35 COLLISION AND PROXIMITY QUERIES

Ming C. Lin and Dinesh Manocha

INTRODUCTION

In a geometric context, a collision or proximity query reports information about the relative configuration or placement of two objects. Some of the common examples of such queries include checking whether two objects overlap in space, or whether their boundaries intersect, or computing the minimum Euclidean separation distance between their boundaries. Hundreds of papers have been published on different aspects of these queries in computational geometry and related areas such as robotics, computer graphics, virtual environments, and computer-aided design. These queries arise in different applications including robot motion planning, dynamic simulation, haptic rendering, virtual prototyping, interactive walkthroughs, computer gaming, and molecular modeling. For example, a large-scale virtual environment, e.g., a walkthrough, creates a model of the environment with virtual objects. Such an environment is used to give the user a sense of presence in a synthetic world and it should make the images of both the user and the surrounding objects feel solid. The objects should not pass through each other, and objects should move as expected when pushed, pulled, or grasped; see Fig. 35.0.1. Such actions require fast and accurate collision detection between the geometric representations of both real and virtual objects. Another example is rapid prototyping, where digital representations of mechanical parts, tools, and machines, need to be tested for interconnectivity, functionality, and reliability. In Fig. 35.0.2, the motion of the pistons within the combustion chamber wall is simulated to check for tolerances and verify the design.

This chapter provides an overview of different queries and the underlying algorithms. It includes algorithms for collision detection and distance queries among convex polytopes (Section 35.1), nonconvex polygonal models (Section 35.2), penetration depth queries (Section 35.3), curved objects (Section 35.4), dynamic queries (Section 35.5), and large environments composed of multiple objects (Section 35.6). Finally, it briefly describes different software packages available to perform some of the queries (Section 35.7).

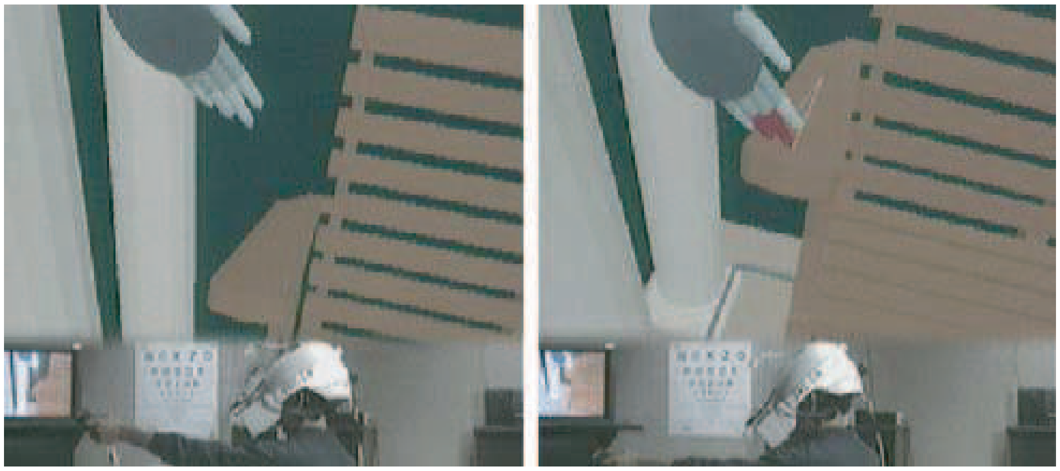
PROBLEM CLASSIFICATION

Collision Detection: Checks whether two objects overlap in space or their boundaries share at least one common point.

Separation Distance: Length of the shortest line segment joining two sets of points, A and B:

$$\mathrm{dist}(A,B) = \min_{a\in A}\min_{b\in B}|a-b|.$$

FIGURE 35.0.1
A hand reaching toward a chair on a virtual porch, at top. The corresponding image of the user in the real world is shown on the bottom. Darkened finger tips indicate contacts between the user's hand and the virtual chair.



Hausdorff Distance: Maximum deviation of one set from the other:

$$\text{haus}(A, B) = \max_{a \in A} \min_{b \in B} |a - b|.$$

Spanning Distance: Maximum distance between the points of two sets:

$$\text{span}(A, B) = \max_{a \in A} \max_{b \in B} |a - b|.$$

Penetration Depth: Minimum distance needed to translate one set to make it disjoint from the other:

$$\text{pen}(A, B) = \text{minimum } ||\mathbf{v}|| \text{ such that} \min_{a \in A} \min_{b \in B} |\mathbf{a} - \mathbf{b} + \mathbf{v}| > 0.$$

There are two forms of collision detection query: ***Boolean*** and ***enumerative***. The Boolean distance query computes whether the two sets have at least one point in common. The enumerative form yields some representation of the intersection set.

There are at least three forms of the distance queries: exact, approximate, and Boolean. The exact form asks for the exact distance between the objects. The approximate form yields an answer within some given error tolerance of the true measure—the tolerance could be specified as a relative or absolute error. The Boolean form reports whether the exact measure is greater or less than a given value. Furthermore, the norm by which distance is defined may be varied. The Euclidean norm is the most common, but in principle other norms are possible, such as the L_1 and L_∞ norms.

Each of these queries can be augmented by adding the element of time. If the trajectories of two objects are known, then the next time can be determined at which a particular Boolean query (collision, separation distance, or penetration) will become TRUE or FALSE. In fact, this "time-to-next-event" query can have exact,

FIGURE 35.0.2
In this virtual prototyping application, the motion of the pistons is simulated to check for tolerances by performing distance queries.

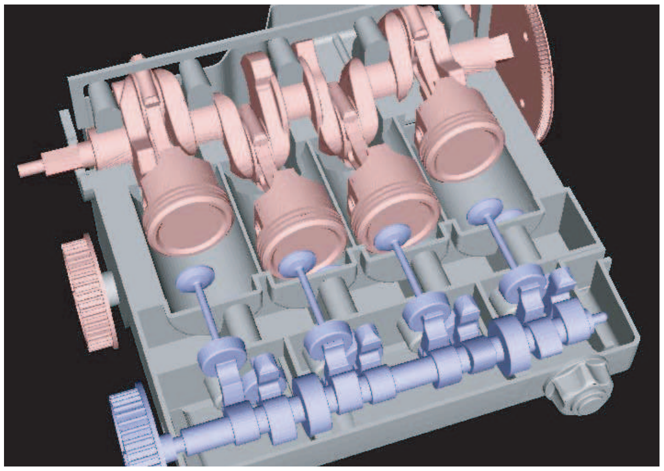


TABLE 35.0.1 Classification of Proximity Queries

CRITERIA	TYPES
Report	Boolean, exact, approximate, enumerative
Measure	Separation, span, Hausdorff, penetration, collision
Multiplicity	2-body, n-body
Temporality	Static, dynamic
Representation	Polyhedra, convex objects, implicit, parametric, NURBS, quadrics, set-theoretic combinations
Dimension	2,3,d

approximate, and Boolean forms. These queries are called ***dynamic queries***, whereas the ones that do not use motion information are called ***static queries***. In the case where the motion of an object can not be represented as a closed-form function of time, the underlying application often performs static queries at specific time steps in the application.

These measures, as defined above, apply only to pairs of sets. However, some applications work with many objects, and need to find the proximity information among all or a subset of the pairs. Thus, most of the query types listed above have associated N-body variants.

Finally, the primitives can be represented in different forms. They may be convex polytopes, general polygonal models, curved models represented using parametric or implicit surfaces, set-theoretic combination of objects, etc. Different set of algorithms are known for each representation. A classification of proximity queries based on these criteria is shown in Table 35.1.1.

35.1 CONVEX POLYTOPES

In this section, we give a brief survey of algorithms for collision detection and separation-distance computation between a pair of convex polytopes. A number of algorithms with good asymptotic performance have been proposed. The algorithm with the current best runtime for Boolean collision queries takes $O(\log^2 n)$ time, where n is the number of features [DK90]. It precomputes the Dobkin-Kirkpatrick (DK) hierarchy for each polytope and uses it to perform the query. In practice, three classes of algorithms are commonly used for convex polytopes: linear programming, Minkowski sums, and tracking closest features based on Voronoi diagrams.

LINEAR PROGRAMMING

The problem of checking whether two convex polytopes intersect or not can be posed as a linear programming (LP) problem. In particular, two convex polytopes do not overlap if and only if there exists a separation plane between them. The coefficients of the separation plane equation are treated as unknowns. Linear constraints result by requiring that all vertices of the first polytope lie in one halfspace of this plane and those of the other polytope lie in the other halfspace. The linear programming algorithms are used to check whether there is any feasible solution to the given set of constraints. Given the fixed dimension of the problem, some of the well-known linear programming algorithms (e.g., [Sei90]; cf. Chapter 45) can be used to perform the Boolean collision query in expected linear-time. By caching the last pair of witness points to compute the new separating planes, Chung and Wang [CW96] proposed an iterative method that can quickly update the separating axis or the separating vector in nearly "constant time" in dynamic applications with high motion coherence.

MINKOWSKI SUMS AND CONVEX OPTIMIZATION

Collision and distance queries can be performed based on the Minkowski sum of two objects. It has been shown [CC86] that the minimum separation distance between two objects is the same as the minimum distance from the origin of the Minkowski sums of A and $-B$ to the surface of the sums. The Minkowski sum is also referred to as the ***translational C-space obstacle*** (TCSO). While the Minkowski sum of two convex polytopes can have $O(n^2)$ features [DHKS93], a fast algorithm for separation-distance computation based on convex optimization that exhibits linear-time performance in practice has been proposed by Gilbert et al. [GJK88], also known as the GJK algorithm. It uses pairs of vertices from each object that define simplices within each polytope and a corresponding simplex in the TCSO. Initially the simplex is set randomly and the algorithm refines it using local optimization, until it computes the closest point on the TCSO from the origin of the Minkowski sums. The algorithm assumes that the origin is not inside the TCSO.

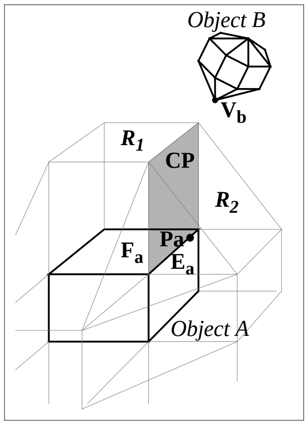


FIGURE 35.1.1
A walk across external Voronoi region of Object A. Vertex $\mathbf{V}_b$ *of Object B lies in the Voronoi region of* $\mathbf{E}_a$*.*

TRACKING CLOSEST FEATURES USING GEOMETRIC LOCALITY AND MOTION COHERENCE

Lin and Canny [LC91] proposed a distance-computation algorithm between non-overlapping convex polytopes. Often referred to as the LC algorithm, it tracks the closest features between the polytopes. This is the first approach that explicitly takes advantages of motion coherence and geometric locality. The features may correspond to a vertex, face, or an edge on each polytope. It precomputes the external Voronoi region for each polytope. At each time step, it starts with a pair of features and checks whether they are the closest features, based on whether they lie in each other's Voronoi region. If not, it performs a local walk on the boundary of each polytope until it finds the closest features. See Figure 35.1.1. In applications with high motion coherence, the local walk typically takes nearly "constant time" in practice. Typically the number of neighbors for each feature of a polytope is constant and the extent of "local walk" is proportional to the amount of the relative motion undergone by the polytopes.

Mirtich [Mir98] further optimized this algorithm by proposing a more robust variation that avoids some geometric degeneracies during the local walk, without sacrificing the accuracy or correctness of the original algorithm.

Guibas et al. [GHZ99] proposed an approach that exploits both coherence of motion using LC and hierarchical representations by Dobkin and Kirkpatrick [DK90] to reduce the runtime dependency on the amount of the local walks.

Ehmann and Lin [EL00] modified the LC algorithm and used an error-bounded level-of-detail (LOD) hierarchy to perform different types of proximity queries, using the progressive refinement framework (cf. Chapter 54). The implementation of this technique, "multi-level Voronoi Marching," outperforms the existing libraries for collision detection between convex polytopes. It also uses an initialization technique based on directional lookup using hashing, resembling that of [DZ93].

By taking the similar philosophy as LC, Cameron [Cam97] presented an extension to the basic GJK algorithm by exploiting motion coherence and geometric locality in terms of connectivity between neighboring features. The algorithm tracks the ***witness points***, a pair of points from the two objects that realize the minimum separation distance between them. Rather than starting from a random simplex in the TCSO, the algorithm starts with the witness points from the previous iteration and performs hill climbing to compute a new set of witness points for the current configuration. The running time of this algorithm is a function of the number of refinement steps that the algorithm performs.

TABLE 35.1.1 Algorithms for convex polytopes.

METHOD	FEATURES
DK	$O(\log^2 n)$ query time, collision query only
LP	Linear running time, collision query
GJK	Linear-time behavior in practice, collision and separation-distance queries
LC	Expected constant-time in coherent environments, collision and separation-distance queries

KINETIC DATA STRUCTURES

Recently a new class of algorithms using "kinetic data structures" (or KDS for short) have been proposed for collision detection between moving convex polygons and polyhedra [BEG$^+$99, EGSZ99, KSS02] (cf. Chapter 50). These algorithms are based on the formal framework of KDS to keep track of closest features of polytopes during their motion and exploits motion coherence and geometric locality. The performance of KDS-based algorithms is separation sensitive, and may depend on the amount of the minimum distance between the objects during their motion, relative to their size. The type of motion includes straight-line linear motion, translation along an algebraic trajectory, or algebraic rigid motion (including both rotation and translation).

35.2 GENERAL POLYGONAL MODELS

Algorithms for collision and separation-distance queries between general polygons models can be classified based whether they assume closed polyhedral models, or are represented as a collection of polygons. The latter, also referred to as "polygon soups," make no assumption related to the connectivity among different faces or whether they represent a closed set.

Some of the most common algorithms for collision detection and separation-distance computation use spatial partitioning or bounding volume hierarchies (BVHs). The spatial subdivisions are a recursive partitioning of the embedding space, whereas bounding volume hierarchies are based on a recursive partitioning of the primitives of an object. These algorithms are based on the divide-and-conquer paradigm. Examples of spatial partitioning hierarchies include k-D trees and octrees [Sam89], R-trees and their variants [HKM95], cone trees, BSPs [NAT90] and their extensions to multi-space partitions [WG91]. The BVHs use bounding volumes (BVs) to bound or contain sets of geometric primitives, such as triangles, polygons, curved surfaces, etc. In a BVH, BVs are stored at the internal nodes of a tree structure. The root BV contains all the primitives of a model, and children BVs each contain separate partitions of the primitives enclosed by the parent. Leaf node BVs typically contain one primitive. In some variations, one may place several primitives at a leaf node, or use several volumes to contain a single primitive. BVHs are used to perform collision and separation-distance queries. These include sphere-trees [Hub95, Qui94], AABB-trees [BKSS90, HKM95, PML97], OBB-trees [GLM96, BCG$^+$96, Got00], spherical shell-trees [KPLM98, KGL$^+$98], k-DOP-trees [HKM96, KHM$^+$98], SSV-trees[LGLM99], multiresolution hierarchies [OL03], and convex hull-trees [EL01], as shown in Table 35.2.1.

TABLE 35.2.1 Types of bounding volume hierarchies.

NAME	TYPE OF BOUNDING VOLUME
Sphere-Tree	Sphere
AABB-Tree	Axis-aligned bounding box (AABB)
OBB-Tree	Oriented bounding box (OBB)
Spherical Shell-Tree	Spherical shell
k-DOP-Tree	Discretely oriented polytope defined by k vectors (k-DOP)
SSV-Tree	Swept-sphere volume (SSV)
Convex hull-Tree	Convex polytope

COLLISION DETECTION

Collision queries are performed by traversing the BVHs. Two models are compared by recursively traversing their BVHs in tandem. Each recursive step tests whether BVs A and B, one from each hierarchy, overlap. If they do not, the recursion branch is terminated. But if A and B overlap, the enclosed primitives may overlap and the algorithm is applied recursively to their children. If A and B are both leaf nodes, the primitives within them are compared directly.

SEPARATION-DISTANCE COMPUTATION

The structure of the separation-distance query is very similar to the collision query. As the query proceeds, the smallest distance found from comparing primitives is maintained in a variable δ. At the start of the query, δ is initialized to ∞, or to the distance between an arbitrary pair of primitives. Each recursive call with BVs A and B must determine if some primitive within A and some primitive within B are closer than, and therefore will modify, δ. The call returns trivially if BVs A and B are farther than the current δ, as this precludes any primitives within them being closer than δ. Otherwise the algorithm is applied recursively to its children. For leaf nodes it computes the exact distance between the primitives, and if the new computed distance is less than δ, it updates δ.

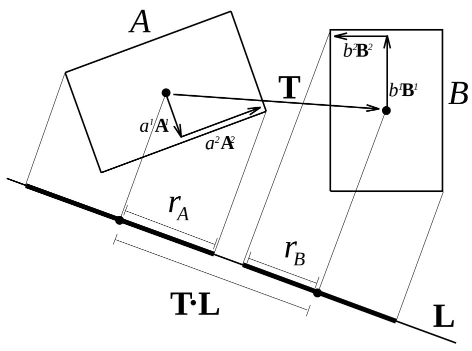


FIGURE 35.2.1
L *is a separating axis for OBBs A and B because projection onto* **L** *renders them disjoint intervals.*

To perform an approximate distance query, the distance between BVs A and B is used as a lower limit to the exact distances between their primitives. If this bound prevents δ from being reduced by more than the acceptable tolerance, that recursion branch is terminated.

QUERIES ON BOUNDING VOLUMES

Algorithms for collision detection and distance computation need to perform the underlying queries on the BVHs, including whether two BVs overlap, or computing the separation distance between them. The performance of the overall proximity query algorithm is governed largely by the performance of the subalgorithms used for proximity queries on a pair of BVs.

A number of specialized and highly optimized algorithms have been proposed to perform these queries on different BVs. It is relatively simple to check whether two spheres overlap. Two AABBs can be checked for overlap by comparing their dimensions along the three axes. The separation distance between them can be computed based on the separation along each axis. The overlap test can be easily extended to k-DOPs, where their projections are checked along the k fixed axis [KHM$^+$98].

An efficient algorithm to test two OBBs for overlap based on the separating axis theorem (SAT) has been presented in [GLM96, Got00]. It computes the projection of each OBB along 15 axes in 3D. The 15 axes are computed from the face normals of the OBBs (6 face normals) and by taking the cross-products of the edges of the OBBs (9 cross-products). It is shown that two OBBs overlap if and only if their projection along each of these axes overlap. Furthermore, an efficient algorithm that performs overlap tests along each axis has been described. In practice, it can take anywhere from 80 to 240 arithmetic operations to check whether two OBBs overlap. The computation is robust and works well in practice [GLM96]. Figure 35.2.1 shows one of the separating axis tests for two rectangles in 2D.

Algorithms based on different ***swept-sphere volumes*** (SSVs) have been presented in [LGLM99]. Three types of SSVs are suggested: point swept-sphere (PSS), line swept-sphere (LSS), and a rectangular swept-sphere (RSS). Each BV is formulated by taking the Minkowski sum of the underlying primitive—a point, line, or a rectangle in 3D, respectively—with a sphere. Algorithms to perform collision or distance queries between these BVs can be formulated as computing the distance between the underlying primitives. Larsen et al. [LGLM99] have presented an efficient and robust algorithm to compute distance between two rectangles in 3D (as well rectangles degenerating to lines and points). Moreover, they used priority directed search and primitive caching to lower the number of bounding volume tests for separation-distance computations.

In terms of higher-order bounding volumes, fast overlap tests based on spherical shells have been presented in [KPLM98, KGL$^+$98]. Each spherical shell corresponds to a portion of the volume between two concentric spheres. The overlap test between two spherical shells takes into account their structure and reduces to checking whether there is a point contained in a circle that lies in the positive halfplane defined by two lines. The two lines and the circles belong to the same plane.

PERFORMANCE OF BOUNDING VOLUME HIERARCHIES

The performance of BVHs on proximity queries is governed by a number of design parameters, including techniques to build the trees, the maximum number of children per node, and the choice of BV type. An additional design choice is the descent rule. This is the policy for generating recursive calls when a comparison of two BVs does not prune the recursion branch. For instance, if BVs A and B failed to prune, one may recursively compare A with each of the children of B, B with each of the children of A, or each of the children of A with each of the children of B. This choice does not affect the correctness of the algorithm, but may impact the performance. Some of the commonly used algorithms assume that the BVHs are binary trees and each primitive is a single triangle or a polygon. The cost of performing the proximity query is given as [GLM96, LGLM99]:

$$T = N_{bv} \times C_{bv} + N_p \times C_p,$$

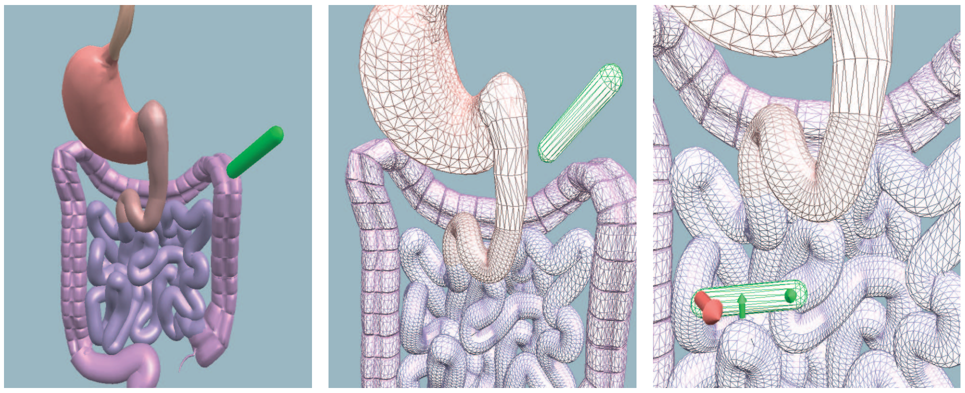
where T is the total cost function for proximity queries, N_{bv} is the number of bounding volume pair operations, and C_{bv} is the total cost of a BV pair operation, including the cost of transforming each BV for use in a given configuration of the models, and other per BV-operation overhead. N_p is the number of primitive pairs tested for proximity, and C_p is the cost of testing a pair of primitives for proximity (e.g., overlaps or distance computation).

Typically, for tight-fitting bounding volumes, e.g., oriented bounding boxes (OBBs), N_{bv} and N_p are relatively small, whereas C_{bv} is relatively high. In contrast, C_{bv} is low while N_{bv} and N_p may be larger for simple BV types like spheres and axis-aligned bounding boxes (AABBs). Due to these opposing trends, no single BV yields optimum performance for proximity queries in all situations.

35.3 PENETRATION-DEPTH COMPUTATION

In this section, we briefly review ***penetration depth*** (PD) computation algorithms between convex polytopes and general polyhedral models. The PD of two interpenetrating objects A and B is defined as the minimum translation distance that one object undergoes to make the interiors of A and B disjoint. It can be also defined in terms of the TCSO. When two objects are overlapping, the origin of the Minkowski sum of A and $-B$ is contained inside the TCSO. The penetra-

FIGURE 35.3.1
Penetration depth is applied to virtual exploration of a digestive system using haptic interaction to feel and examine different parts of the model. The distance computation and penetration depth computation algorithms are used for disjoint (D) and penetrating (P) situations, respectively, to compute the forces at the contact areas.



tion depth corresponds to the minimum distance from the origin to the surface of TCSO [Cam97]. PD computation is often used in motion planning [HKL+98], contact resolution for dynamic simulation [MZ90, ST96] and force computation in haptic rendering [KOLM02]. Fig. 35.3.1 shows a haptic rendering application of penetration-depth and separation-distance computation. For example, computation of dynamic response in penalty-based methods often needs to perform PD queries for imposing the nonpenetration constraint for rigid body simulation. In addition, many applications, such as motion planning and dynamic simulation, require a continuous distance measure when two (nonconvex) objects collide for a well-posed computation.

Several algorithms for PD computation involve computing Minkowski sums and the closest point on its surface from the origin. The worst-case complexity of the overall PD algorithm is dominated by computing Minkowski sums, which can be $\Omega(n^2)$ for convex polytopes and $\Omega(n^6)$ for general (or nonconvex) polyhedral models [DHKS93]. Given the complexity of Minkowski sums, many approximation algorithms have been proposed in the literature for fast PD estimation.

CONVEX POLYTOPES

Dobkin et al. [DHKS93] proposed a hierarchical algorithm to compute the directional PD using Dobkin and Kirkpatrick polyhedral hierarchy. For any direction d, it computes the directional penetration depth in $O(\log n \log m)$ time for polytopes with m and n vertices. Agarwal et al. [AGHP+00] designed a randomized approach to compute the PD values [AGHP+00], achieving $O(m^{\frac{3}{4}+\epsilon}n^{\frac{3}{4}+\epsilon} + m^{1+\epsilon} + n^{1+\epsilon})$ expected time for any positive constant ϵ. Cameron [Cam97] presented an extension to the GJK algorithm [GJK88] to compute upper and lower bounds on the PD between convex polytopes. Bergen further elaborated this idea in an expanding polytope algorithm [Ber01]. The algorithm iteratively improves the result of the PD computation by expanding a polyhedral approximation of the Minkowski sums of two polytopes. Kim et al. [KLM02] presented an incremental algorithm that marches toward a "locally optimal" solution by walking on the surface of the Minkowski sum. The surface of the TCSO is implicitly computed by constructing a local Gauss map and performing a local walk on the polytopes.

POLYHEDRAL MODELS

Algorithms for penetration-depth estimation between general polygonal models are based on discretization of the object space containing the objects, or use of digital geometric algorithms that perform computations on a finite resolution grid. Fisher and Lin [FL01] presented a PD estimation algorithm based on the distance-field computation using the fast marching level-set method. It is applicable to all polyhedral objects as well as deformable models, and it can also check for self-penetration. Hoff et al. [HZLM01, HZLM02] proposed an approach based on performing discretized computations on graphics rasterization hardware. It uses multi-pass rendering techniques for different proximity queries between general rigid and deformable models, including penetration depth estimation. Kim et al. [KLM02] presented a fast approximation algorithm for general polyhedral models using a combination of object-space as well discretized computations. Given the global nature of the PD problem, it decomposes the boundary of each polyhedron into convex pieces, computes the pairwise Minkowski sums of the resulting convex polytopes and uses graphics rasterization hardware to perform the closest-point query up to a given discretized resolution. The results obtained are refined using a local walking algorithm. To further speed up this computation and improve the estimate, the algorithm uses a hierarchical refinement technique that takes advantage of geometry culling, model simplification, accelerated ray-shooting, and local refinement with greedy walking. The overall approach combines discretized closest-point queries with geometry culling and refinement at each level of the hierarchy. Its accuracy can vary as a function of the discretization error.

OTHER METRICS

Other metrics to characterize the intersection between two objects include the ***growth distance*** defined by Gilbert and Ong [GO94]. This is a consistent distance measure regardless of whether the objects are disjoint or overlapping; it is differs from the PD between two interpenetrating convex objects.

35.4 SPLINE AND ALGEBRAIC OBJECTS

Most of the algorithms highlighted above are limited to polygonal objects. In many applications of geometric and solid modeling, curved objects whose boundaries are described using rational splines or algebraic equations are used (cf. Chapter 53). Algorithms to perform different proximity queries on these objects may be classified by methods: subdivision methods, tracing methods, and analytic methods. See [Pra86, Hof89, Man92] for surveys. Next, we briefly enumerate these methods.

SUBDIVISION METHODS

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 [LR80]. Depending on various criteria, the domains 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 some finite precision.

TRACING METHODS

The tracing method begins with a given point known to be on the intersection curve [BFJP87, MC91, KM97]. Then the intersection curve is traced in sufficiently small steps until the edge of the patch is found, or until the curve loops back to itself. In practice, it is easy to check for intersections with a patch boundary, but difficult to know when the tracing point has returned to its starting position. Frequently this is posed as an initial-value differential equations problem [KPW90], or as solving a system of algebraic equations [MC91, KM97, LM97]. 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.

ANALYTIC METHODS

Analytic methods usually involve implicitizing one of the parametric surfaces—obtaining an implicit representation of the model [SAG84, MC92]. 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 $\mathbb{R}$. Substituting the parametric functions $f_x(u, v), f_y(u, v), f_z(u, v)$ for x, y, z of the implicit function leads to 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 [AB88, KM97, KCMh99].

35.5 DYNAMIC QUERIES

In this section we give a brief overview of algorithms used to perform *dynamic queries*. Unlike static queries, which check for collisions or perform separation-distance queries at discrete instances, these algorithms use continuous techniques based on the object motion to compute the time of first collision.

Many algorithms assume that the motion of the objects can be expressed as a closed-form function of time. Cameron [Cam90] has presented algorithms that pose the problem as interference computation in a 4-dimensional space. Given a parametric representation of each object's boundary as well as its motion, Herzen et al. [HBZ90] presented a collision detection algorithm that subdivides the domain of the surface, including the time dimension. They use Lipschitz conditions, based on bounds on the various derivatives of the mapping, to compute bounds on the extent of the resulting function. The bounds are used to check two objects for overlap. Snyder et al. [Sea93] improved the runtime performance of this algorithm by introducing more conditions that prune the search space for collisions and combined it with interval arithmetic [Moo79].

Other continuous techniques use the object motion to estimate the time of first contact. For prespecified trajectories consisting of a sequence of individual translations and rotations about an arbitrary axis, Boyse [Boy79] presented an algorithm for detecting and analyzing collisions between a moving and a stationary objects. Canny [Can86] described an algorithm for computing the exact points of collision for objects that are simultaneously translating and rotating. It can deal with any path in the space that can be expressed as a polynomial function of time. Given bounds on the maximum velocity and acceleration of the objects are known, Lin [Lin93] presented a scheduling scheme that maintains a priority queue and sorts the object based on approximate time to collision. The approximation is computed from the separation distance as well as from bounds on velocity and acceleration. Redon et al. [RKC00] proposed an algorithm that replaces the unknown motion between two discrete instances by an arbitrary rigid motion. It reduces the problem of computing the time of collision to computing a root of a univariate cubic polynomial.

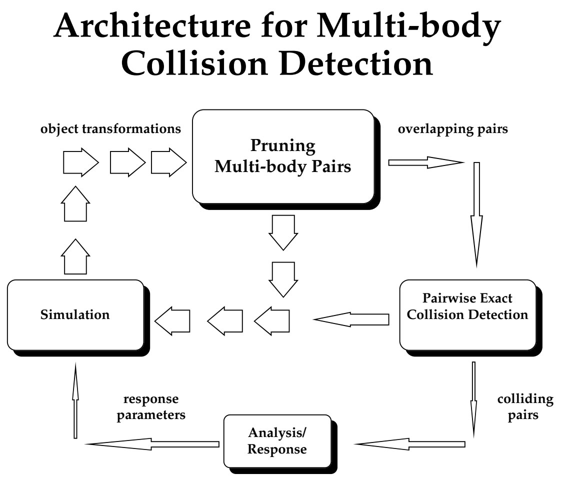
35.6 LARGE ENVIRONMENTS

Large environments are composed of multiple moving objects. Different methods have been proposed to overcome the bottleneck of $O(n^2)$ pairwise tests in an environment composed of n objects. The problem of performing proximity queries in large environments is typically divided into two parts [Hub95, CLMP95]: the *broad phase*, in which we identify the pair of objects on which we need to perform different proximity queries, and the *narrow phase*, in which we perform the exact pairwise queries. An architecture for multi-body collision detection algorithm is shown in Figure 35.6.1. In this section, we present a brief overview of algorithms used in the broad phase.

The simplest algorithms for large environments are based on spatial subdivisions. The space is divided into cells of equal volume, and at each instance the objects are assigned to one or more cells. Collisions are checked between all object pairs belonging to each cell. In fact, Overmars presented an efficient algorithm based on hash table to efficiently perform point location queries in fat subdivisions [Ove92] (see also Chapter 34). This approach works well for sparse environments in which the objects are uniformly distributed through the space. Another approach operates directly on 4D volumes swept out by object motion over time [Cam90]. Efficient algorithms for maintenance and self-collision testing for kinematic chains composed of multiple links have been been presented in [LSHL02].

Several algorithms compute an axis-aligned bounding box (AABB) for each object, based on their extremal points along each direction. Given n bounding boxes, they check which boxes overlap in space. A number of efficient algorithms are known for the static version of this problem. In 2D, the problem reduces to checking 2D intervals for overlap using interval trees and can be performed in

FIGURE 35.6.1
Typically, the object's motion is constrained by collisions with other objects in the simulated environment. Depending on the outcome of the proximity queries, the resulting simulation computes an appropriate response.



$O(n \log n + s)$ where s is the total number of intersecting rectangles [Ede83]. In 3D, algorithms of complexity $O(n \log^2 n + s)$ complexity are known, where s is the number of overlapping pairwise bounding boxes [HSS83, HD82]. Algorithms for N-body proximity queries in dynamic environments are based on the ***sweep and prune*** approach [CLMP95]. This incrementally computes the AABBs for each object and checks them for overlap by computing the projection of the bounding boxes along each dimension, and sorting the interval endpoints using insertion sort or bubble sort [MD76, Bar92, CLMP95]. In environments where the objects make relatively small movements between successive frames, the lists can be sorted in expected linear time, leading to expected-time $O(n + m)$, where m is the number of overlapping intervals along any dimension. These algorithms are limited to environments where objects undergo rigid motion. Govindaraju et al. [GRLM03] have presented a general algorithm for large environments composed of rigid as well as nonrigid motion. This algorithm uses graphics hardware to prune the number of objects that are in close proximity and eventually checks for overlapping triangles between the objects. In practice, it works well in large environments composed of nonrigid and breakable objects. However, its accuracy is governed by the resolution of the rasterization hardware.

OUT-OF-CORE ALGORITHMS

In many applications, it may not be possible to load a massive geometric model composed of millions of primitives in the main memory for interactive proximity queries. In addition, algorithms based on spatial partitioning or bounding volume hierarchies also add additional memory overhead. Thus, it is important to develop proximity-query algorithms that use a relatively small or bounded memory footprint.

Wilson et al. [WLML99] presented an out-of-core algorithm to perform collision and separation-distance queries on large environments. It uses ***overlap graphs*** to exploit locality of computation. For a large model, the algorithm automatically encodes the proximity information between objects and represents it using an overlap graph. The overlap graph is computed off-line and preprocessed using graph partitioning, object decomposition, and refinement algorithms. At run time it traverses localized subgraphs and orders the computations to check the corresponding geometry for proximity tests, as well as pre-fetch geometry and associated hierarchical data structures. To perform interactive proximity queries in dynamic environments, the algorithm uses the BVHs, modifies the localized subgraph(s) on the fly, and takes advantage of spatial and temporal coherence.

35.7 PROXIMITY QUERY PACKAGES

Many systems and libraries have been developed for performing different proximity queries. These include:

I-COLLIDE: I-COLLIDE is an interactive and exact collision-detection system for environments composed of convex polyhedra or union of convex pieces. The system is based on the LC incremental distance computation algorithm [LC91] and an algorithm to check for collision between multiple moving objects [CLMP95]. It takes advantage of temporal coherence. `http://gamma.cs.unc.edu/I_COLLIDE`.

RAPID: RAPID is a robust and accurate interference detection library for a pair of unstructured polygonal models. It is applicable to polygon soups—models which contain no adjacency information and obey no topological constraints. It is based on OBBTrees and uses a fast overlap test based on Separating Axis Theorem to check whether two OBBs overlap [GLM96]. `http://gamma.cs.unc.edu/OBB/OBBT.html`

V-COLLIDE: V-COLLIDE is a collision detection library for large dynamic environments [HLC$^+$97], and unites the N-body processing algorithm of I-COLLIDE with the pair processing algorithm of RAPID. Consequently, it is designed to operate on large numbers of static or moving polygonal objects, and the models may be unstructured. `http://gamma.cs.unc.edu/V_COLLIDE`

Enhanced GJK Algorithm: It is a library for distance computation based on the enhanced GJK algorithm [GJK88] developed by Cameron [Cam97]. It takes advantage of temporal coherence between successive frames. `http://www.comlab.ox.ac.uk/oucl/users/stephen.cameron/distances.html`

SOLID: SOLID is a library for interference detection of multiple 3D polygonal objects undergoing rigid motion. The shapes used by SOLID are polygon soups. The library exploits frame coherence by maintaining a set of pairs of proximate objects using incremental sweep and pruning on hierarchies of axis-aligned bounding boxes. Though slower for close proximity scenarios, its performance is comparable to that of V-COLLIDE in other cases. `http://www.win.tue.nl/cs/tt/gino/solid`

PQP: PQP, a Proximity Query Package, supports collision detection, separation-distance computation or tolerance verification. It uses OBBTree for collision queries and a hierarchy of swept-sphere volumes to perform distance queries [LGLM99]. It assumes that each object is a collection of triangles and can handle polygon soup models. `http://gamma.cs.unc.edu/SSV`

SWIFT: SWIFT a library for collision detection, distance computation, and contact determination between 3D polygonal objects undergoing rigid motion. It assumes that the input primitives are convex polytopes or a union of convex pieces. The underlying algorithm is based on a variation of LC [EL00]. The resulting system is faster, more robust and memory efficient than I-COLLIDE. `http://gamma.cs.unc.edu/SWIFT`

SWIFT++: SWIFT++ a library for collision detection, approximate and exact distance computation, and contact determination between closed and bounded polyhedral models. It decomposes the boundary of each polyhedra into convex patches and precomputes a hierarchy of convex polytopes [EL01]. It uses the SWIFT library to perform the underlying computations between the bounding volumes. `http://gamma.cs.unc.edu/SWIFT++`

QuickCD: QuickCD is a general-purpose collision detection library, capable of performing exact collision detection on complex models. The input model is a collection of triangles, with assumptions on the structure or topologies of the model. It precomputes a hierarchy of k-DOPs for each object and uses them to perform fast collision queries [KHM+98]. `http://www.ams.sunysb.edu/~jklosow/quickcd/QuickCD.html`

OPCODE: OPCODE is a collision detection library between general polygonal models. It uses a hierarchy of AABBs. It is memory efficient in comparison to RAPID, SOLID, or QuickCD. `http://www.codercorner.com/Opcode.htm`

DEEP: DEEP estimates the penetration depth and the associated penetration direction between two overlapping convex polytopes. It uses an incremental algorithm the computes a "locally optimal solution" by walking on the surface of the Minkowski sum of two polytopes [KLM02]. `http://gamma.cs.unc.edu/DEEP`

PIVOT: PIVOT computes generalized proximity information between arbitrary objects using graphics hardware. It uses multipass rendering techniques and accelerated distance computation, and provides an approximate solution for different proximity queries. These include collision detection, distance computation, local penetration depth, contact region and normals, etc. [HZLM01, HZLM02]. It involves no preprocessing and can handle deformable models. `http://gamma.cs.unc.edu/PIVOT`

RELATED CHAPTERS

Chapter 23. Voronoi diagrams and Delaunay triangulations
Chapter 34. Point location
Chapter 38. Geometric intersection
Chapter 47. Algorithmic motion planning
Chapter 50: Modeling Motion
Chapter 54. Surface simplification and 3D geometry compression
Chapter 64. Software

REFERENCES

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

[AGHP+00] P. Agarwal, L.J. Guibas, S. Har-Peled, A. Rabinovitch, and M. Sharir. Penetration depth of two convex polytopes in 3d. *Nordic J. Computing*, 7:227–240, 2000.

[Bar92] D. Baraff. *Dynamic simulation of non-penetrating rigid body simulation.* Ph.D. thesis, Cornell Univ., 1992.

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

[BEG+99] J. Basch, J. Erickson, L. Guibas, J. Hershberger, and L. Zhang. Kinetic collision detection between two simple polygons. In *Proc. 10th Sympos. Discrete Algorithms*, pages 102–111, 1999.

[Ber01] G. Bergen. Proximity queries and penetration depth computation on 3d game objects. *Game Developers Conf.*, 2001.

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

[BKSS90] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *Proc. SIGMOD Conf. Management Data*, pages 322–331, 1990.

[Boy79] J.W. Boyse. Interference detection among solids and surfaces. *Commun. ACM*, 22:3–9, 1979.

[Cam90] S. Cameron. Collision detection by four-dimensional intersection testing. *Proc. Internat. Conf. Robot. Autom.*, pages 291–302, 1990.

[Cam97] S. Cameron. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *Proc. Internat. Conf. Robot. Autom.*, pages 3112–3117, 1997.

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

[CC86] S. Cameron and R.K. Culley. Determining the minimum translational distance between two convex polyhedra. *Proc. Internat. Conf. Robot. Autom.*, pages 591–596, 1986.

[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. ACM Interactive 3D Graphics Conf.*, pages 189–196, 1995.

[CW96] K. Chung and W. Wang. Quick collision detection of polytopes in virtual environments. In *Proc. ACM Sympos. Virtual Reality Soft. Tech.*, 1996.

[DHKS93] D. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9:518–533, 1993.

[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.

[DZ93] P. Dworkin and D. Zeltzer. A new model for efficient dynamics simulation. *Proc. EG Workshop Comput. Animat. Simul.*, pages 175–184, 1993.

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

[EGSZ99] J. Erickson, L. Guibas, J. Stolfi, and L. Zhang. Separation sensitive collision detection for convex objects. *Proc. 10th ACM-SIAM Sympos. Discrete Algorithms*, pages 327–336, 1999.

[EL00] S. Ehmann and M.C. Lin. Accelerated proximity queries between convex polyhedra using multi-level Voronoi marching. *Proc. IEEE/RSJ Internat. Conf. Intell. Robots Sys.*, pages 2101–2106, 2000.

[EL01] S. Ehmann and M.C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Comput. Graph. Forum*, 20(3), pages 500–510, 2001.

[FL01] S. Fisher and M.C. Lin. Deformed distance fields for simulation of non-penetrating flexible bodies. *Proc. EG Workshop Comput. Animat. Simul.*, pages 99–111, 2001.

[GHZ99] L. Guibas, D. Hsu, and L. Zhang. *H-Walk*: Hierarchical distance computation for moving convex bodies. *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 265–273, 1999.

[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. Robot. Autom.*, 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. ACM Conf. SIGGRAPH 96*, pages 171–180, 1996.

[GO94] E.G. Gilbert and C.J. Ong. New distances for the separation and penetration of objects. In *Proc. Internat. Conf. Robot. Autom.*, pages 579–586, 1994.

[Got00] S. Gottschalk. *Collision Queries using Oriented Bounding Boxes*. Ph.D. thesis, Univ. North Carolina. Dept. Computer Science, 2000.

[GRLM03] N. Govindraju, S. Redon, M. Lin and D. Manocha. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In *Proc. ACM SIGGRAPH/Eurographics Workshop Graphics Hardware*, pages 25–32, 2003.

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

[HD82] H.Six and D.Wood. Counting and reporting intersections of D-ranges. *IEEE Transactions on Computers*, pages 46–55, 1982.

[HKL$^+$98] D. Hsu, L. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. *Proc. 3rd Workshop Algorithmic Found. Robot.*, pages 141–154, 1998.

[HKM95] M. Held, J.T. Klosowski, and J.S.B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 205–210, 1995.

[HKM96] M. Held, J. Klosowski, and J.S.B. Mitchell. Real-time collision detection for motion simulation within complex environments. In *ACM SIGGRAPH 96 Visual Proc.*, page 151, 1996.

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

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

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

[Hub95] P.M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graphics*, 15:179–210, 1995.

[HZLM01] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast and simple geometric proximity queries using graphics hardware. *Proc. ACM Sympos. Interactive 3D Graphics*, pages 145–148, 2001.

[HZLM02] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast 3D geometric proximity queries between rigid and deformable models using graphics hardware acceleration. Tech. Rep. TR02-004, Dept. of Comput. Sci., Univ. North Carolina, 2002.

[KCMh99] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. MAPC: A library for efficient and exact manipulation of alge braic points and curves. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 360–369, 1999.

[KGL$^+$98] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar. Rapid and accurate contact determination between spline models using shelltrees. *Computer Graphcis Forum*, 17:C315–C326, 1998.

[KHM$^+$98] J. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Trans. Visualization Comput. Graph.*, 4:21–37, 1998.

[KLM02] Y. Kim, M. Lin, and D. Manocha. Deep: An incremental algorithm for penetration depth computation between convex polytopes. *Proc. IEEE Conf. Robot. Autom.*, pages 921–926, 2002.

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

[KOLM02] Y. Kim, M. Otaduy, M. Lin, and D. Manocha. 6-DOF haptic display using localized contact computations. *Proc. Haptics Sympos.*, pages 209–216, 2002.

[KPLM98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shell: A higher order bounding volume for fast proximity queries. In *Proc. 3rd Internat. Workshop Algorithmic Found. Robot.*, pages 122–136, 1998.

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

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

[KSS02] D. Kirkpatrick, J. Snoeyink, and B. Speckman. Kinetic collision detection for simple polygons. *Internat. J. Comput. Geom. Appl.*, 12:3–27, 2002.

[LSHL02] I. Lotan, F. Schwarzer, D. Halperin and J.-C. Latombe. Efficient maintenance and self-collision testing for kinematic chains. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, pages 43–52, 2002.

[LC91] M.C. Lin and J.F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Conf. Robot. Autom.*, pages 1008–1014, 1991.

[LGLM99] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Tech. Rep. TR99-018, Dept. of Comput. Sci., Univ. North Carolina, 1999.

[Lin93] M.C. Lin. *Efficient Collision Detection for Animation and Robotics.* Ph.D. thesis, Dept. Elec. Eng. Comput. Sci., Univ. California, Berkeley, 1993.

[LM97] M.C. Lin and D. Manocha. Efficient contact determination between geometric models. *Internat. J. Comput. Geom. Appl.*, 7:123–151, 1997.

[LR80] J.M. Lane and R.F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2:150–159, 1980.

[Man92] D. Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics.* Ph.D. thesis, Dept. Elec. Eng. Comput. Sci, Univ. California, Berkeley, 1992.

[MC91] D. Manocha and J.F. Canny. A new approach for surface intersection. *Internat. Comput. Geom. Appl.*, 1:491–516, 1991. Special issue on Solid Modeling.

[MC92] D. Manocha and J.F. Canny. Algorithms for implicitizing rational parametric surfaces. *Comput. Aided Geom. Design*, 9:25–50, 1992.

[MD76] M.Shamos and D.Hoey. Geometric intersection problems. *Proc. 17th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 208–215, 1976.

[Mir98] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. *ACM Trans. Graph.*, 17:177–208, 1998.

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

[MZ90] M. McKenna and D. Zeltzer. Dynamic simulation of autonomous legged locomotion. *Proc. ACM Conf. SIGGRAPH 90*, pages 29–38, 1990.

[NAT90] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yield polyhedral modeling results. In *Proc. ACM Conf. SIGGRAPH 90*, pages 115–124, 1990.

[OL03] M. Otaduy and M. Lin. CLODs: Dual hierarchies for multiresolution collision detection. In *Proc. Eurographics Sympos. Geom. Processing*, pages 94–101, 2003.

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

[PML97] M. Ponamgi, D. Manocha, and M. Lin. Incremental algorithms for collision detection between solid models. *IEEE Trans. Visualization Comput. Graph.*, 3:51–67, 1997.

[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.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. Internat. Conf. Robot. Autom.*, pages 3324–3329, 1994.

[RKC00] S. Redon, A. Kheddar, and S. Coquillart. An algebraic solution to the problem of collision detection for rigid polyhedral objects. *Proc. IEEE Conf. Robot. Autom.*, 2000.

[SAG84] T.W. Sederberg, D.C. Anderson, and R.N. Goldman. Implicit representation of parametric curves and surfaces. *Comput. Vision Graph. Image Process.*, 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. *Comput. Vision Graph. Image Process.*, 22:222–238, 1983.

[Sea93] J. Snyder, A.R. Woodbury, K. Fleischer, B. Currin, A.H. Barr. Interval methods for multi-point collisions between time dependent curved surfaces. In *Proc. ACM Conf. SIGGRAPH 93*, pages 321–334, 1993.

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 211–215, Berkeley, California, 1990.

[ST96] D.E. Stewart and J.C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *Internat. J. Numerical Methods in Engineering*, 39:2673–2691, 1996.

[WG91] W. Bouma and G. Vanecek. Collision detection and analysis in a physically based simulation. *Proc. EG Workshop Comput. Animat. Simul.*, pages 191–203, 1991.

[WLML99] A. Wilson, E. Larsen, D. Manocha, and M.C. Lin. Partitioning and handling massive models for interactive collision detection. *Comput. Graph. Forum*, 18:319–329, 1999.