40 COLLISION AND PROXIMITY QUERIES

Ming C. Lin, Dinesh Manocha and Young J. Kim

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. 40.0.1. 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. 40.0.2, 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 40.1), nonconvex polygonal models (Section 40.2), continuous collision detection (Section 40.3), penetration depth queries (Section 40.4), Hausdorff distance queries (Section 40.5), curved objects (Section 40.6), and large environments composed of multiple objects (Section 40.7). Finally, it briefly describes different software packages available to perform some of the queries (Section 40.8).

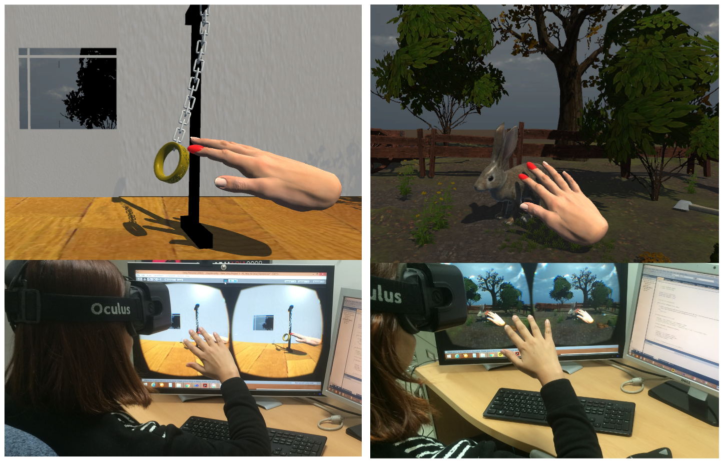
PROBLEM CLASSIFICATION

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 40.0.1
A hand reaching toward a ring and a bunny at top. The corresponding image of the user in the real world is shown at bottom. Red finger tips indicate contacts between the user's hand and the virtual ring and bunny.



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 between two overlapping objects needed to translate one object to make it just touch the other:

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

Generalized Penetration Depth: Minimal rigid motion under some distance metric d to make one object just touch the other:

$$\text{gpd}(A, B) = \text{minimum } ||\mathbf{M}||_d \text{ such that} \min_{a \in A} \min_{b \in B} |\mathbf{a} - \mathbf{M}\mathbf{b}| \geq 0.$$

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.

FIGURE 40.0.2
In this virtual prototyping application, the motion of the pistons is simulated to check for tolerances by performing distance queries.

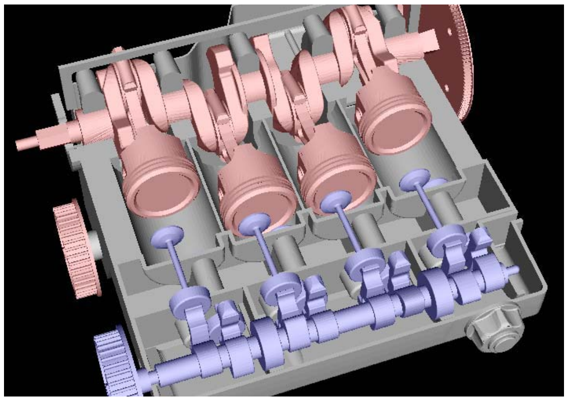


TABLE 40.0.1 Classification of Proximity Queries.

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

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, approximate, and Boolean forms. These queries are called ***dynamic queries***, whereas the ones that do not use motion information are called ***static queries***. In a 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 40.0.1.

40.1 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 optimal runtime algorithm for Boolean collision queries takes $O(\log n)$ time, where n is the number of features [BL15]. It precomputes the bounded Dobkin-Kirkpatrick (BDK) hierarchy for each polytope in linear time and space 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.

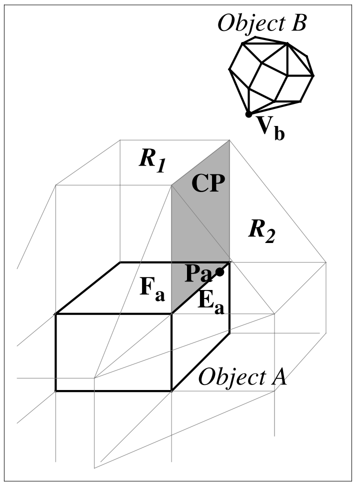


FIGURE 40.1.1
A walk across external Voronoi region of Object A. Vertex $\mathbf{V}_b$ *of Object B lies in the Voronoi region of* $\mathbf{E}_a$*.*

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 40.1.1. 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 a philosophy similar to that of 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 40.1.1 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

40.2 GENERAL POLYGONAL MODELS

Algorithms for collision and separation-distance queries between general polygon models can be classified based on whether they assume closed polyhedral models, or are represented as a collection of polygons. The latter, also referred to as "polygon soups," makes 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 [Hub93, Qui94], ellipsoid-trees [LWH$^+$07], AABB-trees [BKSS90, HKM95, PML97], OBB-trees [GLM96, BCG$^+$96, Got00], spherical shell-trees [KPLM98, KGL$^+$98], k-DOP-trees [HKM96, KHM$^+$98], SSV-trees [LGLM99], multiresolution hierarchies [OL03], convex hull-trees [EL01], SCB-trees [LAM09], and k-IOS trees [ZK12], as shown in Table 40.2.1. Readers are referred to a book written by Ericson [Eri04] for more details on spatial partitioning and BVHs.

TABLE 40.2.1 Types of bounding volume hierarchies.

NAME	TYPE OF BOUNDING VOLUME
Sphere-tree	Sphere
Ellipsoid-tree	Ellipsoid
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
SCB-tree	Slab cut ball (SCB)
k-IOS-tree	Intersection of k-spheres (IOS)

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 by 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.

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 finding out 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 sub-algorithms used for proximity queries on a pair of BVs.

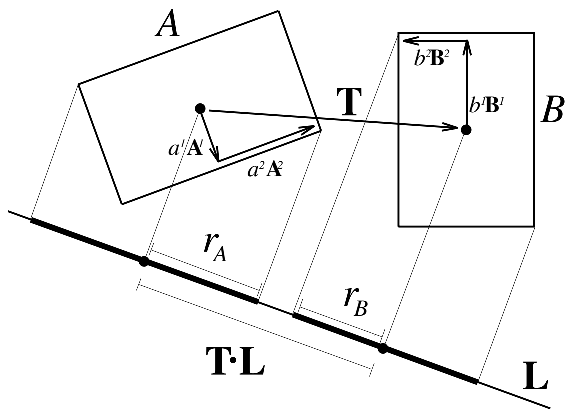


FIGURE 40.2.1
L *is a separating axis for OBBs A and B because projection onto* **L** *renders them disjoint intervals.*

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 40.2.1 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 half-plane defined by two lines. The two lines and the circles belong to the same plane.

It is also possible to devise parallel algorithms to traverse the bounding volume hierarchy using multi-core CPUs or many-core GPUs. Lee and Kim [LK10] propose parallel proximity algorithms using heuristic task decomposition to perform both Euclidean distance calculation and collision detection between rigid models. Many parallel algorithms [LMM10, TMLT11] exploit thread and data parallelism on many-core GPUs to perform fast hierarchy construction, updates, and traversal using bounding volumes for rigid and deformable models.

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.

40.3 CONTINUOUS COLLISION DETECTION

Earlier collision detection algorithms only check for collisions at sample configurations. As a result, they may miss a collision that occurs between two successive configurations or time steps. This is sometimes known as the tunneling problem, because it typically occurs when a rapidly moving object passes undetected through a thin obstacle. Continuous collision detection (CCD) algorithms avoid the tunneling problem by interpolating a continuous motion trajectory between successive configurations and checking for collisions along that trajectory. If a collision occurs, the first time of contact (ToC) between the moving objects is reported.

RIGID MODELS

At a broad level, CCD algorithms can be classified into algebraic equation solvers, swept-volume formulations, adaptive bisection approaches, kinetic data structures (KDS), Minkowski sum formulations, and conservative advancement (CA).

Algebraic equation solvers [Can86, CWLK06, KR03, RKC00] compute the ToC by numerically finding roots of low-order polynomials, while swept-volume formulations [Cam90, AMBJ06] operates directly on 4D volumes swept out by object motion over time. ***Kinetic data structures*** [ABG$^+$00, KGS97, KSS02] are based on the formal framework of KDS to keep track of collision events of models during their motion and exploits motion coherence and geometric locality. ***Minkowski sum formulations*** [Ber04] pose a CCD problem as ray shooting against the translational configuration space obstacle, which is equivalent to Minkowski sums between two translating models. ***Adaptive bisection approaches*** [SWF$^+$93, HBZ90, RKC02, SSL02] find the ToC by adaptively subdividing the expected collision-time interval, and ***conservative advancement*** [Lin93, Mir96, ZKM08, TKM09, TKM10] conservatively advances the time step while avoiding collision until it reaches the ToC.

Most of these approaches are unable to perform fast CCD queries on general polygonal models, although some can handle polygon-soup models [RKC02, SSL02]. Redon et al. [RKC02] use a continuous version of the separating axis theorem to extend the static OBB-tree algorithm [GLM96] to CCD, and demonstrate real-time performance on polygonal models. But the algorithm becomes overly conservative when there is a large rotation between two configurations. For polyhedral models, there is a faster algorithm [ZKM08] using conservative advancement that finds the ToC by comparing an upper bound of motion trajectory against the velocity of the models. Tang et al. [TKM09] use controlled advancement to detect collision for polygon-soup model by adaptively adjusting the advancement step, and they extend this work by using different acceleration techniques. This approach is also applicable to articulated models [TMK14]. FCL [PCM12] also uses an implementation of [TKM09], which forms part of a generic collision and proximity detection library.

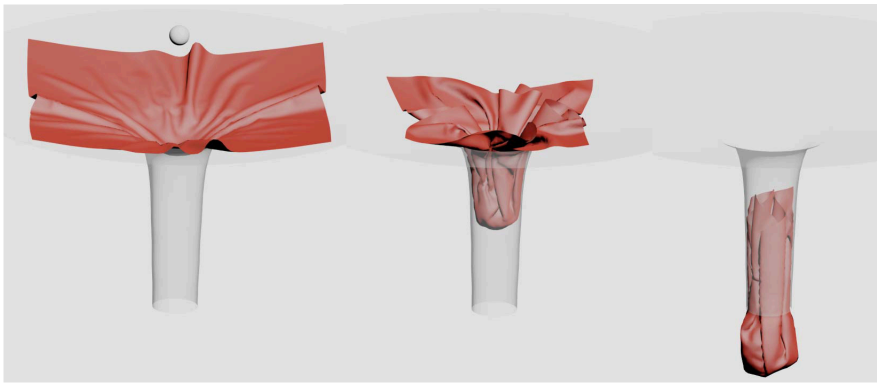
ARTICULATED MODELS

A conservative condition is used in [SSL02] to guarantee a collision-free motion between two configurations, but such a condition is likely to become overly conservative when an object slides over another object. Redon et al. [RKLM04] describe an extension of their previous algorithm [RKC02] to articulated models, but this algorithm is relatively slow for complicated objects. Zhang et al. have extended their approach [ZKM08] to articulated models [ZRLK07] by modeling each link as a polyhedron.

DEFORMABLE MODELS

CCD algorithms for deformable models compute all possible times of contact between pairs of overlapping primitive during the motion, including self-collisions between the primitives. In order to perform a continuous collision query between two triangle primitives, an approach based on algebraic equation solvers was first introduced by Moore and Wilhelms [MW88] using fifth-order algebraic equations. These equations were further reduced to cubic by taking into account co-planarity

FIGURE 40.3.1
In this simulation, a cloth falls into a funnel and pass through it under the pressure of a ball. This model has 47K vertices, 92K triangles, and a lot of self-collisions. The GPU-based CCD algorithm [TMLT11] takes 4.4ms and 10ms per frame to compute all the collisions.



constraints. In this case, detecting collisions between deforming triangles corresponds to performing pairwise six face-vertex and nine edge-edge elementary tests. Each elementary test reduces to solving a cubic algebraic equation [Pro97, BFA02]. Some filters can be used to accelerate these tests [TMT10a]. Another approach based on conservative advancement (CA) for deforming objects was presented by Tang et al. [TKM10], which was extended from [ZLK06, ZRLK07, TKM09]. The CA-based algorithms are able to avoid solving high-order algebraic equations. In order to avoid redundant elementary tests, the connectivity and adjacency information of a deformable mesh can be used [GKJ$^+$05, TYM08] or feature-based hierarchies [CTM08] can be also employed.

Most CCD algorithms for deformable objects try to reduce the number of self-collision queries [TCYM08] and redundant elementary tests [TMY$^+$11]. The self-collisions can be classified into two types: self-collision between adjacent, and between non-adjacent triangles or the primitives. In practice, the cost of self-collisions can account for 50-90% of the total running time of the algorithm [GKJ$^+$05]. The normal cone technique was proposed to cull non-self-colliding triangles [Pro97] for discrete collision detection and was extended to CCD by Tang et al. [TKM10]. The chromatic decomposition technique can also be used to accelerate self-collisions [GKJ$^+$05]. Schvartzman et al. [SPO10] presented a star-contour test as a sufficient condition to determine whether the contour of a projected surface patch is collision-free or not. These approaches are designed to reduce the number of elementary tests. By exploiting the abundant parallelism some techniques have been proposed to utilize multi-core CPUs [KHY09] [TMT10b], GPUs [LGS$^+$09, TMLT11], and their hybrid combinations [KHH$^+$09].

RELIABLE METHODS

Most CCD algorithms are implemented using finite-precision arithmetic, potentially resulting in a false negative or a false positive. It is important to perform reliable CCD queries, as it is well-known that even a single missed collision can affect the accuracy of the entire simulation system.

The most reliable algorithms are based on exact computations [BEB12] that can perform reliable queries with no false negatives or false positives. Recently, Tang et al. [TTWM14] presented an exact algorithm based on Bernstein sign classification. Both of these methods are based on an exact computation paradigm and use extended precision libraries to perform accurate CCD computations. In practice, these exact arithmetic operations can be expensive for real-time applications. Furthermore, it can be difficult to implement such exact arithmetic operations or libraries on GPUs or embedded processors. The second category of accurate solutions for CCD computations is based on performing floating-point error analysis and using appropriate error tolerances [Wan14]. This approach can be used on any processors that support IEEE floating-point arithmetic operations. The resulting CCD algorithm (SafeCCD) eliminates false negatives altogether but can still result in a high number of false positives. Wang et al. [WTTM15] describe a formulation based on Bernstein sign classification that takes advantage of the geometry properties of Bernstein basis and Bézier curves to perform Boolean collision queries. This approach eliminates all the false negatives and $90 \sim 95\%$ of the false positives.

40.4 PENETRATION-DEPTH COMPUTATION

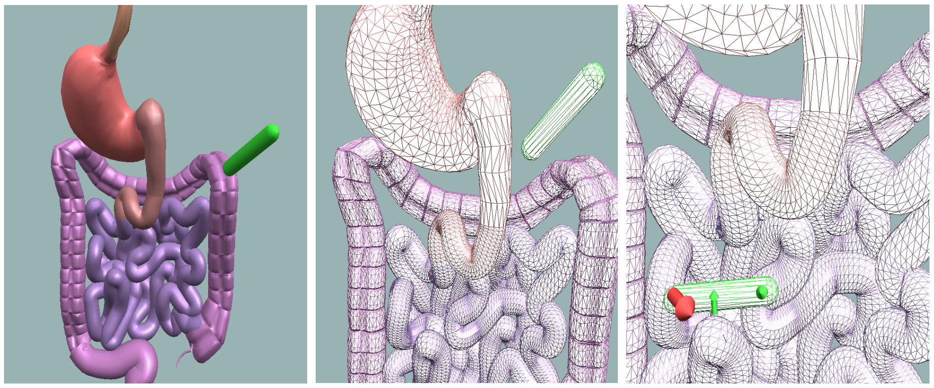
In this section, we briefly review ***penetration depth*** (PD) computation algorithms between convex polytopes and general polyhedral models. The PD of two interpenetrating objects A and B is defined as the minimum translation distance that one object undergoes to make the interiors of A and B disjoint. 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 penetration 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. 40.4.1 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 non-penetration 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,

FIGURE 40.4.1
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.



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 as 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. Recently, Je et al. [JTL$^+$12] propose a real-time algorithm that finds the PD between general polygonal models based on iterative and local optimization techniques. The iterative optimization consists of two projection steps, in- and out-projection. The in-projection and out-projection steps are formulated as a variant of the continuous collision detection algorithm, and a linear complementarity problem, respectively. A locally optimal solution is finally obtained using a type of Gauss-Seidel iterative algorithm. This algorithm can process complicated models consisting of tens of thousands triangles at interactive rates.

GENERALIZED PENERATION-DEPTH COMPUTATION

Conventional penetration-depth algorithms tend to overestimate the amount of penetration, as the resulting separating motion is often limited to translation only. Another set of algorithms [ZKVM07] use a different penetration measure, called generalized PD, which is defined as a minimal rigid motion used to separate overlapping objects; here, the minimality is based on some distance metric in configuration space such as D_g [ZKVM07], S and geodesic [NPR09], displacement [ZKM07b, ZKM07a], and object norm [ZKM07a]. However, the exact computation of generalized PD may require an arrangement of high dimensional contact surfaces, resulting in $O(n^{12})$ combinatorial complexity for a rigid model with n triangles in 3D [ZKVM07]. Thus, all existing approaches use an approximate method.

Very few algorithms exist to compute generalized PD. Zhang et al. [ZKVM07] first proposed the notion of generalized PD, and presented a method to compute only the lower and upper bounds of generalized PD for rigid models. Later, the same authors proposed a more efficient technique, using a contact-space sampling and displacement metric [ZKM07a]. Nawratil et al. [NPR09] presented methods based on kinematical geometry, using S and geodesic metrics. However, all of these methods are rather slow for interactive applications, and it is not clear whether they are applicable to articulated models. Recently, Tang and Kim [TK14] extended the idea of iterative, contact-space projection techniques [JTL$^+$12] to the generalized PD problem. This algorithm is capable of computing the generalized PD for both rigid and articulated models at interactive rates. Pan et al. [PZM13] presented an algorithm to approximate the boundary of C-obstacle space using active learning and use that approximation to compute the PD.

OTHER METRICS

Other metrics used 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.

40.5 HAUSDORFF DISTANCE

Since the seminal work by [Ata83], different algorithms for calculating Hausdorff distances have been proposed in the literature. In this section, we briefly survey work relevant to Hausdorff distance computation for polygonal models in $\mathbb{R}^2$ and in $\mathbb{R}^3$. We refer readers to [AG00, TLK09] for more extensive surveys of the field.

For $\mathbb{R}^2$, [Ata83] presented a linear-time algorithm for convex polygons. For simple, non-convex polygons with n and m vertices, [ABB95] presented an $O((n+m)\log(n+m))$-time algorithm based on an observation that the Hausdorff distance can only be realized at the points of intersection between Voronoi boundary surfaces and the polygon edges. In $\mathbb{R}^3$, for polygon-soup models with n triangles, [God98, ABG$^+$03] presented deterministic and randomized algorithms that run respectively in $O(n^5)$ and $O(n^{3+\epsilon})$, where $\epsilon > 0$. [Lla05] proposed an algorithm using random covering, and also demonstrated implementation results for simple, convex ellipsoids. Since then, more practical algorithms have been put forward, but all these algorithms only approximate the Hausdorff distance due to the complexity of the exact computation. [GBK05] use polygon subdivision to approximate the solution within an error bound. [CRS98, ASCE02] sample a polygonal surface to approximate the distance, but no sampling analysis is provided. More recently, a real-time algorithm has been proposed to approximate the Hausdorff distance within a error bound between complex, polygonal models in $\mathbb{R}^3$ using bounding volume hierarchies [TLK09].

40.6 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 used to perform different proximity queries on these objects may be classified by subdivision methods, tracing methods, analytic methods and bounding volume hierarchy methods. See [Hof89, Man92, Far14] 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, SWF$^+$93, MP09]. 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 that 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 pre-images 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]. Recently, a parallel technique utilizing the CPU and GPU multicore architecture has been introduced to find surface-surface intersections [PEK$^+$11].

BOUNDING VOLUME HIERARCHY METHODS

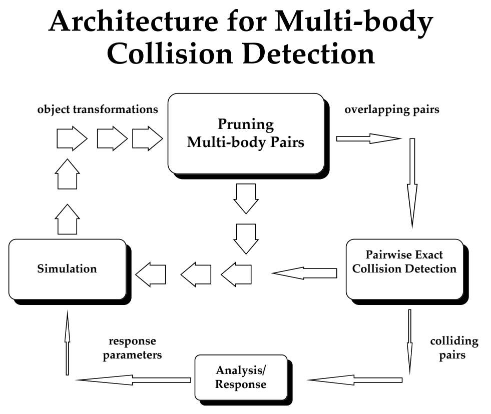
For free form surfaces, BVHs can be also constructed to accelerate distance calculation between surfaces. Kim et al. [KOY$^+$11] proposed the Coons BVH of free form surfaces by recursively computing the bilinear surface of the four corners of patches (i.e. Coons patch) comprising the underlying surface. The resulting Coons BVH can be used to accelerate collision detection and Euclidean distance calculation [KOY$^+$11] as well as Hausdorff distance calculation [KOY$^+$13] while saving the memory space of BVH considerably compared to conventional BVHs for a polyhedral surface.

40.7 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 [Hub93, 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 40.7.1. 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 subdi-

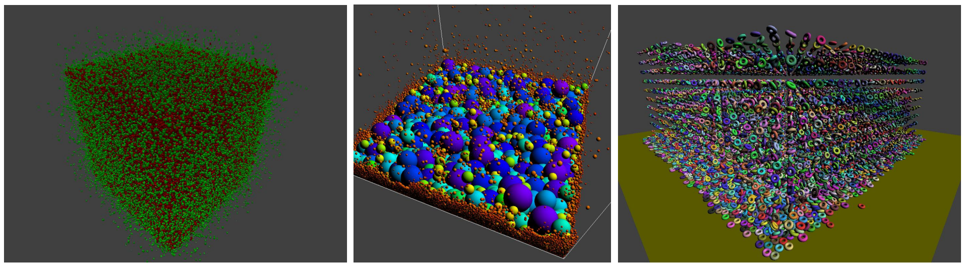
FIGURE 40.7.1
Typically, the object's motion is constrained by collisions with other objects in the simulated environment. Depending on the outcome of the proximity queries, the resulting simulation computes an appropriate response.



visions. 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 tests 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 $O(n \log n + s)$ where s is the total number of intersecting rectangles [Ede83]. In 3D, algorithms of $O(n \log^2 n + s)$ 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 [SH76, 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. A parallel version of sweep and prune has been implemented using graphics hardware by performing sweeps in a massively parallel fashion [LHLK10]. It is capable of finding all overlaps among one million AABB pairs at interactive rates. 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

FIGURE 40.7.2
GPU-based collision detection and simulation at interactive rates for massive bodies [LHLK10]. From left to right: N-body collision detection for 1M arbitrarily moving boxes, simulation of 0.3M particles, rigid-body dynamics for 16K torus models.



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.

40.8 PROXIMITY QUERY PACKAGES

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

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 more 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 earlier collision packages. `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 that 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`

FAST/C^{2}A: These two packages perform continuous collision detection (CCD) between 2-manifold, rigid models and between non-manifold, rigid models, respectively, and report the first time of contact between them as well as their associated contact features. Both packages can perform CCD at interactive rates on a standard PC for complex models consisting of 10K $\sim$ 70K triangles [ZLK06, TKM09]. `http://graphics.ewha.ac.kr/FAST, ~/C2A`

CATCH: CATCH performs CCD between articulated models consisting of 2-manifold links and reports the first time of contact between them as well as their associated contact features. CATCH is also capable of reporting self-collisions [ZRLK07]. `http://graphics.ewha.ac.kr/CATCH`

SelfCCD: SelfCCD can be used for continuous collision detection between deformable models. It can also compute self-collisions and uses many acceleration

techniques described in [CTM08, TCYM08, TMT10a]. It has been used for self-collisions in cloth and other deformable benchmarks.
`http://gamma.cs.unc.edu/SELFCCD`

MCCD: MCCD is a parallel extension of SelfCCD that can exploit the multiple CPU cores to accelerate the collision computations. It has been used to obtain almost linear speedups on 8 or 16 cores for complex benchmarks [TMT10b]. `http://gamma.cs.unc.edu/MCCD`

BSC/TightCCD: These two packages are used to perform reliable continuous collision queries between triangulated models [TTWM14, WTTM15]. They are based on extended precision arithmetic or floating point error bounds.
`http://gamma.cs.unc.edu/BSC`

FCL: FCL is a flexible collision library that can perform multiple queries including collision detection, separation distance, continuous collision detection and penetration depth [PCM12]. Furthermore, it can handle rigid, deformable, articulated and point-cloud models. It has also been integrated into ROS. `http://gamma.cs.unc.edu/FCL`

PolyDepth: PolyDepth estimates the penetration depth and the associated penetration direction between two overlapping polygonal models. PolyDepth calculates a "locally optimal" solution and is also capable of calculating multiple solutions for each overlapping region [JTL$^+$12]. `http://graphics.ewha.ac.kr/polydepth`

gSaP: gSaP cull collisions between very large numbers of moving bodies using graphics processing units (GPUs). gSaP implemented entirely on GPUs using the CUDA framework can handle a million moving objects at interactive rates. [LHLK10]. `http://graphics.ewha.ac.kr/gsap`

Bullet: Bullet is an open source collision detection, rigid body and soft body dynamics library, primarily designed for use in computer games, visual effects and robotic simulation. It supports both discrete and continuous collision detection for convex and non-convex mesh models. `http://bulletphysics.org`

RELATED CHAPTERS

Chapter 29: Voronoi diagrams and Delaunay triangulations
Chapter 39: Point location
Chapter 43: Geometric intersection
Chapter 51: Algorithmic motion planning
Chapter 54: Modeling motion
Chapter 58: Surface simplification and 3D geometry compression
Chapter 69: Software

REFERENCES

[AB88] S.S. Abhyankar and C. Bajaj. Computations with algebraic curves. In *Proc. Int. Symp. Symbolic Algebraic Computation*, vol. 358 of *Lecture Notes Comp. Sci.*, pages 279–284, Springer, Berlin, 1988.

[ABB95] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Ann. Math. Artif. Intell.*, 13:251–266, 1995.

[ABG$^+$00] P.K. Agarwal, J. Basch, L.J. Guibas, J. Hershberger, and L. Zhang. Deformable free space tiling for kinetic collision detection. *Int. J. Robotic Res.*, 21:179–198, 2002.

[ABG$^+$03] H. Alt, P. Braß, M. Godau, C. Knauer, and C. Wenk. Computing the Hausdorff distance of geometric patterns and shapes. *Discrete Comput. Geom.*, 25:65–76, 2003.

[AG00] H. Alt and L.J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 121–153, Elsevier, Amsterdam, 2000.

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

[AMBJ06] K. Abdel-Malekl, D. Blackmore, and K. Joy. Swept volumes: Foundations, perspectives, and applications. *Int. J. Shape Modeling*, 12:87-127, 2006.

[ASCE02] N. Aspert, D. Santa-Cruz, and T. Ebrahimi. Mesh: Measuring errors between surfaces using the Hausdorff distance. In *Proc. IEEE Int. Conf. Multimedia Expo*, pages 705–708, 2002.

[Ata83] M.J. Atallah. A linear time algorithm for the Hausdorff distance between convex polygons. *Inform. Process. Lett*, 17:207–209, 1983.

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

[BCG$^+$96] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. Boxtree: A hierarchical representation of surfaces in 3D. *Comp. Graphics Forum*, 15:387–396, 1996.

[BEB12] T. Brochu, E Edwards, and R. Bridson. Efficient geometrically exact continuous collision detection. *ACM Trans. Graph.*, 31:96, 2012.

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

[Ber04] G. van den Bergen. Ray casting against general convex objects with application to continuous collision detection. Preprint, `http://dtecta.com/papers/jgt04raycast.pdf`, 2004.

[BFA02] R. Bridson, R. Fedkiw, and J. Anderson. Robust treament for collisions, contact and friction for cloth animation. *ACM Trans. Graphics*, 21:594–603, 2002.

[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. on Management of Data*, pages 322–331, 1990.

[BL15] L. Barba and S. Langerman. Optimal detection of intersections between convex polyhedra. In *Proc. 26th ACM-SIAM Sympos. Discrete Algorithms*, pages 1641–1654, 2015.

[Cam90] S. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Trans. Robotics Automation*, 6:291–302, 1990.

[Cam97] S. Cameron. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. In *Proc. IEEE Int. Conf. Robotics Automation*, pages 3112–3117, 1997.

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

[CC86] S. Cameron and R.K. Culley. Determining the minimum translational distance between two convex polyhedra. In *Proc. IEEE Int. Conf. Robotics Automation*, 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.

[CRS98] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Comput. Graphics Forum*, 17:167–174, 1998.

[CTM08] S. Curtis, R. Tamstorf, and D. Manocha. Fast collision detection for deformable models using representative-triangles. *Proc. ACM Sympos. Interactive 3D Graphics Games*, pages 61–69, 2008.

[CW96] K. Chung and W. Wang. Quick collision detection of polytopes in virtual environments. In *Proc. 21st ACM Sympos. Virtual Reality Software and Technology*, 1996.

[CWLK06] Y.-K. Choi, W. Wang, Y. Liu, and M.-S. Kim. Continuous collision detection for elliptic disks. *IEEE Trans. Robotics*, 2006.

[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 Int. Colloq. Automata Lang. Program.*, vol. 443 of *Lecture Notes Comp. Sci.*, pages 400–413, Springer, Berlin, 1990.

[DZ93] P. Dworkin and D. Zeltzer. A new model for efficient dynamics simulation. *Proc. Eurographics Workshop Animation and Simulation*, pages 175–184, 1993.

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

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

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

[Eri04] C. Ericson. *Real-Time Collision Detection.* CRC Press, Boca Raton, 2004.

[Far14] G. Farin. *Curves and Surfaces for Computer-Aided Geometric Design: a Practical Guide.* Elsevier, Amsterdam, 2014.

[FL01] S. Fisher and M. C. Lin. Deformed distance fields for simulation of non-penetrating flexible bodies. *Proc. Eurogaphics Workshop Comp. Animation and Simulation*, pages 99–111, 2001.

[GBK05] M. Guthe, P. Borodin, and R. Klein. Fast and accurate Hausdorff distance calculation between meshes. *J. WSCG* 13:41–48, 2005.

[GHZ99] L. Guibas, D. Hsu, and L. Zhang. H-Walk: Hierarchical distance computation for moving convex bodies. *Proc. Sympos. Comput. Geom.*, ACM Press, 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. Robotics Automation*, RA-4:193–203, 1988.

[GKJ$^+$05] N. Govindaraju, D. Knott, N. Jain, I. Kabal, R. Tamstorf, R. Gayle, M. Lin, and D. Manocha. Collision detection between deformable models using chromatic decomposition. *ACM Trans. Graphics)*, 24:991–999, 2005.

[GLM96] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. *Proc. 23rd Conf. Computer Graphics Interactive Techniques*, pages 171–180, 1996.

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

[God98] M. Godau. *On the Complexity of Measuring the Similarity between Ggeometric Objects in Higher Dimensions.* PhD thesis, Freien Universität, 1998.

[Got00] S. Gottschalk. *Collision Queries using Oriented Bounding Boxes.* PhD thesis, University of North Carolina, 2000.

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

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

[HD82] H. Six and D. Wood. Counting and reporting intersections of D-ranges. *IEEE Trans. Computers*, C-31:181–187, 1982.

[HKL$^+$98] D. Hsu, L. Kavraki, J. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. *Proc. 3rd Workshop Algorithmic Found. Robotics*, pages 25–32, A.K. Peters, Naticks, 1998.

[HKM95] M. Held, J.T. Klosowski, and J.S.B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Canadian Conf. Comput. Geom.*, 1995.

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

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

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

[Hub93] P.M. Hubbard. Interactive collision detection. In *Proc. IEEE Sympos. Research Frontiers in Virtual Reality*, pages 24–31, 1993.

[HZLM01] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha. Fast and simple 2d 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. Technical Report TR02-004, University of North Carolina, 2002.

[JTL$^+$12] C. Je, M. Tang, Y. Lee, M. Lee, and Y. Kim. PolyDepth: Real-time penetration depth computation using iterative contact-space projection. *ACM Trans. Graphics*, 31:5, 2012.

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

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

[KGS97] D. Kim, L.J. Guibas, and S. Shin. Fast collision detection among multiple moving spheres. In *Sympos. Comput. Geom.*, pages 373–375, ACM Press, 1997.

[KHH$^+$09] D. Kim, J.-P. Heo, J. Huh, J. Kim, and S.-E. Yoon. HPCCD: Hybrid parallel continuous collision detection using CPUs and GPUs. *Comp. Graphics Forum (Pacific Graphics)*, 28, 2009.

[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 Comp. Graphics*, 4:21–37, 1998.

[KHY09] D. Kim, J.-P. Heo, and S.-E. Yoon. PCCD: parallel continuous collision detection. In *ACM SIGGRAPH Posters*, page 50, 2009.

[KLM02] Y.J. Kim, M.C. Lin, and D. Manocha. DEEP: an incremental algorithm for penetration depth computation between convex polytopes. In *Proc. IEEE Conf. Robotics Automation*, pages 921–926, 2002.

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

[KOLM02] Y.J. Kim, M.A. Otaduy, M.C. Lin, and D. Manocha. Six-degree-of-freedom haptic display using localized contact computations. *Proc. 10th Sympos. Haptic Interfaces for Virtual Environment and Teleoperator Systems.*, pages 209–216, 2002.

[KOY$^+$11] Y.J. Kim, Y.-T. Oh, S.-H. Yoon, M.-S. Kim, and G. Elber. Coons BHV for freeform geometric models. *ACM Trans. Graph.*, 30:169, 2011.

[KOY$^+$13] Y.J. Kim, Y.-T. Oh, S.-H. Yoon, M.-S. Kim, and G. Elber. Efficient Hausdorff distance computation for freeform geometric models in close proximity. *Computer-Aided Design*, 45:270–276, 2013.

[KPLM98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shell: A higher order bounding volume for fast proximity queries. In *Proc. 3rd Int. Workshop Algorithmic Found. Robotics*, pages 177–190, A.K Peters, Natick, 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*, 24:41–55, 1990.

[KR03] B. Kim and J. Rossignac. Collision prediction for polyhedra under screw motions. In *ACM Conf. Solid Modeling Appl.*, pages 4–10, 2003.

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

[LAM09] T. Larsson and T. Akenine-Möller. Bounding volume hierarchies of slab cut balls. *Comp. Graphics Forum*, 28:2379–2395, 2009.

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

[LGLM99] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, University of North Carolina, 1999.

[LGS$^+$09] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH construction on GPUs. In *Comp. Graphics Forum*, 28:375–384, 2009.

[LHLK10] F. Liu, T. Harada, Y. Lee, and Y.J. Kim. Real-time collision culling of a million bodies on graphics processing units. *ACM Trans. Graph.*, 29:154:1–154:8, 2010.

[Lin93] M.C. Lin. *Efficient Collision Detection for Animation and Robotics.* PhD thesis, University of California, Berkeley, 1993.

[LK10] Y. Lee and Y.J. Kim. Simple and parallel proximity algorithms for general polygonal models. *Comp. Animation Virtual Worlds*, 21:365–374, 2010

[Lla05] B. Llanas. Efficient computation of the hausdorff distance between polytopes by exterior random covering. *Comput. Optim. Appl.*, 30:161–194, 2005.

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

[LMM10] C. Lauterbach, Q. Mo, and D. Manocha. gProximity: hierarchical GPU-based operations for collision and distance queries. *Comp. Graphics Forum*, 29:419–428, 2010.

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

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

[LWH$^+$07] S. Liu, C.C.L. Wang, K.-C. Hui, X. Jin, and H. Zhao. Ellipsoid-tree construction for solid objects. In *Proc. ACM Sympos. Solid and Physical Modeling*, pages 303–308, 2007.

[Man92] D. Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics.* PhD thesis, University of California, Berkeley, 1992.

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

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

[Mir96] B.V. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems.* PhD thesis, University of California, Berkeley, 1996.

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

[MP09] B. Mourrain and J.P. Pavone. Subdivision methods for solving polynomial equations. *Symbolic Comput.*, 44:292–306, 2009.

[MW88] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *ACM SIGGRAPH Comp. Graphics*, 22:289–298, 1988.

[MZ90] M. McKenna and D. Zeltzer. Dynamic simulation of autonomous legged locomotion. *ACM SIGGRAPH Comp. Graphics*, 24:29–38, 1990.

[NAT90] B. Naylor, J. Amanatides, and W. Thibault. Merging BSP trees yield polyhedral modeling results. In *ACM SIGGRAPH Comp. Graphics*, 24:115–124, 1990.

[NPR09] G. Nawratil, H. Pottmann, and B. Ravani. Generalized penetration depth computation based on kinematical geometry. *Comput. Aided Geom. Design*, 26:425–443, 2009.

[OL03] M.A. Otaduy and M.C. Lin. Sensation preserving simplification for haptic rendering. *ACM Trans. Graphics*, 543–553, 2003.

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

[PCM12] J. Pan, S. Chitta, and D. Manocha. Fcl: A general purpose library for collision and proximity queries. In *Proc. IEEE Int. Conf. Robotics Automation*, pages 3859–3866, 2012.

[PEK$^+$11] C.-H. Park, G. Elber, K.-J. Kim, G.-Y. Kim, and J.-K. Seong. A hybrid parallel solver for systems of multivariate polynomials using CPUs and GPUs. *Computer-Aided Design*, 43:1360–1369, 2011.

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

[Pro97] X. Provot. Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface*, pages 177–189, 1997.

[PZM13] J. Pan, X. Zhang, and D. Manocha. Efficient penetration computation using active learning. *ACM Trans. Graphics*, 32:1476–1484, 2013.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. IEEE Int. Conf. Robotics Automation*, 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 Int. Conf. Robotics Automation*, pages 3733–3738, 2000.

[RKC02] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. *Comp. Graphics Forum*, 21:279–287, 2002.

[RKLM04] S. Redon, Y.J. Kim, M.C. Lin, and D. Manocha. Fast continuous collision detection for articulated models. In *Proc. ACM Sympos. Solid Modeling Appl.*, pages 145–156, 2004.

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

[SWF$^+$93] J.M. Snyder, A.R. Woodbury, K. Fleischer, B. Currin, and A.H. Barr. Interval methods for multi-point collisions between time dependent curved surfaces. In *Proc. 20th Conf. Comp. Graphics Interactive Techniques*, pages 321–334, 1993.

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

[SH76] M. Shamos and D. Hoey. Geometric intersection problems. *Proc. 17th IEEE Symp. Found. Comp. Sci.*, pages 208–215, 1976.

[SPO10] Sara C. Schvartzman, Álvaro G. Pérez, and Miguel A. Otaduy. Star-contours for efficient hierarchical self-collision detection. *SIGGRAPH*, 29:3, 2010.

[SSL02] F. Schwarzer, M. Saha, and J.-C. Latombe. Exact collision checking of robot paths. In *Algorithmic Foundations of Robotics V*, vol. 7 of Springer Tracts Advanced Robotics, pages 25–41, 2002.

[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. Numer. Methods Engrg.*, 39:2673–2691, 1996.

[SWF$^+$93] J.M. Snyder, A.R. Woodbury, K. Fleischer, B. Currin, and A.H. Barr. Interval methods for multi-point collisions between time-dependent curved surfaces. In *Proc.*

20th Conf. Computer Ggraphics Interactive Techniques, pages 321–334, ACM Press, 1993.

[TCYM08] M. Tang, S. Curtis, S.-E. Yoon, and D. Manocha. Interactive continuous collision detection between deformable models using connectivity-based culling. In *Proc. ACM Sympos. Solid Physical Modeling*, pages 25–36, 2008.

[TK14] M. Tang and Y.J. Kim. Interactive generalized penetration depth computation for rigid and articulated models using object norm. *ACM Trans. Graph.*, 33:1, 2014.

[TKM09] M. Tang, Y.J. Kim, and D. Manocha. C^2A: Controlled conservative advancement for continuous collision detection of polygonal models. In *Proc. IEEE Int. Conf. Robotics Automation*, pages 849–854, 2009.

[TKM10] M. Tang, Y.J. Kim, and D. Manocha. Continuous collision detection for non-rigid contact computations using local advancement. In *Proc. IEEE Int. Conf. Robotics Automation*, pages 4016–4021, 2010.

[TLK09] M. Tang, M. Lee, and Y.J. Kim. Interactive Hausdorff distance computation for general polygonal models, *ACM Trans. Graphics*, 28:74, 2009.

[TMK14] M. Tang, D. Manocha, and Y.J. Kim. Hierarchical and controlled advancement for continuous collision detectionof rigid and articulated models. *IEEE Trans. Visualization Comp. Graphics*, 20:755–766, 2014.

[TMLT11] M. Tang, D. Manocha, J. Lin, and R. Tong. Collision-streams: Fast GPU-based collision detection for deformable models. In *Proc. ACM SIGGRAPH Sympos Interactive 3D Graphics and Games*, pages 63–70, 2011.

[TMT10a] M. Tang, D. Manocha, and R. Tong. Fast continuous collision detection using deforming non-penetration filters. In *Proc. ACM SIGGRAPH Ssympos. Interactive 3D Graphics and Games*, pages 7–13, 2010.

[TMT10b] M. Tang, D. Manocha, and R Tong. MCCD: Multi-core collision detection between deformable models using front-based decomposition. *Graphical Models*, 72:7–23, 2010.

[TMY$^+$11] Min Tang, Dinesh Manocha, Sung-Eui Yoon, Peng Du, Jae-Pil Heo, and Ruo-Feng Tong. Volccd: Fast continuous collision culling between deforming volume meshes. *ACM Trans. Graphics*, 30:111, 2011.

[TTWM14] M. Tang, R. Tong, Z. Wang, and D. Manocha. Fast and exact continuous collision detection with bernstein sign classification. *ACM Trans. Graphics*, 33:186, 2014.

[TYM08] M. Tang, S. Yoon, and D. Manocha. Interactive continuous collision detection between deformable models using connectivity-based culling. In *Proc. ACM Sympos. Solid and Physical Modeling*, pages 25–36, 2008.

[Wan14] H. Wang. Defending continuous collision detection against errors. *ACM Trans. Graphics*, 33:122, 2014.

[WG91] W. Bouma and G. Vanecek. Collision detection and analysis in a physically based simulation. *Proc. Eurographics Workshop Animation and Simulation*, pages 191–203, 1991.

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

[WTTM15] Z. Wang, M. Tang, R. Tong, and D. Manocha. TightCCD: Efficient and robust continuous collision detection using tight error bounds. In *Comp. Graphics Forum*, 34:289–298, 2015.

[ZK12] X. Zhang and Y.J. Kim. k-IOS: Intersection of spheres for efficient proximity query. In *IEEE Int. Conf. Robotics Automation*, pages 354–359, 2012.

[ZKM08] L. Zhang, Y. Kim, and D. Manocha. A simple path non-existence algorithm using c-obstacle query. In *Algorithmic Foundation of Robotics VII*, vol. 47 of *Springer Tracts Advanced Robotics*, pages 269–284, 2008.

[ZKM07a] L. Zhang, Y. Kim, and D. Manocha. A fast and practical algorithm for generalized penetration depth computation. In *Proc. Robotics: Science and Systems Conf.*, 2007.

[ZKM07b] L. Zhang, Y.J. Kim, and D. Manocha. C-DIST: efficient distance computation for rigid and articulated models in configuration space. In *ACM Sympos. Solid Physical Modeling*, pages 159–169, 2007.

[ZKVM07] L. Zhang, Y.J. Kim, G. Varadhan, and D. Manocha. Generalized penetration depth computation. *Computer-Aided Design*, 39:625–638, 2007.

[ZLK06] X. Zhang, M. Lee, and Y.J. Kim. Interactive continuous collision detection for non-convex polyhedra. *The Visual Computer*, pages 749–760, 2006.

[ZRLK07] X. Zhang, S. Redon, M. Lee, and Y.J. Kim. Continuous collision detection for articulated models using taylor models and temporal culling. *ACM Trans. Graphics*, 26:15, 2007.