Collision detection between geometric models: a survey

Ming C. Lin & Stefan Gottschalk

University of North Carolina

Abstract

In this paper, we survey the state of the art in collision detection between general geometric models. The set of models include polygonal objects, spline or algebraic surfaces, CSG models, and deformable bodies. We present a number of techniques and systems available for contact determination. We also describe several N-body algorithms to reduce the number of pairwise intersection tests.

1 Introduction

The goal of *collision detection* (also known as interference detection or contact determination) is to automatically report a geometric contact when it is about to occur or has actually occurred. The geometric models may be polygonal objects, splines, or algebraic surfaces. The problem is encountered in computer-aided design and machining (CAD/CAM), robotics and automation, manufacturing, computer graphics, animation and computer simulated environments. Collision detection enables simulation-based design, tolerance verification, engineering analysis, assembly and dis-assembly, motion planning, animated figure articulation, walkthrough, etc. All these tasks involve contact analysis and spatial reasoning among static or moving objects. In many of these application areas, collision detection is considered a major computational bottleneck.

Collision detection is an essential component of robot motion planning and control, where it helps to steer the robot away from its surrounding obstacles. In virtual prototyping, it is used to refine designs without productions of physical prototypes in the initial design stage. Collision detection can also be a significant aid in simulation for engineering analysis. Experiments which are impractical to conduct can be simulated, such as support design for tunnels. Another example is automobile crash test and ergonomics analysis which can be done systematically at much lower

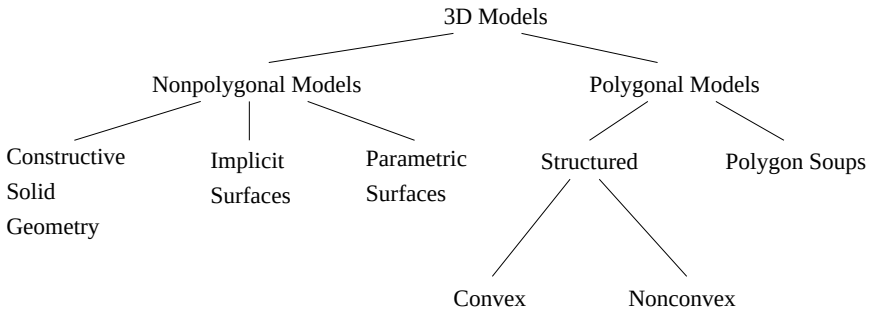

FIGURE 1. A Taxonomy of 3D Model Representations

cost and under more controlled, varied conditions using collision detection algorithms in a computer simulated environment.

In this paper, we survey the state of the art in collision detection between geometric models represented by a collection of polygons smooth surfaces. The rest of the paper is organized in the following manner. In Section 2, we present a classification of the the collision detection problem. We briefly survey the algorithms available for polygonal models in Section 3. Then, we describe the state of art in collision detection for general geometric models and discuss the advantages and shortcomings of each method in Section 4. Section 5 highlights techniques in reducing the number of pairwise intersection tests for a computer simulated environment consisting of multiple static or moving objects. We list a number of public domain packages for collision detection in Section 6.

2 Problem domain classification

A wide range of techniques, including hierarchical representation, geometric reasoning, algebraic formulations, spatial partitioning, analytical methods, and optimization methods, have been proposed. Algorithm design depends on the model representation, the desired query types, and the simulation environment.

2.1 Model representations

There are many types of model representations used in CAD/CAM and 3D graphics. One possible taxonomy that we adopt (for the ease of organization) in this paper is shown in Figure 1.

2.1.1 Polygonal models

Polygonal objects are the most commonly used models in computer graphics and modeling. They have a simple representation. They are versatile. Hardware-accelerated rendering of polygon is widely available.

The most general class of polygonal model is the "*polygon soup*," which is a collection of polygons that are not geometrically connected and has no topology information available. If the polygons form a closed manifold, then the model has a well-defined inside and outside – it is a proper solid. Some geometric 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.

2.1.2 Constructive solid geometry

Constructive Solid Geometry or CSG forms objects from primitives such as blocks, spheres, cylinders, cones, and tori, by combining them with set theoretic operations such as union, intersection, and set difference [RR92, Hof89]. One strength of the CSG representation 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. It also makes finding a collision witness easier [Cam91]. 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, an accurate boundary or surface representation, useful for rendering or interference computations, can be hard to compute from CSG representations [Hof89, KKM97].

2.1.3 Implicit surfaces

Implicit surfaces are defined using implicit functions. They are defined with 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, $f(x, y, z) > 0$. Consequently, implicit surfaces are generically closed manifolds, a desirable property.

If the function is polynomial in x, y, and z, then it is called *algebraic*, which includes the algebraic surfaces [Sed90], higher-order functions [BW90] and convolution surfaces. Implicits are also often used as the primitives in CSG systems.

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. They are widely used in

a number of applications and a number of specialized algorithms have been developed for intersection computations between quadrics [FNO89, MG91, SJ91].

2.1.4 Parametric surfaces

Parametric surfaces are mappings from some subset of the plane to space, $f : R^2 \mapsto R^3$. Unlike implicit surfaces, parametric surfaces are not generally closed manifolds. Hence, unlike CSG and implicit surfaces, they do not represent a complete solid model, but rather a description of surface boundary.

Parametric surfaces are easier to polygonalize and render as compared to the implicits, and a special class called the Non-Uniform Rational B-Spline (NURBS) has gained in popularity in CAD [LR80, Far93]. NURBS have some very nice properties which make them easier to operate on. They can also be represented using Bézier patches. It is worth noting that rational parametric surfaces (like NURBS and Bézier patches) are a proper subset of algebraic surfaces.

2.2 Different types of queries

In the simplest case, we want to know whether two models touch. Sometimes, we must find which parts (if any) touch, i.e. find their intersection. Sometimes we want to know their separation: if two objects are disjoint, what is the minimum Euclidean distance between them? If they penetrate, what is the minimum translational distance required to separate them [CC86]? Finally, if we know the objects' placements and motions, when will be their next collision? This is ETA computation, borrowing from the phrase, "estimated time of arrival".

Different applications need different queries. Distance information is useful for computing interaction forces and penalty functions in robot motion planning [Lat91] and dynamic simulation [Lin93, MC95]. Intersection computation is important for physically-based modeling and animation systems which must know all contacts in order to compute the collision response. The ETA solution permits us to control the time step in a simulation [Lin93, LM95].

2.3 Simulation environments

Special characteristics of each simulation are often considered in designing and choosing the most appropriate algorithm for collision detection. Here we examine a few common cases.

2.3.1 Pair processing vs. nbody processing

If the problem involves only a pair of models, we call it "pair processing." If we have many different parts, we call it "Nbody processing," in reference to the classic problem in celestial mechanics (many bodies moving under mutual gravitational influence).

2.3.2 Motions: static vs. dynamic

Queries are often executed repeatedly on the same models in the same environment, as the objects rotate and translate (or possibly subject to non-rigid transformations [HLMD96]) at successive time steps. In these dynamic environments, the geometric relationship may only differ slightly from that of the previous step, if the motion between steps is relatively small. Algorithms that can capitalize on this property are said to be exploit *temporal coherence*.

In order to exploit temporal coherence, some algorithms require bounds [Lin93, LM95] on the motion of the objects (e.g. objects' velocities or accelerations). Other algorithms, such as the ones based on interval arithmetic, need a closed-form expression of the motion as a function of time. Some algorithms demand no information on the motion, but need only the placements of the objects at successive time steps.

Sometimes the problem involves objects which are not in motion. For example, given a model of an entire power-plant, design engineers may be interested in performing static interference checks among components of the entire plant for tolerance verification and access clearance.

2.3.3 Rigid bodies vs. 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 time steps are small, some algorithms may be able to exploit temporal coherence in this case as well.

3 Collision detection for polygonal models

Most of the earlier work in collision detection has focused on algorithms for convex polytopes. A number of algorithms with good asymptotic performance have been proposed in the computational geometry literature. Using hierarchical representations, an $O(log^2 n)$ algorithm is given in [DK90] for polytope-polytope overlap problem, where n is the number of vertices. This elegant approach has not been robustly implemented in

3D, however. Good theoretical and practical approaches based on linear complexity of the linear programming problem are known [Meg83, Sei90]. Minkowski difference and convex optimization techniques are used in [GJK88] to compute the distance between convex polytopes by finding the closest points.

In applications involving rigid motion, geometric coherence has been exploited to design algorithms for convex polyhedra based on local features [Bar90, LC91, Lin93]. Local properties have been used in the earlier motion planning algorithms by [Don84, LPW79] when two objects come into contact. These algorithms exploit the spatial and temporal coherence between successive queries and work well in practice.

A number of hierarchies have been used for collision detection between general polygonal models. Typical examples of bounding volumes include axis-aligned boxes (cubes are a special case) and spheres, and they are chosen for their fast overlap tests. Other structures include cone trees, k-d trees and octrees [Sam89], sphere trees [Hub93, Qui94], trees based on S-bounds [Cam91] etc. Binary space partitions (BSP) [NAT90] and extensions to multi-space partitions [BV91], and spatial partitionings based on space-time bounds or four-dimensional testing [AANJ94, Cam90, Can86, Hub93] have been used. All of these hierarchical methods do very well in performing "rejection tests" whenever two objects are far apart. However, when the two objects are in close proximity and can have multiple contacts, these algorithms either use subdivision techniques or check very large number of bounding volume pairs for potential contacts. In such cases, their performance slows down considerably.

More recent work seems to have focused on tighter-fitting bounding volumes. Gottschalk et al. [GLM96], have presented a fast algorithm and a system, called RAPID, for interference detection based on oriented bounding boxes, which approximate geometry better than do axis-aligned bounding boxes. Barequet et al. [BCG+96] have also used oriented bounding boxes for computing hierarchical representations of surfaces for performing collision detection. Klosowski et al. [KHM+96] have used discrete orientation polytopes (k-DOPs), which also are superior approximations to bounded geometry. Krishnas et. al. [KPLM98] have proposed a higher order bounding volume, designed to match curvature of the underlying 3D geometry, especially suited for Bézier patches and NURBS.

4 Algorithms for non-polygonal models

In geometric and solid modeling, the problem of computing the intersection of surfaces represented as splines or algebraic surfaces has received a great deal of attention [Pra86, Hof89]. Given two surfaces, the prob-

lem corresponds to computing all components of the intersection curve, robustly and accurately. It includes work on curves and surface intersections [SWZ89, SP86, BHHL88, Hof89, Hof90, MD94, MD95, KM97]. All these algorithms have focussed on accurate computation of the intersection set for static models. However, for collision detection we are actually dealing with a restricted version of this problem. That is, given two surfaces we want to know whether they intersect. Furthermore, we are interested in dynamic environments composed of moving objects.

In general, given two spline surfaces, there is *no* good and quick solution to the problem of whether they intersect or have a common geometric contact. The simplest solution is based on using subdivision and checking the control polytopes or convex bounding boxes for collision.

4.1 Constructive solid geometry models

Since CSG objects are defined using 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 collide . However, this requires efficient ways to compute whether an intersection yields the empty set or not. This is sometimes called the "Null Object Detector", or NOD [Hof89, Man88]. One possible solution involves computing a boundary representation of the resulting solid. However, efficient, accurate and robust computation of the boundary remains a hard problem for CSG models described using curved primitives [Hof89, KKM97].

4.1.1 S-bounds

Cameron [Cam91] introduced "S-Bounds" as a means of speeding up intersection evaluation. 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. In practice, Cameron observed that using S-Bounds increased the speed of intersection testing by one to two orders of magnitude on his CSG system [Cam91].

4.1.2 Interval arithmetic and CSG combinations of implicit functions

Duff [Duf92] 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 obtained from adaptive subdivision of space. This technique is intrinsicaly approximate. It may not be able to determine the contact status of disjoint models which are almost touching.

Duff applies this approach to a list of two or more CSG models, so it is an nbody algorithm. He also extends this method to distance computation and the ETA problem when closed-form solution of the motions of the objects are known.

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 an expensive method which will not perform real time collision detection on large models with current computing hardware.

4.2 Parametric surfaces

There are four techniques 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 in [Pra86, Hof89].

4.2.1 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 patch subsections. Depending on various criterion, the subsections are further subdivided and recursively examined, 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.

4.2.2 Subdivision of domain with interval arithmetic

Snyder [Sny92] uses interval arithmetic to adaptively subdivide the domains of the surfaces to refine an approximation of the intersection curve

until it is of satisfactory precision. Interval arithmetic is used to obtain conservative bounding boxes in space for the surface subsections.

The method does not determine with certainty that a contact has occured – it can only report a conservative upper and lower bounds on the patches' closest approach in space. Intersection is said to have occured when the lower bound on distance remains zero. The algorithm can handle distance queries by refining the bounds as the patch subdivisions are directed by a priority queue.

4.2.3 Subdivision of domains of time-varying parametric surfaces

A time-varying surface is a time-dependent mapping from a (u, v)-parameter patch to space: $f : (u, v) \times t \mapsto R^3$. The dependence on time can reflect motion as well as deformations with time.

Herzen, Barr, and Zatz [HBZ90] adaptively subdivide the domain (u, v, and t) of the mappings into subregions and use use Lipschitz Conditions (bounds on various derivatives of the mappings) to obtain bounds on the scope of the subregion's range in space. A priority queue is used to direct the subdivision so as to locate the earliest pair of subregions (ie. containing the smallest t) which overlap in space.

Hence, they have an ETA algorithm for deformable parametric surfaces, which is robust and accurate to any desired precision. It's disadvantages are that it is time-consuming, fundamentally approximate, and requires the motion of the models to be expressed as closed-form functions of time (such motion functions are not always available or convenient).

4.2.4 Improved interval arithmetic methods for deformable surfaces

In 1993, Snyder et al. [Sny93] 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.

4.2.5 Lattice methods

The intersection curve of two surfaces 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 [Pra86]. Selecting a value for u on one of the patches reduces the dimensionality of the search space to one – the 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 missed by the isoparametric curves – this most often occurs when the surfaces are grazing or barely penetating. In some cases, lattice methods or subdivision methods are used to find starting points for use by the tracing methods.

4.2.6 Tracing methods

The tracing method begins with a given point known to be on the intersection curve [BFJP87, BHHL88, Hoh91, MC91, KM97]. 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 [KPW90] or solving a system of algebraic equations [BHHL88, MC91, KM97]. 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.

4.2.7 Analytic methods

Analytic methods usually involve implicitizing one of the parametric surfaces – obtaining an implicit representation of the model [SAG84]. 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 [KPP90, MC91, KM97, Sar83]. Based on its representation as an algebraic plane curve, efficient algorithms have been proposed by a number of researchers [KM97].

4.3 Implicit surface

Pentland and Williams, [PW89], proposed using implicit functions to represent shape and the property of the "inside-outside" functions for collision detection. Besides its restriction to implicits, this algorithm has a drawback in terms of robustness as it only uses point samples. Lin and Manocha have presented efficient algorithms for curved models composed of either spline surfaces and algebraic surfaces undergoing rigid motion, using extension of their earlier algorithm for polyhedra, hierarchical representation and equation solving techniques [LM93, LM95, LM97]. These algorithms work well on mostly low degree primitives.

5 Nbody processing

For environments consisting of n (possibly moving) objects, performing $O(n^2)$ pairwise interference checks becomes a computational bottleneck, when n is large. In order to eliminate some unnecessary pairwise checks and to speed up the runtime performance, several techniques have been proposed. Algorithms of complexity $O(nlog^2n + m)$ have been presented for spheres in [HSS83] and rectangular bounding boxes in [Ede83], where m corresponds to the actual number of overlaps. Some of the fastest practical algorithms assume the knowledge of maximum acceleration and velocity; others exploit the spatial arrangement to reduce the number of pairwise interference tests without assuming any knowledge of trajectories.

5.1 Scheduling scheme

With bounds on velocities and accelerations we can estimate lower bounds on potential collision times. Scheduling algorithms [Lin93, LM95] maintain a queue of all object pairs that might collide, which is sorted by lower bounds on time to collision. These lower bounds on the time to collision are calculated adaptively and updated when a critical event, such as a collision, occurs. This technique has been successfully incorporated in the Impulse-Based Dynamics Simulator [MC95], reducing the frequency of the collision checks and thereby speeding up dynamics simulation.

5.2 Sorting-based sweep and prune

More recently, Cohen et al. have presented algorithms and a system, I-COLLIDE, based on spatial and temporal coherence, for large environments composed of multiple moving objects [CLMP95]. The number of object pair interactions is reduced to only the pairs within close proximity by sorting axis-aligned bounding boxes (AABBs) surrounding the objects. It is output sensitive and its run time is linearly dependent on the number of objects in the environment instead of quadratic dependence. It uses dynamically sized AABBs, linear *sweep and prune*, and geometric coherence to quickly reject the object pairs, that are unlikely to collide within the next time step.

5.3 Interval tree for 2D intersection tests

We can use the interval tree [Ede83] for static query, as well as for the rectangle intersection problem. Each query of interval intersection takes $O(logn + k)$ time where k is the number of reported intersection and n is the number of intervals. Therefore, reporting intersection among n rectangles can be done in $O(nlogn + K)$ where K is the total number of intersecting rectangles.

5.4 Uniform spatial subdivision

We can divide the space into unit cells (or volumes) and place each object (or bounding box) in some cell(s) [Ove92]. To check for collisions, we have to examine the cell(s) occupied by each box to verify if the cell(s) is(are) shared by other objects. But, it is difficult to set a near-optimal size for each cell and it requires tremendous amount of allocated memory. If the size of the cell is not properly chosen, the computation can be expensive. For an environment where objects are of uniform size [Tur89], this is a rather ideal algorithm and especially suitable for parallelization. Overmars has shown that using a hash table to look up an entry and O(n) storage space we can perform the point location queries in constant time [Ove92].

6 Public Domain Software Packages

Most of public domain systems are applicable to polygonal models and some are also applicable to large environments composed of multiple moving objects. It is nearly impossible to compare different algorithms and systems *fairly*, since their performance varies, depending on the simulation

environments (models, varieties of contacts, query types, motion description, etc.) and other factors. Here we only list them in the chronological order of their release and briefly describe their special characteristics.

6.1 I-COLLIDE collision detection system

http://www.cs.unc.edu/~geom/I_COLLIDE.html.

I-COLLIDE is an interactive and exact collision-detection library for environments composed of *many* convex polyhedra or union of convex pieces, based on the *expected constant time*, incremental distance computation algorithm [LC91, Lin93] and algorithms to check for collision between multiple moving objects [CLMP95].

6.2 RAPID interference detection system

http://www.cs.unc.edu/~geom/OBB/OBBT.html.

RAPID is a robust and accurate polygon interference detection library for pairs of unstructured polygonal models. It is applicable to polygon soups – models which contain no adjacency information and obey no topological constraints. It is most suitable for *close proximity configurations* between highly tesselated smooth surfaces. [GLM96].

6.3 V-COLLIDE collision detection system

http://www.cs.unc.edu/~geom/V_COLLIDE.

V-COLLIDE is a collision detection library for large dynamic environments [HLC+97], and unites the nbody processing algorithm of I-COLLIDE and the pair processing algorithm of RAPID. It is designed to operate on large numbers of static or moving polygonal objects to allow dynamic addition or deletion of objects between timesteps.

6.4 Distance computation between convex polytopes

http://www.comlab.ox.ac.uk/oucl/users/stephen.cameron/distances.html

This package is an enhanced and dynamic version [Cam97b, Cam97a] of the distance routine of Gilbert, Johnson and Keerthi (GJK), which allows the tracking of the distance between a pair of convex polyhedra. It requires a list of all the edges in each convex polyhedra for best performance. Its performance is comparable to Lin-Canny convex polytope overlap test.

6.5 SOLID interference detection system

http://www.win.tue.nl/cs/tt/gino/solid/

SOLID is a library for interference detection of multiple three-dimensional polygonal objects (including polygon soups) undergoing rigid motion. Its performance and applicability is comparable to that of V-COLLIDE.

6.6 V-Clip collision detection system

http://www.merl.com/people/mirtich/vclip.html

The Voronoi Clip, or V-Clip, algorithm is a low-level collision detection algorithm for polyhedral objects – an improvement of the closest-feature tracking algorithm [LC91, Lin93]. It operates on a pair of convex polyhedra, or nonconvex hierarchies of them. In addition to distance computation, it can also report penetration points and estimated penetration distance between overlapping models.

7 Future work

Despite abundant wealth of the literature in collision detection, there are several open research issues. Much remains to be done on detecting contacts between deformable models *accurately and efficiently*. In dynamic simulation, computing collision response requires *robust* and *interactive* computation of the closest features or contact points between general geometric models, as well as rapid calculation of penetration distance. This problem is especially difficult for those models with smooth surfaces and many concavities. There are also new challenges in applying collision detection algorithms to *massive models*, which consist of millions of primitives and are often too large to fit in the main memory. These may include developing external memory algorithms, dynamic pre-fetching techniques and parallel computing methods for collision detection.

Acknowledgment: We would like to acknowledge partial support provided by Intel and Honda and the reviewers for their suggestions.

References

[AANJ94] A.Garica-Alonso, N.Serrano, and J.Flaquer. Solving the collision detection problem. *IEEE Computer Graphics and Applications*, 13(3):36–43, 1994.

[AB88] S.S. Abhyankar and C. Bajaj. Computations with algebraic curves. In *Lecture Notes in Computer Science*, volume 358, pages 279–284. Springer Verlag, 1988.

[Bar90] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *ACM Computer Graphics*, 24(4):19–28, 1990.

[BCG+96] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. Boxtree: A hierarchical representation of surfaces in 3d. In *Proc. of Eurographics'96*, 1996.

[BFJP87] R. Barnhill, G. Farin, M. Jordan, and B. Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4(3):3–16, 1987.

[BHHL88] C.L. Bajaj, C.M. Hoffmann, J.E.H. Hopcroft, and R.E. Lynch. Tracing surface intersections. *Computer Aided Geometric Design*, 5:285–307, 1988.

[BV91] W. Bouma and G. Vanecek. Collision detection and analysis in a physically based simulation. *Proceedings Eurographics workshop on animation and simulation*, pages 191–203, 1991.

[BW90] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, pages 109–116, March 1990.

[Cam84] Stephen Cameron. *Modelling Solids in Motion*. PhD thesis, University of Edinburgh, 1984.

[Cam85] S. Cameron. A study of the clash detection problem in robotics. *Proceedings of International Conference on Robotics and Automation*, pages 488–493, 1985.

[Cam90] S. Cameron. Collision detection by four-dimensional intersection testing. *Proceedings of International Conference on Robotics and Automation*, pages 291–302, 1990.

[Cam91] S. Cameron. Approximation hierarchies and s-bounds. In *Proceedings. Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 129–137, Austin, TX, 1991.

[Cam97a] S. Cameron. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[Cam97b] Stephen Cameron. A comparison of two fast algorithms for computing the distance between convex polyhedra. *IEEE Transactions on Robotics and Automation*, 13(6):915–920, December 1997.

[Can86] J. F. Canny. Collision detection for moving polyhedra. *IEEE Trans. PAMI*, 8:200–209, 1986.

[CC86] S. Cameron and R. K. Culley. Determining the minimum translational distance between two convex polyhedra. *Proceedings of International Conference on Robotics and Automation*, pages 591–596, 1986.

[CL90] J. F. Canny and M. C. Lin. An opportunistic global path planner. *Proceedings of International Conference on Robotics and Automation*, pages 1554–1559, 1990.

[CLMP95] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Interactive 3D Graphics Conference*, pages 189–196, 1995.

[DK90] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra – A unified approach. In *Proc. 17th Internat. Colloq. Automata Lang. Program.*, volume 443 of *Lecture Notes Comput. Sci.*, pages 400–413. Springer-Verlag, 1990.

[Don84] B. R. Donald. Motion planning with six degrees of freedom. Master's thesis, MIT Artificial Intelligence Lab., 1984. AI-TR-791.

[Duf92] Tom Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *ACM Computer Graphics*, 26(2):131–139, 1992.

[Ede83] H. Edelsbrunner. A new approach to rectangle intersections, Part I. *Internat. J. Comput. Math.*, 13:209–219, 1983.

[Far93] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press Inc., 1993.

[FNO89] R.T. Farouki, C.A. Neff, and M. O'Connor. Automatic parsing of degenerate quadric-surface intersections. *ACM Transactions on Graphics*, 8:174–203, 1989.

[GJK88] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between objects in three-dimensional space. *IEEE J. Robotics and Automation*, vol RA-4:193–203, 1988.

[GLM96] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM Siggraph'96*, pages 171–180, 1996.

[HBZ90] B. V. Herzen, A. H. Barr, and H. R. Zatz. Geometric collisions for time-dependent parametric surfaces. *Computer Graphics*, 24(4):39–48, 1990.

[HLC+97] T. Hudson, M. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-collide: Accelerated collision detection for vrml. In *Proc. of VRML Conference*, pages 119–125, 1997.

[HLMD96] M. Hughes, M. Lin, D. Manocha, and C. Dimattia. Efficient and accurate interference detection for polynomial deformation and soft object animation. In *Proceedings of Computer Animation*, pages 155–166, Geneva, Switzerland, 1996.

[HMPY97] C. Hu, T. Maekwa, N. Patrikalakis, and X. Ye. Robust interval algorithm for surfaces intersections. *Computer-Aided Design*, 29(9):617–627, 1997.

[Hof89] C.M. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, San Mateo, California, 1989.

[Hof90] C.M. Hoffmann. A dimensionality paradigm for surface interrogations. *Computer Aided Geometric Design*, 7:517–532, 1990.

[Hoh91] M.E. Hohmeyer. A surface intersection algorithm based on loop detection. *International Journal of Computational Geometry and Applications*, 1(4):473–490, 1991. Special issue on Solid Modeling.

[HSS83] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. Efficient detection of intersections among spheres. *The International Journal of Robotics Research*, 2(4):77–80, 1983.

[Hub93] P. M. Hubbard. Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, October 1993.

[KHM+96] J. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. In *Siggraph'96 Visual Proceedings*, page 151, 1996.

[KKM97] J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate b-rep generation of low degree sculptured solids using exact arithmetic. In *ACM/SIGGRAPH Symposium on Solid Modeling*, pages 42–55, 1997.

[KM97] S. Krishnan and D. Manocha. An efficient surface intersection algorithm based on the lower dimensional formulation. *ACM Transactions on Graphics*, 16(1):74–106, 1997.

[KPLM98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shell: A higher order bounding volume for fast proximity queries. In *Proc. of Third International Workshop on Algorithmic Foundations of Robotics*, 1998.

[KPP90] G.A. Kriezis, P.V. Prakash, and N.M. Patrikalakis. Method for intersecting algebraic surfaces with rational polynomial patches. *Computer-Aided Design*, 22(10):645–654, 1990.

[KPW90] G.A. Kriezis, N.M. Patrikalakis, and F.E. Wolter. Topological and differential equation methods for surface intersections. *Computer-Aided Design*, 24(1):41–55, 1990.

[Lat91] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[LC91] M.C. Lin and John F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, pages 1008–1014, 1991.

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

[LM93] M.C. Lin and Dinesh Manocha. Interference detection between curved objects for computer animation. In *Models and Techniques in Computer Animation*, pages 43–57. Springer-Verlag, 1993.

[LM95] M.C. Lin and Dinesh Manocha. Fast interference detection between geometric models. *The Visual Computer*, 11(10):542–561, 1995.

[LM97] M.C. Lin and Dinesh Manocha. Efficient contact determination between geometric models. *International Journal of Computational Geometry and Applications*, 7(1):123–151, 1997.

[LPW79] T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979.

[LR80] J.M. Lane and R.F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):150–159, 1980.

[Man88] M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Maryland, 1988.

[MC91] D. Manocha and J.F. Canny. A new approach for surface intersection. *International Journal of Computational Geometry and Applications*, 1(4):491–516, 1991. Special issue on Solid Modeling.

[MC95] B. Mirtich and J. Canny. Impulse-based simulation of rigid bodies. In *Proc. of ACM Interactive 3D Graphics*, Monterey, CA, 1995.

[MD94] D. Manocha and J. Demmel. Algorithms for intersecting parametric and algebraic curves I: simple intersections. *ACM Transactions on Graphics*, 13(1):73–100, 1994.

[MD95] D. Manocha and J. Demmel. Algorithms for intersecting parametric and algebraic curves ii: multiple intersections. *Computer Vision, Graphics and Image Processing: Graphical Models and Image Processing*, pages 81–100, 1995.

[Meg83] N. Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM J. Computing*, 12:pp. 759–776, 1983.

[MG91] J. Miller and R. Goldman. Combining algebraic rigor with geometric robustness for the detection and calculation of conic sections in the intersection of two quadric surfaces. *Proceedings of ACM Solid Modeling*, pages 221–233, 1991.

[Moo79] R.E. Moore. *Methods and applications of interval analysis.* SIAM studies in applied mathematics. Siam, 1979.

[NAT90] B. Naylor, J. Amanatides, and W. Thibault, "Merging bsp trees yield polyhedral modeling results", in *Proc. of ACM Siggraph*, 1990, pp. 115–124.

[Ove92] M. H. Overmars. Point location in fat subdivisions. *Inform. Proc. Lett.*, 44:261–265, 1992.

[Pat93] N.M. Patrikalakis. Surface-to-surface intersections. *IEEE Computer Graphics and Applications*, 13(1):89–95, 1993.

[Pra86] M.J. Pratt. Surface/surface intersection problems. In J.A. Gregory, editor, *The Mathematics of Surfaces II*, pages 117–142, Oxford, 1986. Claredon Press.

[PW89] Alex Pentland and John Williams. Good vibrations: Modal dynamics for graphics and animation. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 215–222, July 1989.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[RR92] A.A.G. Requicha and J.R. Rossignac. Solid modeling and beyond. *IEEE Computer Graphics and Applications*, pages 31–44, September 1992.

[SAG84] T.W. Sederberg, D.C. Anderson, and R.N. Goldman. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics and Image Processing*, 28:72–84, 1984.

[Sam89] H. Samet. *Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods.* Addison Wesley, 1989.

[Sar83] R F Sarraga. Algebraic methods for intersection. *Computer Vision, Graphics and Image Processing*, 22:222–238, 1983.

[Sny93] J. Snyder and et. al. Interval methods for multi-point collisions between time dependent curved surfaces. In *Proceedings of ACM Siggraph*, pages 321–334, 1993.

[Sed90] T.W. Sederberg. Techniques for cubic algebraic surfaces. *IEEE Computer Graphics and Applications*, pages 14–25, July 1990.

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, Berkeley, California, 1990.

[SJ91] C. Shene and J. Johnstone. On the planar intersection of natural quadrics. *Proceedings of ACM Solid Modeling*, pages 234–244, 1991.

[Sny92] J. Snyder. Interval arithmetic for computer graphics. In *Proceedings of ACM Siggraph*, pages 121–130, 1992.

[SP86] T.W. Sederberg and S.R. Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, 1986.

[SWZ89] T.W. Sederberg, S. White, and A. Zundel. Fat arcs: A bounding region with cubic convergence. *Computer Aided Geometric Design*, 6:205–218, 1989.

[Tur89] G. Turk. Interactive collision detection for molecular graphics. Master's thesis, Computer Science Department, University of North Carolina at Chapel Hill, 1989.

[ZSP93] J. Zhou, E.C. Sherbrooke, and N.M. Patrikalakis. Computation of stationary points of distance functions. *Engineering with Computers*, 9(4):231–246, 1993.