, where contacts among some or all need to be found. This is called "nbody processing," in reference to the classic problem in celestial mechanics (many bodies moving under mutual gravitational influence).

The contact form of the nbody problem is usually not of interest: is there contact among n given bodies? Generally we want to know *which* of the $O(n^2)$ pairs of bodies are in contact – so this is an intersection reporting problem. The distance form of the nbody problem is to report the distances between all $O(n^2)$ bodies, but it is often better to report distance among pairs which have come closer than some given distance – and it is not always necessary to measure the distance to determine whether or not that condition has been met. As mentiond previously, the ETA solution can be very useful for efficient dynamics simulation, especially when large numbers of objects take part in the simulation.

Hubbard [Hubbard93] describes the nbody problem as the "broad phase", in which overlapping pairs of bounding boxes for the objects are located. Most systems which address the nbody problem will perform this "broad phase" step, and then apply pair processing algorithms to resolve the collision detection query, which Hubbard calls the "narrow phase".

3.3 Temporal Coherence and Motion Description

Often collision detection is employed in an animation system, or a physically-based modeling application. Frequently, the collision detection task is executed repeatedly on the same models in the same environment, but subject to rotations and translations (or possibly nonrigid transformations) at successive time steps. In this case, the task at each step may be very similar to the previous step, if the motion between steps is slight. Algorithms which can capitalize on this fact are said to be exploiting temporal coherence.

In order to exploit temporal coherence, some algorithms must be given bounds on the motions of the objects. Other algorithms must be given a closed-form expression of the motion as a function of time. Some algorithms require no information on the motion, but merely require the placements of the objects at successive time steps.

3.4 Deformable Models

When the component of time is introduced, there is also the possibility that the models deform over time. Assuming that the deformations between steps are small, some algorithms may be able to exploit temporal coherence in this case as well. Most algorithms, however, preprocess the models with the assumption that they will not deform, and will be subject only to rigid transformations (plus uniform scaling, in some cases).

4 Fields Concerned with the 3D Collision Detection Problem

There are primarily four fields which are addressing the 3D collision detection problem in one form or another: computational geometry, computer graphics, robotics, and computer aided geometric design. Each field addresses a different part of the problem domain, and each has its unique approach.

4.1 Computational Geometry

Computational Geometry derives from both geometry for the rigorous study of geometric objects and from computer science for the study of algorithmic complexity of operations on those objects. Papers on the collision detection problem focus on rigorous analysis of asymptotic complexity both in time and space (memory) requirements.

Problems in the Computational Geometry literature are precisely posed, and the results obtained carry the weight of mathematical proof. That is, the results obtained in this field are objectively true. In terms of asymptotic performance, algorithms described in different papers may be meaningfully compared: any sub-linear algorithm is considered superior to any super-linear algorithm, for instance. This is the great strength of Computational Geoemtry as compared to the other fields discussed here.

The weakness of Computational Geometry as regards commercial enterprise concerns absolute (not asymptotic) performance, robustness issues for practical problems, and ease of algorithm implementation.

Algorithms are usually reported in terms of asymptotic performance, but absolute time (in seconds) required or actual space (in bytes) is often crucial in the design of commercial systems. Even if a given algorithm has superior growth complexity, it may be unusable because of the magnitude of the lower order terms in performance or memory usage.

"Robustness" concerns the assumptions of perfect arithmetic and "general position" of input – neither of which can be guaranteed in many practical applications. Real applications very frequently violate the general position assumption, and nearly nongeneric arrangements tend to accentuate the inadequacies of imperfect arithmetic.

The last issue, ease of implementation, arises from the fact that Computational Geometry is generally more concerned with algorithmic design than with implementation issues. Some of the most efficient algorithms are extremely intricate, and exceedingly difficult to implement.

4.2 Computer Graphics

Computer Graphics refers to the broad field that covers the craft and science of graphical imaging with computers. It's strengths and weaknesses are a mirror image of those of Computational Geometry.

Among the four fields discussed in this paper, Computer Graphics is by far the most concerned with ease-of-implementation and "practical algorithms". However, solutions to problems are often ad-hoc, and frequently conclusions are drawn on the basis of scant experimental evidence and little or no rigorous analysis.

4.3 Robotics

The robotics community is concerned with the collision detection problem because they want to know in advance where it is safe to move a robot. Collision detection is also an important subproblem in the path-planning task. The robotics papers tend to be more rigorous than the computer graphics papers, but less rigorous than the computational geometry papers. Sometimes they include asymptotic analysis, and sometimes not. The algorithms have usually been implemented and tested. The robotics The robotics papers often focus more on robustness issues than those of the other two fields. Consideration is often given to the finite precision of machine arithmetic.

4.4 Computer Aided Geometric Design

Computer Aided Geometric Design (CAGD) is concerned exclusively with representations of and operations on curved surfaces in space. It is primarily a mathematical discipline with a strong component in implementation issues. Frequently finite precision arithmetic and error propagation are a concern of CAGD papers. Collision detection problems arise in the CAGD community in the form of the intersection problem: what is the intersection of two curved parametric surfaces?

5 Survey of Collision Detection Techniques

Now that we have reviewed the domain of the basic collision detection problem, the types of models on which collision detection might be performed, as well as the contributing fields, we will survey some of the techniques which been used to solve it.

5.1 Pair Processing

We will begin with the various pair processing techniques, which are concerned with just two input models.

5.1.1 CSG Objects

Since CSG objects exist in a framework of set operations, the intersection problem is conceptually trivial. To test if two object are touching, simply take their set intersection. If the result is the null set, then the objects are disjoint, otherwise they touch.

What is generally needed is efficient ways to compute whether an intersection yields the empty set or not. This is somtimes called the "Null Object Detector", or NOD.

5.1.1.1 S-Bounds

Cameron [Cameron91] introduced "S-Bounds" as a means of speeding up intersection evaluation.

As described earlier, points in space can be applied to the expression tree of a CSG object in order to yield the "in", "out", or "on" classification of the point. The classical evaluation of the intersection operation generates a large number of sample points which must be classified – this set of points is sufficient to determine whether the intersection is empty or not. The S-Bounds are intended to restrict the scope (spatially) of the sample points, but which is still sufficient to determine emptiness of the intersection.

Cameron found that using S-Bounds increased the speed of intersection testing by one to two orders of magnitude on his CSG system.

5.1.1.2 Interval Arithmetic and CSG of Implicit Functions

Duff [Duff92] employs interval arithmetic to evaluate implicit functions over box-like regions of space to determine whether the regions lie entirely inside, entirely outside, or potentially laying across the boundary of the implicit surfaces. This is the familiar point classification scheme extended to regions.

Interval arithmetic is a formal method for obtaining conservative bounds on the range of values a function can take on given a range of inputs. An interval consists of a lower bound and an upper bound, denoted as, $[u, l]$. Arithmetic operations form new intervals from the components of the given intervals. We are guaranteed that the new interval will contain the result of the operation given two ordinary numbers taken from the operand intervals.

Here is a simple example of adding two intervals: $[3, 6] + [1, 2] = [4, 8]$. Our guarantee is that if we take a real number from each interval, say, $x \in [3, 6]$ and $y \in [1, 2]$, then $x + y \in [4, 8]$. More generally, if we have intervals A and B, then $A \otimes B$ is an interval such that if $a \in A$ and $b \in B$ then $a \otimes b \in A \otimes B$. For a more in depth look at interval analysis see Moore [Moore79].

Interval arithmetic can be extended seamlessly into vector interval arithmetic – where the interval vectors are the box-like regions mentioned above. If the vector interval, representing the box-like region, evaluates to an entirely positive range (both lower and upper bounds are greater than zero), that means the implicit function is positive everywhere in that region (because the value of the implicit function is trapped within the interval everywhere in the region). Similarly, if it evaluates to a purely negative interval, then the region is entirely outside the implicit surface. The the interval contains zero, then the region may or may not lay across the boundary of the box – recall that the range returned is conservative, so the exact range of the function may be close to zero, but the reported interval may have conservatively expanded to include the zero.

Duff's scheme for performing the collision detection among n CSG objects involves evaluating the n CSG expressions over the given regions, and then simplifying the expressions using special rewriting rules. If the expression has not reduced to a trivial quantity (indicating that the region does not with certainty lie entirely inside or outside the implicit surface), then subdivide the region into two subregions, and recursively repeat the process, evaluating the simplified expressions over each subregion. As the recursions proceed, the expressions become simpler and simpler, while the regions become smaller and smaller.

In this manner, the regions of interest become arbitrarily small until they fall entirely inside or outside a CSG object, at which time its expression is dropped from the list for that recursion branch. Whenever we find that the region falls entirely within a CSG object, the object is kept on the list, but no longer evaluated for this recursion branch (because further evaluations will be over subregions, which we know in advance must be entirely covered by the model). Whenever we find that only one CSG expression (or none) remains in our list, then we conclude that no intersection has occurred within that region, and we terminate that recursion branch. If a region is eventually reached which is contained entirely by two objects, then those objects intersect. Otherwise, we recurse until we reach the desired precision.

In this manner, Duff tackles the nbody problem without resorting to the two distinct phases, "broad" and "narrow," as Hubbard [Hubbard93] calls them, which all the other nbody systems use.

This method is adaptive subdivision over the space in which the models are embedded, and the precision of the results is limited to the fineness of the subdivision. Allowing for the finite precision of the method, it is an extremely robust and concise formulation of the problem, as well as easy to implement. It is also a very expensive method which cannot produce real time collision detections on realistic models with the current computing hardware.

Duff also extends his method to the ETA problem when closed-form solution of the motions of the objects are known. In this case the implicit surfaces actually because implicit functions of x, y, z, and t. The regions occupy hypercube-like regions of space-time, and the subdivision occurs as before, but directed so as to favor locating the collision with the earliest t beyond the present. Thus, the next collision can be found.

5.1.2 Quadric Surfaces

Quadric surfaces somewhat special, as mentioned before, because they can represent cones, slabs, spheres, and cylinders within a unified framework. These are commonly used shaped in CAD systems. All quadrics can be represented as either implicit or parametric surfaces.

Levin [Levin79] introduced methods for representing quadric surfaces and for finding their intersection curves. But his methods required the use of fourth degree polynomials, which can be numerically sensitive. Miller [Miller87] improved on these techniques, enabling use of polynomials of only degree 2 or less.

It often arises that quadric surfaces are aligned so that their intersection is not a general fourth degree space curve (as it generically would be). When this occurs, degeneracies and rank-deficiencies cause conventional methods to fail. Farouki [Farouki89] introduces techniques to determine whether the configuration is nongeneric and if it is, what the structure of the intersection curves are.

5.1.3 Pairs of Parametric Surfaces

There are fundamentally 4 methods for finding the intersection of two parametric surfaces: subdivision methods, lattice methods, tracing methods, and analytic methods. Many practitioners actually use some combination of these. A survey of these techniques is given by Patrikalakis [Patrikalakis93].

5.1.3.1 Subdivision Methods for Parametric Surfaces

All subdivision methods for parametric surfaces work by recursively subdividing the domain of the two surface patches in tandem, and examining the spatial relationship between patches. In all cases, depending on various criterion, the patches domains are further subdivided and recursively examine, or the given recursion branch is terminated. In all cases, whether it is the intersection curve or the distance function, the solution is known only to finite precision, according to how finely the domain has been subdivided and how it maps into space.

5.1.3.1.1 Subdivision of Domain with Interval Arithmetic

There are actually several approaches to the parametric subdivision approach to the problem. One of the most elegant is presented by John Snyder [Snyder92], who uses interval arithmetic to recursively subdivide the domain of the surface to refine an approximation of the intersection region until it is of satisfactory precision. This is *very* similar to the method Duff [Duff92] uses with implicit surfaces, but Snyder evaluates the parametric mapping $f : R^2 \mapsto R^3$ over regions in the domain to obtain box-like regions in the range, that is, space.

Snyder begins with two regions covering the entire domains of two parametric surfaces. He then alternately subdivides one or the other, and recursively checks the bounding boxes for overlap in space. If overlap does not occur, he abandons that recursion branch. If overlap occurs, he continues subdividing. Unlike Duff's method, Snyder is never absolutely certain that intersection has occured – only that the patches have come to within a given distance of one another, depending on the size of the touching boxes. Continuing the process to any desired resolution, the preimage of the intersection curve is mapped out in rectangles (regions) in each surface's domain, and the intersection curve is approximated as a string of boxes in space.

Snyder also shows how the synchronous subdivision of two parametric surface domains can yield the minimum distance between them. Again, the subdivision proceeds as before, but the two boxes (which are the images of two regions in the domains) are used to produce a maximum and minimum possible range of distances that points between them can have, and this distance interval is added to a priority queue of distance intervals. Also added to the queue are the two domain regions which produced it. Elements are taken from the queue according to minimum lower range and the regions are subdivided, processed, and more elements are added to the queue. Distance intervals whose lower bound is greater than the lowest upper bound in the queue can be discarded, since these region pairs will never contribute to the shortest distance between the surfaces.

5.1.3.1.2 Subdivision of Domains of Time-Varying Parametric Surfaces

A parametric surface need not be just a function of a (u, v) parameter patch. The mapping can include time, $f : (u, v) \times t \mapsto R^3$. In this case, the dependence on time can reflect motion of the object, as well as deformations with time.

Herzen, Barr, and Zatz [Herzen90] subdivide the domain of the surface, including the time dimension, similar to Snyder. However, Herzen et al. do not use interval arithmetic to compute bounds on where the surface maps to – instead they use Lipschitz Conditions, which are bounds on the various derivatives of the mapping. Just as Snyder used a priority queue to retrieve the closest possible box pairs to find minimum distance, Herzen et al. use a priority queue to retrieve the earliest possible overlapping boxes.

Hence, they have an ETA algorithm for deformable parametric surfaces, which is robust and accurate to any desired precision. As with the interval arithmetic approach, it is very time-consuming, and not suitable for intensive collision detection tasks.

5.1.3.1.3 Improved Interval Arithmetic Methods for Deformable Surfaces

In 1993, Snyder and Woodbury [Snyder93] improved upon the work of Herzen et al. by introducing more conditions with which to prune the search space for collisions.

They added a tangency condition, which states that at the moment of contact, two curved surfaces must have a point in common but with opposing normals (that is, the surfaces must be tangent). He also stated that the converging points on the two surfaces must be moving toward each other.

Consequently, when considering a pair of domain patches for potential collision, they can be eliminated from consideration not only if they don't overlap spatially, but also if they don't contain opposing normals, or if they don't contain converging components in their respective velocity intervals (the derivatives are also interval-valued functions). This helps speed convergence to the solution set.

5.1.3.2 Lattice Methods

The intersection curve in space has a preimage curve on the domains of both patches. Lattice methods attempt to locate specific points of these preimages by selecting many isoparametric curves (where a u- or v-parameter is held constant) which criss-cross the surface like a lattice-work. Selecting a value for u on one of the patches reduces the dimensionality of the search space to one – our only free variable is v on that patch.

So, we can express the points of surface intersection as the zeroes of a function of v. An analysis of the degree of the polynomial and the derivatives allow us to perform root trapping techniques to robustly find where the intersection preimage meets the isoparametric curve.

The difficulty with this method is that the intersection curve can be a very small closed loop which is easily missed by rows of isoparametric curves.

Frequently, lattice methods or subdivision methods are used to find starting points for use by the tracing methods.

5.1.3.3 Tracing Methods

The tracing method begins with a given point known to be on the intersection curve. Then the intersection curve is traced in sufficiently small steps until the edge of the patch is found, or until the curve returns to itself to close a loop.

While it is easy to check for meetings with a patch boundary, it is difficult to know when the tracing point has returned to its starting position, as it requires the use of some arbitrarily chosen tolerance.

Frequently this is posed as an initial-value differential equations problem. At the intersection point on the surfaces, the intersection curve must be mutually orthogonal to the normals of the surfaces. Consequently, the vector field which the tracing point must follow is given by the cross product of the normals.

Barnhill et al. [Barnhill87] use the isoparametric method for finding starting points, and then use the tracing method described above.

Also, Barnhill and Kersey [Barnhill90] discuss improved tracing methods, and Mullenheiim [Mullenheim91] discusses a method for finding starting points for tracing.

5.1.3.4 Analytic Methods

Analytic methods usually involve implicitizing one of the parametric surfaces – obtaining an implicit representation of the same shape. The parametric surface is a mapping from (u, v)-space to (x, y, z)-space, and the implicit surface is a mapping from (x, y, z)-space to the real numbers.

By substituting the parametric functions $f_x(u, v), f_y(u, v), f_z(u, v)$ for the x, y, z of the implicit function, we obtain a scalar function in u and v. The locus of roots of this scalar function map out curves in the (u, v) plane which are the preimages of the intersection curve.

The hope is that the scalar function can be examined to learn about its topology or other structure, to aid in obtaining the root locus.

The above process is described by Goldman [Goldman85].

Hoffman [Hoffman85] provides an excellent tutorial on the subject of implicitization and substitution, in general.

5.1.4 Pairs of Convex Polytopes

The most highly structured problem for two polygonal models is one where both models are convex polytopes with $O(n)$ sides each.

5.1.4.1 Testing Contact Status By Linear Programming

The detection problem can be converted to a linear programming problem, in which we seek a feasible solution to a set of linear constraints. The feasible solution is a plane, and the constraints are that the vertices of one polytope all lie on the opposite side of the plane from the vertices of the other polytope.

Suppose the n vertices of polytope U are given in $u_1, u_2, \ldots, u_n$ and the m vertices of polytope V are given in $v_1, v_2, \ldots v_m$. We can define a plane with the equation $f(x, y, z) = ax + by + cz + d = 0$, and we are looking for the parameters a, b, c, and d which yields a plane which partitions the polytopes, that is, for which,

$$\begin{array}{rcl} f(u_1) & < & 0 \\ f(u_2) & < & 0 \\ \vdots & & \\ f(u_n) & < & 0 \\ f(v_1) & > & 0 \\ f(v_2) & > & 0 \\ \vdots & & \\ f(v_m) & > & 0 \end{array}$$

This is the feasible solution to linear constraints problem, which is a special case of a linear programming problem (the objective function is simply ommitted). There is a feasible solution if and only if there is contact. Seidel [Seidel90] gives an easily implemented randomized algorithm for linear programming, which generaly runs in linear time, although this is not a strict bound.

5.1.4.2 Space Sweep for Intersection Problem

Muller and Preparata [Muller78] find an $O(n \log n)$ algorithm for computing the intersection of two polyhedra (if the intersection exists, at all).

They show how to find a point within the intersection of the polytopes in $O(n \log n)$ time, if it exists. If it does not exist, then this step aids in the production of a separating plane in $O(n \log n)$ time.

If the common point is found, then the duals of the two polytopes are taken with respect to the point, the convex hull of the duals (taken together) is computed in $O(n \log n)$ time, and then the dual of the hull is taken, which is the intersection of the polytopes. No step takes worse than $O(n \log n)$ time.

5.1.4.3 Decomposition into Drums and Binary Search

A faster algorithm to determine contact status over multiple queries is given by Dobkin and Kirkpatrick [Dobkin82]. The drum data structure (described below) requires $O(n)$ space, and there may be $O(n)$ drums for the polytope, so that is $O(n^2)$ space and time required for preprocessing.

The first step is to slice each polyhedron through all its vertices with a series of parallel planes. This chops the polyhedron into a stack of "drums". The polyhedron contact test involves a binary search on the two stacks of drums in which two drums are tested for contact at each step. If the drums do not touch, then half half of one of the polyhedra can be eliminated from consideration. So, $O(\log n)$ tests are required to reduce the problem to the last two drums, one on each polyhedron. Each drum test requires $O(\log n)$ time, so the overall time is $O(\log^2 n)$.

5.1.4.4 Distance Computation by Traversing the Miknowski Polytope

If the distance between the polytopes is required, Gilbert et al. [Gilbert88] describe an implementation which they claim is approximately linear in the total number of vertices. A strong asymptotic analysis is not given, as they estimate the complexity experimentally.

The basic approach is as follows: the Minkowski difference of the polytopes is a convex polytope M, and the distance between the polytopes is the distance of the origin from M. Hence, M contains the origin if and only if the original polytopes intersect.

The algorithm proceeds by walking an approximating simplex (formed from a subset of the vertices of M) from some arbitrary starting position to eventually cover M's closest point to the origin. When the simplex finally covers the closest point, then the simplex's distance to the origin equals M's distance to the origin, which is the separation between the original polytopes.

No preprocessing time is required for this algorithm.

5.1.4.5 Distance Computation over Multiple Queries

Convex polytopes have the important property that their closest points can be verified by an analysis of those points and their local neighborhoods on their respective polytopes. This is noted in the PhD theses of both Lin [Lin93] and Baraff [Baraff92] and is used to exploit temporal coherence in successive distance queries between convex polytopes. The idea is that with successive time steps, the polytopes' closest points will move only a small distance on the surface of their respective polytopes (perhaps only to neighboring facets). Given this, it is possible to compute which direction to walk each of the previously closest points so as to get closer in the current configuration. The walk continues until the points satisfy the closest point conditions, which take only constant time to evaluate since the conditions are local. With small motions between time steps, the walks will be short, of length $O(1)$, leading to a constant time algorithm for successive distance queries. Cameron [Cameron96] later showed that the feature-pairs walk of Lin and the simplex walk of Gilbert et al. are essentially identical, and shows how a modification of the simplex walk can achieve similar $O(1)$ complexity with temporal coherence.

5.1.5 Polygon Soups

If the polygons defining the models have no particular structure, then they are called polygon soups. The most obvious approach is to test each polygon in the first model against each polygon in the second model. This requires $O(n^2)$ tests. All of the techniques to follow are designed to reduce the number of tests necessary to confirm contact status. They all embed the model within some kind of search structure which permits us to trivially reject a large number of potential polygon pair intersection tests.

5.1.5.1 Hierarchical Spatial Subdivisions (HSSs)

Hierarchical Spatial Subdivisions are divide-and-conquer in the spatial domain, that is, adaptive subdivision of the set of points in the embedding space (3-space) of the model.

5.1.5.1.1 Octrees and Use of HSSs

Possibly the most common spatial subdivision is the octree. The entire model is placed within a cube. The cube is then divided into eight equal-sized subcubes, and any subcube which touches too many polygons is recursively further subdivided. The recursion terminates when the leaf subcubes hold some appropriately small number of polygons, or until some maximum depth is reached. Each leaf node holds a list of polygons which touches it.

The query proceeds as follows: given the placements of two models, the top level cubes are tested for overlap. If they do not overlap, then intersections among the models are not possible, and we are done. If they do overlap, then the eight children of the first cube are tested against the eight children of the second cube. This is 64 new cube-cube overlap tests. Those which overlap lead to more tests. Whenever a leaf cube is reached, we test the polygons it contains against the other object. So eventually we are testing polygons against polygons, but hopefully we are testing less than $O(n^2)$ polygon pairs.

This is a heuristic, and there are inputs for which the worst case performance of $O(n^2)$ occurs, but these are unusual cases which rarely occur in practice, if at all.

5.1.5.1.2 Spheres Bounding Octree Nodes

Palmer and Grimsdale [Palmer95] take a variation of this approach, where instead of testing two subcubes for overlap, they test bounding spheres of those subcubes for overlap. This is a more conservative test, and hence may cause more intersection tests later, but these tests are less expensive than the cube/cube tests, so there may be a benefit.

Hubbard [Hubbard93] also used hierarchies of spheres centered on octree nodes. In a later work [Hubbard95], Hubbard creates hierarchies of spheres organized by medial axes or "skeleton" of the model, which generally produces a tighter covering using even fewer spheres.

5.1.5.1.3 k-d Trees

Held et al. [Held95] investigated, among other structures, a k-d tree, which can either be considered a relaxation of an octree, or a special case of a BSP tree. This was compared alongside three other methods (R-trees, tetrahedral mesh, and uniform grid), and was found to be the least effective in the trials they conducted.

5.1.5.2 Bounding Volume Hierarchies (BVHs)

The bounding volume hierarchy is essentially used the same way that the cells of a spatial partition are used: if two volumes overlap, then their children are tested against each other pairwise, until the leaf nodes containing the polygons are reached, at which time the polygons themselves are tested. However, bounding volume hierarchies partition the polygons groups, rather than partitioning the space. No polygon assigned to a bounding volume will extend beyond the volume (or any of that olume's ancestors in the hierarchy).

A typical top-down construction begins with fitting a bounding volume to the entire model. Then the enclosed polygons are (conceptually) assigned red or blue colors, and the two sets are each fitted with its own bounding volume. These two sibling bounding volumes may overlap one another, may spatially cover polygons belonging to the other volume, and may even extend beyond the boundaries of the parent volume.

Among the many design choices of BVHs are choice of bounding volume. A bounding volume of simple form, such as a sphere, will be inexpensive to test for overlap. On the other hand, a more complex bounding volume, such as a convex hull, may yield a more expensive overlap test, but since it might cover the geometry more tightly, fewer overlap tests may be required in order to determine noncontact status. The optimum among the tradeoffs is not at all obvious, and there is some disagreement among practitioners as to the best methods.

5.1.5.2.1 R-Trees

One of the structures which Held et al. [Held95] investigated was R-trees. These are simply axis-aligned bounding boxes, built top-down as described above. Of the four methods, this was found the most effective.

5.1.5.2.2 OBBTrees

Gottschalk et al. [Gottschalk96] use bounding volumes which are boxes oriented to fit the bounded geometry. Compared to axis-aligned bounding boxes (or R-trees), it is not known whether this method is better or worse for most circumstances. It was suggested (but not proven) that for two sufficiently highly tesselated smooth surfaces which are parallel and separated by some small gap, that oriented bounding boxes can determine overlap status with asymptotically less work that axis-aligned bounding boxes.

5.1.5.3 Nonhierarchical Spatial Partitionings (NSPs)

Some spatial partitionings such as uniform rectilinear grids or tetrahedralizations of space have been investigated. Their advantages are that the single layer permits constant-time access to the cells, which typically hold the polygons defining the environment of obstacles for a moving object.

5.1.5.3.1 Tetrahedral Mesh of Free Space

Held et al. [Held95] took the novel approach of tetrahedralizing the free space of the environment with a partitioning of low stabbing number. As the object (a collection of triangles) moved through the space, the triangles composing the object were walked from their starting tetrahedron to their final tetrahedron. The problem resembles a ray-shooting task, except instead of shooting rays, they shoot triangles. In their paper, Held et al. use a convex object and the environment is a connected mesh of triangles but their algorithm does not rely on either of these properties. It could apply to polygon soups as well.

5.1.5.4 Uniform Grids

In that same paper, Held et al. examined the uniform grid, in which cells kept lists of the environment's polygons they touched. For each of the polygons in the moving object, the grid cells which touch it are examined, and the polygons in the cell lists are checked for intersection. The value of a grid is that access to a cell is direct and constant time, regardless of how many grid cells there are. But there is a balance to be achieved: if the grid cells are too small, then a given object triangle touches many of them, and a lot of lists must be traversed. Furthermore, a given environment triangle which covers several grid cells may be tested against the object triangle multiple times. This can be alleviated by taking a courser grid, but in that case the list of a very large cell will include environment triangles which aren't even nearby the moving object. For the models they tested, they found the best grid resolution ranged between $5 \times 5 \times 5$ and $50 \times 50 \times 50$.

Garcia-Alonso et al. [Alonso94] also use a rectilinear grid to eliminate some of the polygon overlap tests. Each model is covered by a bounding box (which is oriented with the model), which is further subdivided into a regular grid of predetermined resolution. Grid cells hold a list of polygons which touch the cell. For a given placement of the models, they compute in constant time the smallest rectangular subgrid of each box covered by the other box. They then test each grid cell in one subgrid against every grid cell in the other subgrid, and for those pairs which overlap, they test each contained polygon in the one against every contained polygon in the other. In other words, they perform all-pairs testing on a restricted set of cells. Their grids were organized so that they contained 512 cells each (they experimented with different cell counts, and found 512 to be best).

Their hope is that the subgrids will be small as objects graze each other and bounce away. But if the objects substantially overlap, then the subgrids will be close to full size. This can happen, for instance, when one object is the polygons defining the hydraulic system in a submarine, and the other object is the system electrical conduits. Both are systems of pipes, essentially, which are fixed to the submarine hull, and they are intermingled, structurally, even if never actually in contact. With the boxes completely overlapping, a quarter-million pairs of grid-cell overlap tests will be performed.

This is a worst case scenario for their approach which can happen in practice for some real applications.

5.2 Nbody Processing

Many simulations track many objects and must determine when contacts occurs. The typical way this is approached is as Hubbard [Hubbard93] described it: during a "broad phase" which determines which pairs of objects are in close proximity, and for those pairs, a "narrow phase" in which the pair undergoes a more thorough examination.

There is a wide variety of techniques applied to the "broad phase" problem. Some techniques are analytic, some focus on temporal coherence, and some use a scheduling algorithm.

5.2.0.1 Analytic Techniques

Hopcroft et al. [Hopcroft83] presented a $O(n \log^2 n)$ method for determining whether any pair among n spheres are touching, but said nothing about how to report which spheres were touching.

Edelsbrunner [Edelsbrunner83] presented a method which could report the intersecting pairs among n axis-aligned boxes in $O(n \log^3 n + k)$ time, where there are k pairs, requiring $O(n \log n)$ space.

Six [Six82] reports an algorithm requiring $O(n \log^2 n + k)$ time and $O(n \log^2 n)$ space. So, relative to Edelsbrunner's approach, they have traded space for time.

These techniques establish upper bounds on the difficulty of the nbody processing task.

5.2.0.2 Space-Time Bounds for Adjusting Time Steps

Hubbard [Hubbard93] uses bounds on the possible acceleration of objects to constrain their potential motion. Knowing such bounds on the motion, he is knows that the location of an object in space-time is constrained to an open-ended parabolic horn – because the further the object slips into the future, the greater its uncertainty of position. The earliest possible collision can occur at the smallest value of t at which two horns touch in space time.

Computing when this occurs is difficult, so Hubbard bounds the horns with hyper-trapezoids, which are much more tractable. He computes the earliest value of t at which there can be a collision, and is able to make a time step to that moment.

5.2.0.3 Insertion Sort for Incremental Adjustments of Bounding Boxes

The broad phase of the I-COLLIDE system presented by Cohen et al. [Cohen95] uses an insertion sort along three axis to track which boxes are overlapping. Two boxes overlap only if they overlap along all three axes. If the boxes' positions are shifted only slightly between time steps, then their ordering along the axes will change only a small amount. The amount of work done during this step is $O(n + e_x + e_y + e_z)$, where the $O(n)$ term comes from the adjustment of each boxes' coordinates, and the remaining terms are the number of times any box switches places with another along the three axes. It is expected that in sparse environments, these quantities will be small.

6 Conclusion

This completes the survey of collision detection techniques applied to the major model representation types.

7 Bibliography

[Alonso94] A. Garcia-Alonso, N Serrano, J. Flaquer. Solving the Collision Detection Problem. IEEE Computer Graphics and Applications, Vol. 14, pp36-43, 1994.

[Baraff92] D. Baraff. Dynamic Simulation of Non-Penetrating Rigid Body Simulation. PhD Thesis, Cornell University, 1992.

[Barnhill87] R. E. Barnhill, G. Farin, M. Jordan, B. R. Piper. Surface/surface intersection. Computer-Aided Geometric Design, Vol. 4, No. 1, pp. 3-16, 1987.

[Barnhill90] R. E. Barnhill, S. N. Kersey. A marching method for parametric surface/surface intersection. Computer-Aided Geometric Design, Vol. 7, No. 4, pp. 257-280, 1990.

[Cameron91] S. Cameron. Approximation hierarchies and S-bounds. In Proceedings of the Symposium on Solid Modeling Foundations and CAD/CAM Applications, pp 129-137, Austin, TX, 5-7 June, 1991.

[Cameron96] S. Cameron. A Comparison of Two Fast Algorithms for Computing the Distance between Convex Polyhedra. Submitted to IEEE TR&A as a comminucations paper, July 1996.

[Chaz87] B. Chazelle, D. P. Dobkin. Intersection of Convex Objects in Two and Three Dimensions. Journal ACM, Vol 34, No 1, pp 1-27, 1987.

[Cohen95] J. D. Cohen, M. C. Lin, D. Manocha, M. K. Ponamgi. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments. Proceedings of ACM Interactive 3D raphics Conference, pp. 189-196, 1995.

[Dobkin83] D. Dobkin, D. G. Kirkpatrick. Fast detection of polyhedral intersection. Theoretical Computer Science, Vol. 27, No. 3, pp241-253

[Duff92] T. Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. Computer Graphics Proceedings. SIGGRAPH '92 pp. 131-138.

[Edelsbrunner83] H. Edelsbrunner. A new approach to rectangle intersections. Part I. International Journal of Computer Mathematics, Vol 13, No. 3-4, pp. 209-219, 1983.

[Farouki89] R. T. Farouki, C. A. Neff, M. A. O'Connor. Automatic parsing of degenerate quadric-surface intersections. ACM Transactions on Graphics, Vol. 8, No. 3, pp. 174-203, 1989.

[Gilbert88] E. G. Gilbert, D. W. Johnson, S. S. Keerthi. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space. IEEE Journal of Robotics and Automation, Vol 4, No. 2, pp. 193-203, 1988.

[Goldman85] R. N. Goldman. The method of resolvents: A technique for the implicitization, inversion, and intersection of non-planar, parametric rational cubic curves. Computer-Aided Geometric Design, Vol. 2, No. 4, pp. 237-255.

[Gottschalk96] S. Gottschalk, M. C. Lin, D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. Computer Graphics Proceedings. SIGGRAPH '96 pp. 171-180.

[Held95] M. Held, J. T. Klosowski, J. S. B. Mitchell. Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs. Prceedings of the 7th Canadian Conference on Computational Geometry, pp 205-210, Quebec, Canada, Aug 10-13, 1995.

[Hertel84] S. Hertel, M. Mantyla, K. Mehlhorn, J. Nievergelt. Space Sweep Solves Intersection of Convex Polyhedra. Acta Informatica, Vol. 21, No. 5, pp. 501-519, 1984.

[Hoffman89] C. Hoffman. Geometric and Solid Modeling. Morgan Kaufmann, 1989.

[Hopcroft83] J. E. Hopcroft, J. T. Schwartz, M. Sharir. Efficient detection of intersections among spheres. Planning, geometry and complexity of robot motion, pp. 240-244, 1987.

[Hubbard93] P. M. Hubbard Interactive Collision Detection. In Proceedings of the 1993 IEEE Symposium on Research Frontiers in Virtual Reality, pp 24-31, San Jose, CA, 25-26 Oct, 1993.

[Hubbard95] P. M. Hubbard. Collision Detection for Interactive Graphics Applications. IEEE Transactions on Visualization and Computer Graphics, Vol 1, No. 3, 1995.

[Levin79] J. Z. Levin. Mathematical models for determining the intersections of quadric surfaces. Computer Graphics and Image Processing, Vol. 11, No. 1, pp. 73-87, 1979.

[Lin93] M. C. Lin. Efficient Collision Detection for Animation and Robotics. PhD Thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1993.

[Miller87] J. R. Miller. Geometric approaches to nonplanar quadric surface intersection curves. ACM Transactions on Graphics, Vol. 6, No. 4, pp. 274-307, 1987.

[Moore79] R. E. Moore. Methods and Applications of Interval Analysis. Society or Industrial and Applied Mathematics (SIAM), Philadelphia, 1979.

[Mullenheim91] G. Mullenheim. On determining start points for a surface/surface intersection algorithm. Computer-Aided Geometric Design, Vol. 8, No. 5, pp. 401-408, 1991.

[Muller78] D. E. Muller, F. P. Preparata. Finding the intersection of two convex polyhedra. Theoretical Computer Science, Vol. 7, No. 2, pp217-236, 1978.

[Palmer95] I. J. Palmer, R. L. Grimsdale. Collision Detection or Animation using Sphere-Trees. Computer Graphics Forum, Vol. 14, No. 2, pp105-116, 1995.

[Patrikalakis93] N. M. Patrikalakis. Surface-to-surface intersections. IEEE Computer Graphics and Applications, Vol. 13, No. 1, pp. 89-95, 1993.

[Seidel90] R. Seidel. Linear programming and convex hulls made easy. In Proc. 6th Annual ACM Conf. On Computational Geometry, pp 211-215, Berkeley, CA, 1990.

[Six82] H. W. Six, D. Wood. Counting and reporting intersections of d-ranges. IEEE Transactions on Computers, Vol. C-31, No. 3, pp. 181-187, 1982.

[Welzl91] E. Welzl. Smallest enclosing disks (balls and ellipsoids). Procedings of New Results and New Trends in Computer Science, pp359-370, Graz, Austria; 20-21 June 1991.

Table of Contents