Chapter 2
A Brief Overview of Collision Detection

In this chapter we will provide a short overview on classical and recent research in collision detection. In the introduction, we already mentioned the general complexity of the collision detection problem due to its theoretical quadratic running time for polygonal models like Chazelle's polyhedron (see Fig. 1.1).

However, this is an artificial example, and in most real world cases there are only very few colliding polygons. Hence, the goal of collision detection algorithms is to provide an output sensitive running time. This means that they try to eliminate as many of the $O(n^2)$ primitive tests as possible, for example by an early exclusion of large parts of the objects that cannot collide. Consequently, the collision detection problem can be regarded as a filtering process.

Recent physics simulation libraries like PhysX [163], Bullet [36] or ODE [203] implement several levels of filtering in a so-called *collision detection pipeline*.

Usually, a scene does not consist only of a single pair of objects, but of a larger set of 3D models that are typically organized in a scenegraph. In a first filtering step, the *broad phase* or *N-body culling*, a fast test enumerates all pairs of potentially colliding objects (the so-called *potentially collision set* (*PCS*)) to be checked for exact intersection in a second step, which is called the *narrow phase*. The narrow phase is typically divided into two parts: first a filter to achieve pairs of potentially colliding geometric primitives is applied and finally these pairs of primitives are checked for collision. Depending on the scene, more filtering levels between these two major steps can be used to further speed-up the collision detection process [247]. Figure 2.1 shows the design of CollDet [250], a typical collision detection pipeline. All data structures that are developed for this work have been integrated into the CollDet framework.

However, the chronological order of the collision detection pipeline is only one way to classify collision detection algorithms, and there exist many more distinctive factors. Other classifications are e.g. rigid bodies vs. deformable objects. Usually, the filtering steps rely on geometric acceleration data structures that are set up in a pre-processing step. If the objects are deformable, these pre-calculated data structures can become invalid. Consequently, deformable objects require other data structures or, at least, additional steps to update or re-compute the pre-processed struc-

R. Weller, *New Geometric Data Structures for Collision Detection and Haptics*,
Springer Series on Touch and Haptic Systems, DOI 10.1007/978-3-319-01020-5_2,

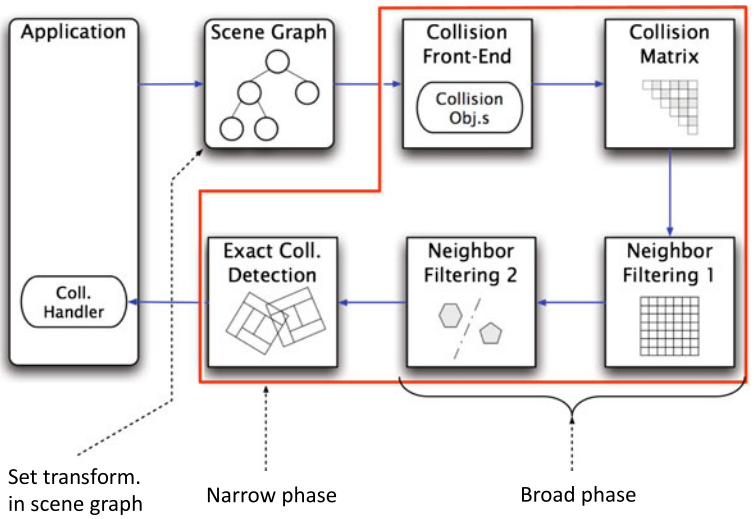

Fig. 2.1 The typical design of a collision detection pipeline

tures. Additionally, deformable objects require a check for self-collisions. Some of these methods are described in Sect. 2.5.

Another distinctive feature is the representation of the geometric objects. Especially in computer graphics, the boundary of objects is usually approximated by polygons. Hence, most collision detection algorithms are designed for polygonal objects. However, in CAD/CAM applications also curved surface representations like *non-uniform rational B-splines* (*NURBS*) play an important role. For instance, Page and Guibault [175] described a method based on oriented bounding boxes (OBBs) especially for NURBS surfaces. Lau et al. [131] developed an approach based on axis aligned bounding boxes (AABBs) for inter-objects as well as self-collision detection between deformable NURBS. Greß et al. [76] also used an AABB hierarchy for trimmed NURBS but transferred the computation to the GPU. Kim et al. [108] proposed an algorithm based on bounding coons patches with offset volumes for NURBS surfaces. Another object modeling technique often used in CAD/CAM is the *constructive solid geometry* (*CSG*). Objects are recursively defined by union, intersection or difference operations of basic shapes like spheres or cylinders. In order to detect collisions between CSG objects, Zeiller [251] used an octree-like data structure for the CSG tree. Su et al. [208] described an adaptive selection strategy of optimal bounding volumes for sub-trees of objects in order to realize a fast localization of possible collision regions.

Point clouds become more and more popular due to cheap depth-cameras that can be used for 3D scanning like Microsoft's Kinect [94]. One of the first approaches to detect collision between point clouds was developed by Klein and Zachmann [116]. They use a bounding volume hierarchy in combination with a sphere covering of parts of the surface. Klein and Zachmann [117] proposed an interpolation search approach of the two implicit functions in a proximity graph in combination with

randomized sampling. El-Far et al. [47] support only collisions between a single point probe and a point cloud. For this, they fill the gaps surrounding the points with AABBs and use an octree for further acceleration. Figueiredo et al. [53] used R-trees, a hierarchical data structure that stores geometric objects with intervals in several dimensions [80], in combination with a grid for the broad phase. Pan et al. [177] described a stochastic traversal of a bounding volume hierarchy. By using machine learning techniques, their approach is also able to handle noisy point clouds. In addition to simple collision tests, they support the computation of minimum distances [178].

This directly leads to the next classification feature: The kind of information that is provided by the collision detection algorithm. Actually, almost all simulation methods work discretely; this means that they check only at discrete points in time whether the simulated objects collide. As a consequence, inter-penetration between simulated objects is often unavoidable. However, in order to simulate a physically plausible world, objects should not pass through each other and objects should move as expected when pushed or pulled. As a result, there exist a number of collision response algorithms to resolve inter-penetrations. For example, the penalty-based method computes non-penetration constraint forces based on the amount of inter-penetration [207]. Other approaches like the impulse-based method or constraint-based algorithms need information about the exact time of contact to apply impulsive forces [110].

Basic collision detection algorithms simply report whether or not two objects intersect. Additionally, some of these approaches provide access to a single pair of intersecting polygons or they yield the set of all intersecting polygons. Unfortunately, this is not sufficient to provide the information required for most collision response schemes. Hence, there also exist methods that are able to compute some kind of *penetration depth*, e.g. a minimum translational vector to separate the objects. More advanced algorithms provide the *penetration volume*. Especially in path-planning tasks, but also in constraint-based simulations, it is helpful to track the *minimum separation distance* between the objects in order to avoid collisions. Finally, the *continuous collision detection* computes the exact point in time when a collision occurs between two object configurations. Section 2.3 provides an overview over algorithms that compute these different penetration measurements. Usually, the more information the collision detection algorithm provide, the longer is its query time.

More classifications of collision detection algorithms are possible. For instance, real-time vs. offline, hierarchical vs. non-hierarchical, convex vs. non-convex, GPU-based methods vs. CPU, etc. This already shows the great variety of different approaches.

Actually, collision detection has been researched for almost three decades. A complete overview over all existing approaches would fill libraries and thus is far beyond the scope of this chapter. So, in the following, we will present classic methods that are still of interest, as well as recent directions that are directly related to our work. As a starting point for further information on the wide field of collision detection we refer the interested reader to the books by Ericson [49], Coutinho [37], Zachmann and Langetepe [249], Eberly [43], Den Bergen [228], Bicchi et al. [18]

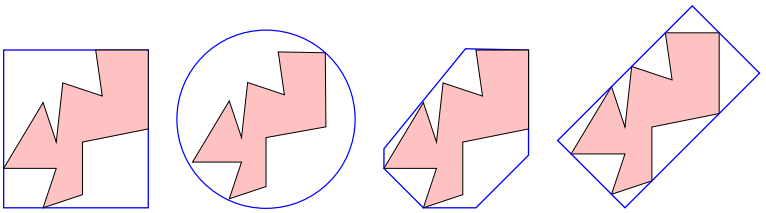

Fig. 2.2 Different bounding volumes

or Lin et al. [141] and the surveys by Jimenez et al. [97], Kobbelt and Botsch [120], Ganjugunte [60], Lin and Gottschalk [140], Avril et al. [8], Kockara et al. [121], Gottschalk [71], Fares and Hamam [51], Teschner et al. [218] and Kamat [103].

2.1 Broad Phase Collision Detection

The first part of the pipeline, called the *broad-phase*, should provide an efficient removal of those pairs of objects that are not in collision. Therefore, objects are usually enclosed into basic shapes that can be tested very quickly for overlap. Typical basic shapes are *axis aligned bounding boxes* (*AABB*), spheres, *discrete oriented polytopes* (*k-DOP*) or *oriented bounding boxes* (*OBB*) (see Fig. 2.2).

The most simple method for the neighbor finding phase is a brute-force approach that compares each object's bounding volume with all others' bounding volumes. The complexity of this approach is $O(n^2)$, where n denotes the number of objects in the scene. Woulfe et al. [241] implemented this brute-force method on a Field-Programmable Gate Array (FPGA) using AABBs. However, even this hardware-based approach cannot override the quadratic complexity.

Moreover, Edelsbrunner and Maurer [45] have shown that the optimal algorithm to find intersections of n AABBs in 3D has a complexity of $O(n \log^2 n + k)$, where k denotes the number of objects that actually intersect. Two main approaches have been proposed to take this into account: spatial partitioning and topological methods.

Spatial partitioning algorithms divide the space into cells. Objects whose bounding volumes share the same cell are selected for the narrow phase. Examples for such spatial partitioning data structures are regular grids [247], hierarchical spatial hash tables [156], octrees [12], kd-trees [17] and binary space partitions (BSP-trees) [162]. The main disadvantage of spatial subdivision schemes for collision detection is their static nature: they have to be rebuilt or updated every time the objects change their configuration. For uniform grids such an update can be performed in constant time and grids are perfectly suited for parallelization. Mazhar [149] presented a GPU implementation for this kind of uniform subdivision. However, the effectiveness of uniform grids disappears if the objects are of widely varying sizes. Luque et al. [147] proposed a semi-adjusting BSP-tree that does not require a complete

re-structuring, but adjusts itself while maintaining desirable balancing and height properties.

In contrast to space partitioning approaches, the topological methods are based on the position of an object in relation to the other objects. The most famous method is called *Sweep-and-Prune* [32]. The main idea is to project the objects' bounding volume on one or more axes (e.g. the three coordinate axis (x, y, z)). Only those pairs of objects whose projected bounding volumes overlap on all axes have to be considered for the narrow phase. Usually, this method does not construct any internal structure but starts from scratch at each collision check.

Several attempts have been proposed to parallelize the classical Sweep-And-Prune approach. For instance, Avril et al. [10] developed an adaptive method that runs on multi-core and multi-threaded architectures [9] and uses all three coordinate axes. Moreover, they presented an automatic workload distribution based on off-line simulations to determine fields of optimal performance [11]. Liu et al. [143] ported the Sweep-and-Prune approach to the GPU using the CUDA framework. They use a principal component analysis to determine a good sweep direction and combine it with an additional spatial subdivision.

Tavares and Comba [217] proposed a topological algorithm that is based on Delaunay triangulations instead of Sweep-and-Prune. The vertices of the triangulation represent the center of mass of the objects and the edges are the object pairs to be checked in the narrow phase.

However, even if all these algorithms are close to the optimal solution proved by Edelsbrunner and Maurer [45], in accordance to Zachmann [247], they are profitable over the brute-force method only in scenarios with more than 100 dynamically simulated objects. This is due to the high constant factor that is hidden in the asymptotic notation. Maybe this is also why much more research is done on the acceleration of the narrow phase.

2.2 Narrow Phase Basics

While the broad phase lists pairs of possible colliding objects, the objective of the narrow phase is to determine exact collision checks between these pairs.

A brute force solution for the narrow phase could simply check all geometric primitives of one object against all primitives of the other object. Surely this would again result in quadratic complexity. Due to the fast evolution of modern graphics hardware, objects can consist of millions of polygons today, and a quadratic running time is not an option. Consequently, more intelligent algorithms are required.

Actually, the narrow phase can be divided into two phases by itself. In a first phase, non-overlapping parts of the objects are culled; in a second step, an accurate collision detection is performed between pairs of geometric primitives that are not culled in the first phase.

Instead of data structures that partition the world-space in the broad phase, in the narrow phase, most often object partitioning techniques are used for the culling

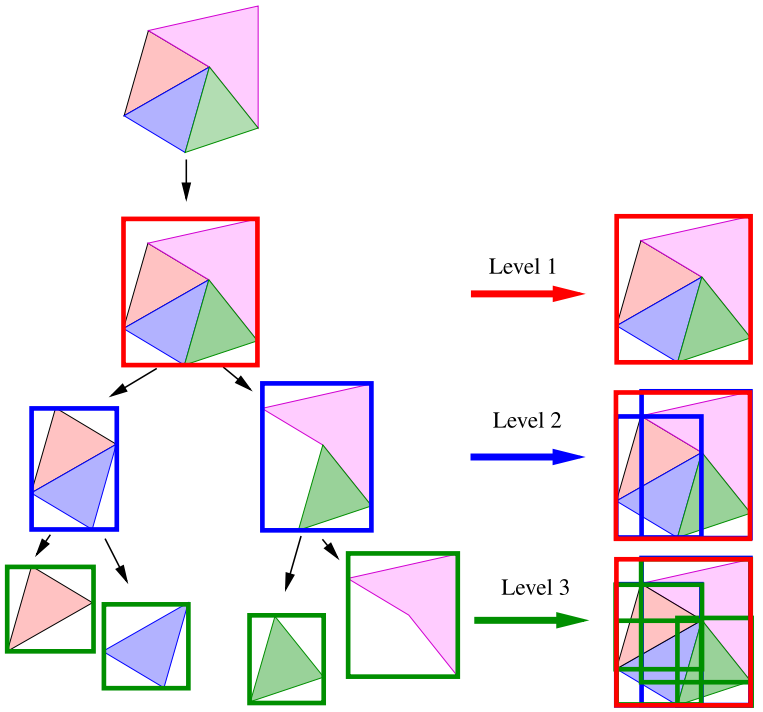

Fig. 2.3 The BVH principle: Geometric objects are divided recursively into subsets of their geometric primitives (*left*) and each node on the tree realizes a bounding volume for all primitives in its sub-tree (*right*)

stage. The common data structures for this task are *bounding volume hierarchies* (*BVHs*). The technique of bounding volumes, known from the previous section (Fig. 2.2), is recursively applied to a whole object. This results in a tree-like structure. Each node in such a tree is associated to a bounding volume that encloses all primitives in its sub-tree (see Fig. 2.3).

Usually, a BVH is constructed in a pre-processing step that can be computationally more or less expensive. During running time a simultaneous recursive traversal of the BVHs of two objects allows a conservative non-intersection pruning: if an intersection is detected in the root of the BVH, the traversal proceeds by checking the bounding volumes of the root node's children and so on until the leaf nodes are reached and an exact collision test between the geometric primitives can be performed. Non-overlapping BVs are discarded from further consideration. The whole traversal algorithm results in a *bounding volume test tree* (*BVTT*) (see Fig. 2.4).

Usually, BVs for the BVHs are spheres [92, 185], AABBs [182, 225] and their memory optimized derivative called BoxTree [248], which is closely related to kd-Trees, k-DOPs [118, 245], a generalization of AABBs, OBBs [2, 15, 70] or convex hull trees [46]. Additionally, a wide variety of special BVs for spe-

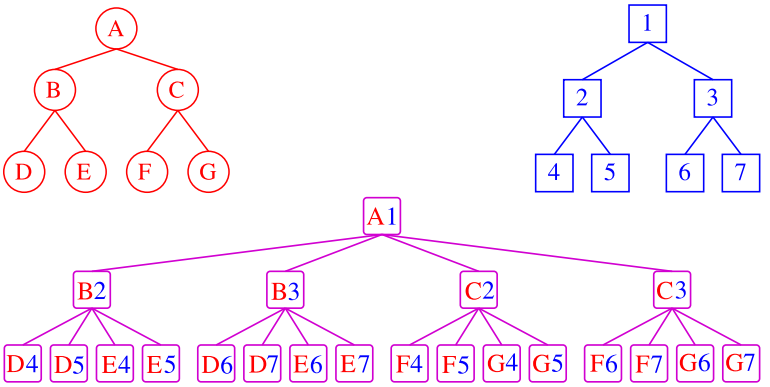

Fig. 2.4 The simultaneous recursive traversal of two BVHs during the collision check results in a bounding volume test tree

cial applications has been developed. For instance, we have spherical shells [125], swept spheres [126], spheres that are cut by two parallel planes called slab cut balls [130], quantized orientation slabs with primary orientations (QuOSPO) trees [85] that combine OBBs with k-DOPs, or combinations of spherical shells with OBBs as proposed by Krishnan et al. [124] for objects that are modeled by Bezier patches.

The optimal bounding volume should

- tightly fit the underlying geometry
- provide fast intersection tests
- be invariant undergoing rigid motion
- not use too much memory
- be able to be build automatically and fast

Unfortunately, these factors are contradictory. For example, spheres offer very fast overlap and distance tests and can be stored very memory efficiently, but they poorly fit flat geometries. AABBs also offer fast intersection tests, but they need to be realigned after rotations. Or, if no realignment is used, a more expensive OBB overlap test is required. But in this case, the tighter fitting OBBs could be used directly. However, they also require more memory. Convex hulls offer the tightest fit among convex BVs, but the overlap test is very complex and their memory consumption depends on the underlying geometry.

Consequently, choosing the right BVHs is always a compromise and depends on the scenario. Basically, the quality of BVH-based algorithms can be measured by the following cost function, which was introduced by Weghorst et al. [235] to analyze hierarchical methods for ray tracing and later was adapted to hierarchical

collision detection methods by Gottschalk et al. [70]:

$$\begin{aligned} T &= N_v C_v + N_p C_p \text{ with} \\ T &= \text{Total cost of testing a pair of models for intersection} \\ N_v &= \text{Number of BV Tests} \\ C_v &= \text{Cost of a BV Test} \\ N_p &= \text{Number of Primitive Tests} \\ C_p &= \text{Cost of a Primitive Test} \end{aligned} \tag{2.1}$$

In addition to the shape of the BV, there are more factors that affect the efficiency of a BVH, including the height of the hierarchy, which may but should not be influenced by its arity or the traversal order during collision queries. The first two factors have to be considered already during the construction of the BVH.

Basically, there exist two major strategies to build BVHs: bottom-up and top-down. The bottom-up approach starts with elementary BVs of leaf nodes and merges them recursively together until the root BV is reached. A very simple merging heuristic is to visit all nearest neighbors and minimize the size of the combined parent nodes in the same level [191]. Less greedy strategies combine BVs by using tilings [137].

However, the most popular method is the top-down approach. The general idea is to start with the complete set of elementary BVs, then split that into some parts and create a BVH for each part recursively. The main problem is to choose a good splitting criterion. A classical splitting criterion is to simply pick the longest axis and split it in the middle of this axis. Another simple heuristic is to split along the median of the elementary bounding boxes along the longest axis. However, it is easy to construct worst case scenarios for these simple heuristics. The *surface area heuristic* (*SAH*) tries to avoid these worst cases by optimizing the surface area and the number of geometric primitives over all possible split plane candidates [68]. Originally developed for ray tracing, it is today also used for collision detection. The computational costs can be reduced to $O(n \log n)$ [232, 233] and there exist parallel algorithms for the fast construction on the GPU [132]. Many other splitting criteria were compared by Zachmann [246].

In addition to the splitting criterion, also the choice of the BV affects the performance of the hierarchy creation process. Even if this is a pre-processing step, extremely high running times are undesirable in many applications. Computing an AABB for a set of polygons or a set of other AABBs is straightforward. Also k-DOPs can be computed relatively easy. But the only optimal solution for OBB computation is $O(n^3)$ and very hard to implement [166]. Chang et al. [24] presented a close to optimal solution based on a hybrid method combining genetic and Nelder-Mead algorithms. Other heuristics, like principal component analysis [100], are not able to guarantee the desired quality in all cases. On the other hand, very complicated BVs, like the convex hull, can be computed efficiently in $O(n \log n)$ [102].

With OBBs, also the computation of a minimum enclosing sphere turns out to be very complicated. Welzl [236] formulated it as a linear programming problem.

However, the choice of spheres as BVs also points to another challenge: the set of elementary BVs. For AABBs, OBBs or k-DOPs, usually a single primitive or a set of adjacent primitives are enclosed in an elementary BV. For spheres this is not an optimal solution, because proximate primitives, often represented by polygons, usually form some kind of flat geometry that poorly fits into a sphere. Therefore, Bradshaw and O'Sullivan [20] presented a method based on the medial axis to group also distant spheres in the same elementary BV.

The influence of the trees' branching factor is widely neglected in the literature. Usually, most authors simply use binary trees for collision detection. But according to Zachmann and Langetepe [249] the optimum can be larger. Mezger et al. [155] stated that, especially for deformable objects, 4-ary or 8-ary trees could improve the performance. This is mainly due to the smaller number of BV updates. However, we will return to this topic in Sect. 2.5.

During running time, the performance of the BVH depends on the traversal order. Usually, a simultaneous recursive traversal of both BVHs is applied. The easiest way to do this is via the *depth-first-search* (*DFS*). Gottschalk [72] additionally proposed a *breath-first-search* (*BFS*) traversal using a queue. For complex objects with many polygons and hence deep trees, the DFS can lead to a stack overflow. However, on modern CPUs with large stack sizes, the DFS is much faster. O'Sullivan and Dingliana [168] proposed a *best-first-search* method for sphere trees. It simply descends into sub-trees with largest BV-overlap first. However, in our experience, the time to keep a priority queue often exceeds its advantages.

The final step in the collision detection pipeline is the primitive test. Most often the surfaces of the objects are represented by polygons or, more specific, triangles. A general polygon–polygon intersection test is described by Chin and Wang [29]. For the special case of triangles, there exist a wide variety of fast intersection tests, e.g. by Möller [159] or Tropp et al. [222]. Even today new optimized approaches are proposed for special cases: for instance Chang and Kim [25] described a triangle test that takes into account that many intermediate computation results from an OBB test can be re-used for the triangle intersection. Many fast intersection tests are implemented by Held [87] and Schneider and Eberly [197].

Another important class of geometric primitives are convex polytopes. Not only because they are widely used in physics-based simulations, but also from an historical point of view: some of the first collision detection algorithms are based on them. Moreover, they can be used as both geometric primitives and bounding volumes. Actually, there exist two main approaches for convex polytopes: feature-based algorithms and simplex-based algorithms.

The first feature-based method was proposed by Lin and Canny [139]. Features of a convex polyhedron are vertices, edges and faces. The Lin–Canny algorithm performs a local search on these features using a pre-computed Voronoi diagram [231]. The convexity guarantees that local minima are avoided. Furthermore, the algorithm uses spatial and temporal coherence between two distinctive queries: usually, objects do not move too much between two frames of a physics-based simulation.

Hence, the closest feature in the current frame is close to the closest feature from the next frame. A major drawback of the algorithm is that it cannot handle intersections. In this case it runs in an endless loop. V-Clip [157], an extension of the classical Linn–Canny method, eliminates this serious defect.

The best known simplex-based algorithm was developed by Gilbert et al. [65]. Instead of using Voronoi diagrams, the GJK-algorithm is based on Minkowski differences. In addition to the boolean collision detection that simply reports whether two objects collide or not, the GJK-algorithm also returns a measure of the inter-penetration [22]. Moreover, it achieves the same almost constant time complexity as Lin–Canny. A stable and fast implementation of the enhanced GJK algorithms was presented by Bergen [226].

Both kinds of algorithms are designed for convex polyhedra. However, by using a convex decomposition of well-behaved concave polyhedrons, they can also be extended to other objects [26]. But finding good convex decompositions is not straightforward and is still an active field of research [81, 138].

2.3 Narrow Phase Advanced: Distances, Penetration Depths and Penetration Volumes

For physics-based simulations a simple boolean answer at discrete points in time to whether a pair of objects intersect or not is often not sufficient. Usually, some kind of contact information is required to compute repelling forces or non-intersection constraints.

As long as a pair of objects rests in a collision-free configuration, a simple way to characterize the extent of repelling forces is to use the *minimum distance* between them. However, collisions are often unavoidable due to the discrete structure of the simulation process. Therefore, a penetration measure is required for configurations where the objects overlap. Some authors proposed a minimum translational vector to separate the objects. This is often called the *penetration depth*. The most complicated, but also the only physically plausible inter-penetration measure is the penetration volume [164], which corresponds directly to the amount of water being displaced by the overlapping parts of the objects. Last but not least, it is possible to compute the exact point in time between two discrete collision checks; this is called *continuous collision detection*. In fact, it is not a measure of the amount of inter-penetration, but the techniques that are used for its computation are very similar to other penetration depth computations.

2.3.1 Distances

The Lin–Canny algorithm, described in the previous section, is already an example of one using minimum distance computations. Tracking of the closest features directly delivers the required distances. Actually, computing minimum distances can

be performed in a very similar way to conventional boolean collision detection using BVHs.

The traditional recursive BVH traversal algorithm, described above, tests whether two BVs—one from each BVH—overlap. If this is the case, the recursion continues to their children. If they do not, the recursion terminates. If two leaves are reached, a primitive intersection test is performed.

The simple recursive scheme can be modified easily for minimum distance computations: just the intersection test of the primitives has to be replaced by a distance computation between the primitives and the intersection test between the BVs by a distance test between the BVs. During the traversal, an upper bound for the distance between two primitives is maintained by a variable δ. This variable can be initialized with ∞ or the distance between any pair of primitives. δ has to be updated if a pair of primitives with a smaller distance is found.

Obviously, BVs with larger distances than δ can be culled, because if the BVs have a larger distance, this must also be true for all enclosed primitives. This is exactly the way most authors using BVHs implemented their algorithms; e.g. Larsen et al. [126] used the swept-sphere method as BVs together with several speed-up techniques, Quinlan [185] proposed sphere trees, Bergen [226] used AABBs in combination with the GJK-based Minkowski difference; Lauterbach et al. [133] implemented OBB trees running on the GPU. Johnson and Cohen [98] generalized the basic BVH-based distance computation in the framework of minimum distance computations.

Actually, all these approaches can be interrupted at any time and they deliver an upper bound for the minimum distance. Other approaches are able to additionally provide a lower bound, like the spherical sector representation presented by Bonner and Kelley [19], or the inner–outer ellipsoids by Ju et al. [101] and Liu et al. [144].

Another alternative for distance computations are distance fields [56], which can also be combined with BVHs [58].

However, all these approaches use the Euclidean distance between the objects. Other authors also proposed different metrics like the Hausdorff-distance, which defines the maximum deviation of one object from the other object [213, 243]. Zhang et al. [256] used a so-called *DISP distance*, which is defined as the maximum length of the displacement vector over every point on the model at two different configurations. This metric can be used for motion planning tasks [134].

A *local* minimum distance for a stable force feedback computation was proposed by Johnson et al. [99]. They used spatialized normal cone pruning for the collision detection. The normal cone approach differs from prior works using BVHs, because it searched for extrema of a minimum distance formulation in the space of normals rather than in Euclidean space.

2.3.2 Continuous Collision Detection

Computing repelling forces on the separating distance can lead to visual artifacts in physics-based simulations, e.g. when the objects bounce away before they really are

in visual contact. Moreover, if the objects move too fast, or the time step between two collision queries is too large, the objects could pass through each other. To avoid errors like this tunneling effect, it would be better to really compute the exact time of impact between a pair of objects [35]. Several techniques have been proposed to solve this *continuous collision detection* problem, which is sometimes also called *dynamic collision detection*.

The easiest way is to simply reuse the well researched and stable algorithms known from static collision detection. Visual interactive applications usually require updating rates of 30 frames per second, i.e. there passes about 30 milliseconds of time between two static collision checks. Recent boolean collision detection algorithms require only a few milliseconds, depending on the objects' configuration. Hence, there is plenty of time to perform more than one query between two frames. A simple method, the so called method of *pseudo-continuous collision*, realizes exactly this strategy: it performs static collision detection with smaller time steps [88]. Even with a higher sampling frequency, it is, however, still possible to miss contacts between thin objects.

Conservative advancement is another simple technique that avoids these problems. The objects are repeatedly advanced by a certain time-step, which guarantees a non-penetration constraint [158]. Usually, the minimum distance is used to compute iteratively new upper bounds for the advancement [259]. Conservative advancement is also perceived as a discrete ancestor of the kinetic data structures that we will review in the next chapter.

Another method is to simply enclose the bounding volumes at the beginning and at the end of a motion step by a swept volume. This can be done very efficiently for AABBs [44]. Coming and Staadt [34] described a velocity-aligned DOP as swept volume for underlying spheres as BVs, and Redon et al. [189] proposed an algorithm for OBBs. Taeubig and Frese [210] used sphere swept convex hulls. Also ellipsoids are an option [30].

The swept volumes guarantee conservative bounds for their underlying primitives, and consequently the swept BVHs can be traversed similarly to the discrete BVHs. However, an additional continuous collision test for the primitives is required to achieve the exact time of impact. Actually, these tests (and in fact, also the tests between the BVs) depend on the trajectories of the primitives, which are usually not known between two simulation steps. Often, a simple linear interpolation is used to approximate the in-between motion [239]. For a pair of triangles this yields six face–vertex and nine edge–edge tests. Each of these elementary tests requires one to solve a cubic equation. This is computationally relatively costly. Therefore, some authors additionally proposed feature-based pre-tests, like the subspace filters by Tang et al. [211] or additional BVs like k-DOPs for the edges [93].

However, more accurate but also more complicated interpolation schemes have been described as well. Canny [23] proposed quaternions instead of Euler angles but still got a 6D complexity. Screw motions are often used [105] because they can also be computed by solving cubic polynomials. Redon et al. [187] combined them with interval arithmetic. Zhang et al. [260] defined Taylor models for articulated models with non-convex links. Von Herzen et al. [230] used Lipschitz bounds and binary subdivision for parametric surfaces.

There exist a few other acceleration techniques; e.g. Kim et al. [106] implement a dynamic task assignment for multi-threaded platforms, or Fahn and Wang [50] avoid BVHs by using a regular grid in combination with an azimuth elevation map. However, continuous collision detection is still computationally too expensive for real-time applications, especially, when many complex dynamic objects are simulated simultaneously.

2.3.3 Penetration Depth

The minimum distance is not a good measure to define repelling forces, and computing the exact time of impact using continuous collision detection is too time consuming for real-time applications. Consequently, in research one has developed another penetration measure: the *penetration depth*. In fact, it is not entirely correct to speak about *the* penetration depth, because there exist many different, partly contradictory, definitions. A widely used definition describes it as the distance that corresponds to the shortest translation required to separate two intersecting objects [41].

The same authors also delivered a method for their computation based on the Dobkin and Kirkpatrick hierarchy and Minkowski differences. They derived a complexity of $O(n^2)$ for convex and $O(n^4)$ for non-convex polyhedral objects consisting of n polygons. Cameron [22] presented a similar approach for convex objects, which can additionally track the minimum distance in non-intersection cases. Especially the computation of the Minkowski difference is very time consuming and difficult. Therefore, several approximation schemes have been developed: for instance Bergen [227] described an expanding polytope algorithm that yields a polyhedral approximation of the Minkowski difference. Agarwal et al. [1] proposed an approximation algorithm based on ray-shooting for convex polyhedra. Kim et al. [109] implicitly constructed the Minkowski difference by local dual mapping on the Gaussian map. Additionally, the authors enhanced their algorithm by using heuristics to reduce the number of features [111, 113]. Other approximations rely on discretized objects and distance fields [54].

Some authors computed *local* approximations of the penetration depth if the objects intersect in multiple disjoint zones. Therefore, penetrating zones were partitioned into coherent regions and a local penetration depth was computed for each of these regions separately. Redon and Lin [188] computed a local penetration direction for these regions and then used this information to estimate a local penetration depth on the GPU. Je et al. [96] presented a method based on their continuous collision detection algorithm using conservative advancement [212]: they constructed a linear convex cone around the collision free configuration found via CCD and then formulated a projection of the colliding configuration onto this cone as a linear complementarity problem iteratively.

Also other metrics have been proposed for the characterization of penetrating objects: for instance, Zhang et al. [255] presented an extended definition of the pen-

etration depth that also takes the rotational component into account, called the *generalized penetration depth*. It differs from the translational penetration depth only in non-convex cases, and the computation of an upper bound can be reduced to the convex containment problem if at least one object is convex [257]. Gilbert and Ong [66] defined a *growth distance* that unifies the penetration measure for intersecting but also disjoint convex objects: basically, it measures how much the objects must be grown so that they were just in contact. Also an algorithm for the computation of the growth distance was presented [165]. Zhu et al. [261] used a gauge function [90] instead of the Euclidean norm to define pseudo-distances for overlapping objects and they presented a constrained optimization-based algorithm for its calculation.

The publication years presented in this subsection already show that penetration depth computation has recently become a very active field of research. This is mainly because computing the penetration depth is still computationally very expensive and becomes practically relevant only on very fast machines. However, using the classical penetration depth still has another serious drawback: the translational vector is not continuous at points lying on the medial axis. This results in flipping directions of the contact normals when used directly as penalty force vector. Moreover, it is not straightforward to model multiple simultaneous contacts. Tang et al. [214] tried to avoid these problems by accumulating penalty forces along the penetration time intervals between the overlapping feature pairs using a linear CCD approach.

2.3.4 Penetration Volume

Compared to other penetration measures, the literature on penetration volume computation is sparse. More precisely, there exist only two other algorithms apart from our approach: one method, proposed by Hasegawa and Sato [84], constructs the intersection volume of convex polyhedra explicitly. For this reason, it is applicable only to very simple geometries, like cubes, at interactive rates.

The other algorithm was developed by Faure et al. [52] simultaneously with our *Inner Sphere Trees*. They compute an approximation of the intersection volume from layered depth images on the GPU. This approach is applicable to deformable geometries but restricted to image space precision. And apart from that, it is relatively slow and it cannot provide continuous forces and torques for collision response.

2.4 Time Critical Collision Detection

Despite the computational power available, the performance of collision detection algorithms is still critical in many applications, especially if a required time budget must never be exceeded. This problem arises in almost all interactive real-time applications where frame rates of at least 30 fps are needed for a smooth visual

feedback. Consequently, only 30 msec remain for rendering and physics-based simulation. For the rendering step, there exists the technique of levels-of-details (LOD) to reduce the workload of the graphics pipeline [146]. The main idea is to store geometric data in several decreasing resolutions and choose the right LOD for rendering according to the distance from the viewpoint. Similar techniques can also be applied to the physics-based simulation; more precisely, to the collision detection step. Hence, this so-called *time-critical collision detection* reduces the computation time at the cost of accuracy.

Typically, time-critical collision detection methods rely on simplifications of the complex objects like the visual LOD representations. This can be done either explicitly or implicitly. Moreover, they often use frame-to-frame coherence because in physics-based simulations there should usually be no discontinuities, and hence the contact information between two collision checks does not differ too much.

For instance, the BVTT derived from a simultaneous BVH traversal (see Fig. 2.4 in the previous section) holds in each node the result of the query between two BVs. Those BV pairs where the traversal stops build a list in the BVTT, the *separation list* [27]. In case of high coherence, the traversal does not have to be restarted at the roots of the BVHs for each query, but this list can be directly re-used. Ehmann and Lin [46] called this the generalized front tracking. Lin and Li [142] enhanced this method by defining an incremental algorithm that prioritizes the visiting order: dangerous regions where collisions may occur with a high probability are prioritized.

These are, however, just examples for coherence. In fact, the classical simultaneous BVH traversal lends itself well to time-critical collision detection: the traversal can simply be interrupted when the time budget is exhausted. This was first proposed by Hubbard [92], who additionally used a round-robin order for the collision checks. This approach was later extended by O'Sullivan and Dingliana [168, 169] and Dingliana and O'Sullivan [39]: like Hubbard [92] they also used an interruptible sphere tree traversal but added a more appropriate collision response solution to Hubbard's elementary response model. A similar method can also be adopted for deformable objects [154]. Another extension using sphere trees with a closest feature map to avoid over-estimations of the contact information was presented by Giang and O'Sullivan [62, 63].

Klein and Zachmann [115] described an average case approach for time-critical traversals (ADB-trees): for each pair of BVs they computed the probability that an intersection of the underlying primitives will occur. Coming and Staadt [33] presented an event-based time-critical collision detection scheme relying on stride-scheduling in combination with kinetic Sweep-and-Prune and an interruptible GJK version.

Other authors created the LOD explicitly. For example, Otaduy and Lin [171] presented a dual hierarchy for both the multi-resolution representation of the geometry and its BVH using convex hulls. A similar approach, called clustered hierarchy of progressive meshes, was developed by Yoon et al. [243] for very large scenes that require out-of-core techniques. James and Pai [95] used the reduced models not only for fast collision detection, but also presented a deformation method based on their bounded deformation trees.

2.4.1 Collision Detection in Haptic Environments

Almost all collision detection approaches described above are primarily designed to work in at least visual real-time. As mentioned in the introduction, for a smooth visual sensation update-rates of 30 Hz are sufficient, whereas haptic rendering requires an update frequency of 1000 Hz for a realistic haptic sensation. Moreover, detailed contact information has to be provided for a realistic perception.

None of the previously described methods, especially those computing penetration depths or times of impact, can be accelerated by a factor of 30 out of the box for reasonable scene complexities in haptic environments. Consequently, collision detection for haptics often leads to further simplifications in order to guarantee the high frequency, but also to compute plausible forces.

2.4.1.1 3 DOF

In the early times of haptic human–computer history, the beginning 1990s [195], a major simplification affected both the design of haptic hardware interfaces and the collision detection: instead of simulating the complex interaction of rigid bodies, only a single point probe was used for the interaction. This required only the computation of three force components at the probe's tip. As a result, many 3 DOF haptic devices, like the SensAble Phantom Omni Massie and Salisbury [148], entered the market and also a lot of research was done on 3 DOF haptic rendering algorithms.

One of the first algorithms for this problem was presented by Zilles and Salisbury [262]. They proposed the usage of a two different points: one represents the real position of the probe's tip, whereas the second, they call it *god object*, is constrained to the surface of the polygonal object. A spring–damper model between these points defines the force. Ruspini et al. [194] extended this approach by sweeping a sphere instead of using a single point in order to avoid the god object slipping into a virtual object through small gaps. Ho et al. [89] also took the movement of the god object into account by using a line between its previous and its recent position. BVHs can be used for accelerating the collision detection. For example, Gregory et al. [75] developed a hybrid hierarchical representation consisting of uniform grids and OBBs.

Also algorithms for other than polygonal object representations have been proposed: Thompson et al. [220] developed an algorithm that is applicable for 3 DOF rendering of NURBS surfaces without the use of any intermediate representation. Gibson [64] and Avila and Sobierajski [7] described approaches for volumetric representations. More recent works also included the GPU for faster collision detection using local occupancy maps [107].

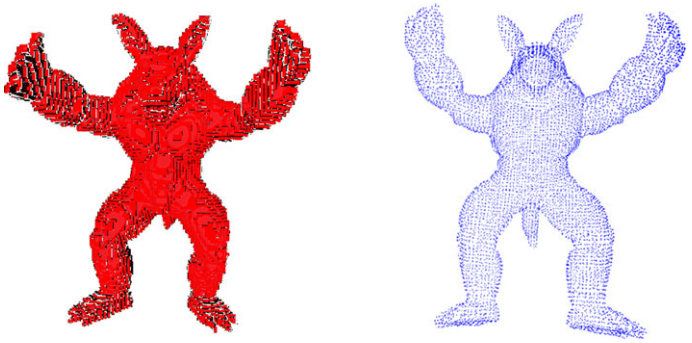

Fig. 2.5 The Voxmap-Pointshell approach for 6 DOF haptic rendering uses two different data structures: A voxelization (*left*) and a point-sampling of the objects' surface (*right*)

2.4.1.2 6 DOF

Many applications, like training or virtual prototyping, require interaction with complex virtual tools instead of just a single point probe to ensure a sufficient degree of realism. As soon as the haptic probe includes 3D objects, the additional rendering of torques becomes important. Also, simultaneous multiple contacts with the environment may occur. This significantly increases the complexity of the collision detection but also of the collision response. Generally, a complete 6 DOF rigid-body simulation, including forces and torques, has to be performed in only 1 millisecond.

For very simple objects, consisting of only a few hundred polygons, the traditional collision approaches described above can be used. Ortega et al. [167] extended the god-object method to 6 DOF haptic rendering using continuous collision detection to derive the position and orientation of the god object. However, they cannot guarantee to meet the time budget; therefore they use asynchronous update processes. Kolesnikov and Zefran [123] presented an analytical approximation of the penetration depth with additional considerations of the rotational motion.

Despite simplifications of temporal constraints, most often geometric simplifications were used. Many 6 DOF haptic rendering approaches are based on the Voxmap Pointshell (VPS) method [151]. The main idea is to divide the virtual environment into a dynamic object that is allowed to move freely through the virtual space and static objects that are fixed in the world. The static environment is discretized into a set of voxels, whereas the dynamic object is described by a set of points that represents its surface (see Fig. 2.5). During query time, for each of these points it is determined with a simple boolean test, whether it is located in a filled volume element or not. Today, voxelization can be efficiently computed using the GPU [42, 179, 198].

Many extensions for the classical VPS algorithms have been proposed: for instance, the use of distance fields instead of simple boolean voxmaps [152] or an additional voxel hierarchy for the use of temporal coherence [153], since also recent computer hardware can perform only a few thousands intersection tests in 1 millisecond. Prior and Haines [183] described a proximity agent method to reduce

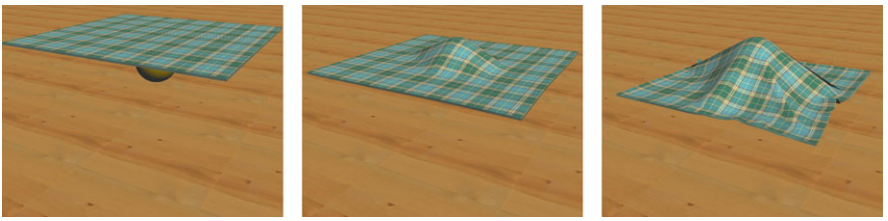

Fig. 2.6 Deformable objects like cloth require special algorithms, because pre-computed data structures become invalid after the deformation. Moreover, collision between parts of the object itself may occur

the number of collision tests for multiple object pairs in collaborative virtual environments. Renz et al. [190] presented extensions to the classic VPS, including optimizations to force calculation in order to increase its stability. Barbič and James [13] developed a distance-field-based approach that can handle contacts between rigid objects and reduced deformable models at haptic rates. Later they extended their approach to cover also deformable versus deformable contacts [14]. Ruffaldi et al. [193] described an implicit sphere tree based on an octree that represents the volumetric data. However, even these optimizations cannot completely avoid the limits of VPS, namely aliasing effects and huge memory consumption.

Other authors use level-of-detail techniques to simplify the complexity of large polygonal models [145]. Otaduy and Lin [172] presented a sensation preserving simplification algorithm and a collision detection framework that adaptively selects a LOD. Later, they added a linearized contact model using contact clustering [170]. Another idea is to combine low-resolution geometric objects along with texture images that encode the surface details [173]. Kim et al. [112] also clustered contacts based on their spatial proximity to speed up a local penetration depth estimation using an incremental algorithm. Johnson et al. [99] approximated the penetration depth by extending their normal cone approach. Glondu et al. [67] developed a method for very large environments using a neighborhood graph: for objects that are closer to the haptic probe they used the LOD.

2.5 Collision Detection for Deformable Objects

Usually, collision detection algorithms rely on pre-computed data structures like BVHs. This works fine, as long as the geometry of the objects does not change, i.e. if the objects are rigid. However, our world consists not only of rigid objects but includes a lot of deformable objects, like cloth (see Fig. 2.6). Consequently, a realistic simulation should also be able to handle deformable models. Beside cloth simulation, popular deformable applications include character animation, surgery simulation, and fractures.

An additional challenge for collision detection of deformable objects is the possibility that parts of one object intersect other parts of the same object, the so-called

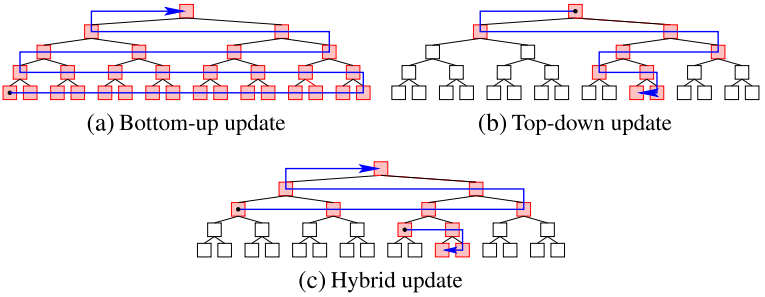

Fig. 2.7 Different updating strategies for BVHs

self-collisions. Actually, BVHs can easily be employed to find self-collisions by simply checking the BVH of an object against itself and rejecting collisions between adjacent primitives [229]. Additionally, techniques like hierarchies of normal cones [184] or power diagrams [77] can be used for further acceleration.

Since BVHs have proven to be very efficient for rigid objects, and, moreover, they can easily be extended to self-collision detection, researchers also want to use them for deformable objects. As the BVHs become invalid after deformations, several approaches have been published to handle this problem: the easiest method is to rebuild the BVH from scratch after each deformation. Unfortunately, it turns out that a complete rebuild is computationally too expensive. Even modern GPU acceleration cannot guarantee real-time performance for BVH construction in reasonably complex scenes [132]. Some authors reduced the rebuild to interesting regions. For example, Smith et al. [202] used a lazy reconstruction of an octree for all primitives in the overlap region, or they keep a more complex data structure like a full octree and simply reinsert all primitives in the leaves in each frame [61]. Other approaches completely avoid hierarchies but used regular spatial subdivision data structures like uniform grids [224, 252]. Spatial hashing helps to reduce the high memory requirements of uniform grids [219]. However, choosing the right grid size remains an unsolved problem due to the inherent "teapot in a stadium" problem [82].

Another method is to avoid the complete rebuild by simply updating the BVs of a pre-computed BVH after deformations. Bergen [225] stated that updating is about ten times faster compared to a complete rebuild of an AABB hierarchy, and as long as the topology of the object is conserved, there is no significant performance loss in the collision check compared to rebuilding. Basically, there exist two main techniques for updating a BVH: bottom-up and top-down. Bottom-up updates start by refitting the BVs of the primitives and merge them upwards with the root of the tree. This can be done efficiently for AABB trees [229] and sphere trees [21]. However, during a collision query usually not all of these BVs are visited. Hence a lot of work may be done on updates that are not required. A simple strategy to reduce the number of updated BVs is to update them on-line, when they are in fact visited during a traversal. This requires the traversal of all primitives placed under a BV. This is the typical top-down approach [127]. Of course, this raises the question: Which of the two methods is better?

Basically, the performance of deformable collision detection algorithms can be derived by a simple extension of the cost function for rigid objects (see Eq. (2.1)):

$$\begin{aligned} T &= N_v C_v + N_p C_p + N_u C_u \quad \text{with} \\ T &= \text{Total cost of testing a pair of models for intersection} \\ N_v &= \text{Number of BV Tests} \\ C_v &= \text{Cost of a BV Test} \\ N_p &= \text{Number of Primitive Tests} \\ C_p &= \text{Cost of a Primitive Test} \\ N_u &= \text{Number of BV Updates} \\ C_u &= \text{Cost of a BV Update} \end{aligned} \tag{2.2}$$

Usually, N_u is higher for the bottom-up update than for the top-down approach. On the other hand, C_u is higher for the top-down method. Consequently, there is no definite answer to the question. Actually, according to Larsson and Akenine-Möller [127], if many deep nodes in a tree are reached, it gives a better overall performance to update the AABBs in a tree bottom-up. In simple cases, however, with only a few deep nodes visited in a collision test, the top-down update performs better. As a compromise, the authors proposed a hybrid updating strategy: for a tree with depth n, initially the first $\frac{n}{2}$ should be updated bottom-up. The lower nodes should be updated top-down on the fly during collision traversal (see Fig. 2.7). Mezger et al. [155] accelerated the update by omitting the update process for several time steps. Therefore, the BVs are inflated by a certain distance, and as long as the enclosed polygon does not move farther than this distance, the BV does not need to be updated.

If specific information about the underlying deformation scheme or the geometric objects is available, additional updating techniques can be used for further acceleration. For instance, Larsson and Akenine-Möller [128] proposed a method for morphing objects, where the objects are constructed by interpolation between some morphing targets: one BVH is constructed for each of the morph targets so that the corresponding nodes contain exactly the same vertices. During running time, the current BVH can be constructed by interpolating the BVs. Spillmann et al. [206] presented a fast sphere tree update for meshless objects undergoing geometric deformations that also supports level-of-detail collision detection. Lau et al. [131] described a collision detection framework for deformable NURBS surfaces using AABB hierarchies. They reduce the number of updates by searching for special deformation regions. Guibas et al. [77] used cascade verification in a sphere tree for deformable necklaces. Sobottka et al. [205] extended this approach to hair simulation using AABBs and k-DOPs [204].

Refitting BVHs works as long as the objects do not deform too much, that is, when the accumulated overlap of the refitted BVs is not too large. This problem arises for example in simulations of fracturing objects. In this case, a complete or

partial rebuild of the BVH may increase the running time significantly. Larsson and Akenine-Möller [129] proposed an algorithm that can handle highly dynamic breakable objects efficiently: they start a refitting bottom-up update at the BVs in the separation list and use a simple volume heuristic to detect degenerated sub-trees that must be completely rebuilt. Otaduy et al. [174] used a dynamic re-structuring of a balanced AVL-AABB tree. Tang et al. [215] described a two-level BVH for breakable objects based on mesh connectivity and bounds on the primitives' normals.

2.5.1 Excursus: GPU-Based Methods

Popular methods for real-time simulation of deformable objects like mass–spring systems [136, 160], but also multi-body simulations [48, 216], can be easily parallelized. Consequently, they are perfectly suited for modern GPU architectures. Hence, it is obvious to develop also collision detection schemes that work directly on the graphics hardware instead of copying data back and forth between main memory and GPU memory.

Actually, GPU-based algorithms have been proposed for all parts of the collision detection pipeline: the broad-phase Le Grand [135], Liu et al. [143], the narrow-phase Chen et al. [28], Greß et al. [76] and even for the primitive tests [73, 240].

The first approaches relied on the fixed-function graphics pipeline of at least OpenGL 1.6 and used image space techniques. For instance, Knott and Pai [119] implemented a ray-casting algorithm based on frame buffer operations to detect static interferences between polyhedral objects. Heidelberger et al. [86] described an algorithm for computation of layered depth images using depth and stencil buffers.

Later, the fixed function pipelines had been replaced by programmable vertex and fragment processors. This also changed the GPU collision detection algorithms: for example, Zhang and Kim [258] performed massively parallel pairwise intersection tests of AABBs in a fragment shader. Kolb et al. [122] used shaders for the simulation of large particle systems, including collisions between the particles.

Today, GPU processors are freely programmable via APIs such as OpenCL or CUDA. This further improves the flexibility of GPU-based collision detection algorithms, like the approach by Pan and Manocha [176] that uses clustering and collision-packet traversal or the method based on linear complementary programming for convex objects by Kipfer [114].

Moreover, several special hardware designs to accelerate collision detection were developed [6, 186]. With the Ageia PhysX card [38], one saw a special hardware card even managing to enter the market. But due to increasing performance and flexibility of GPUs it seems that special physics processing hardware has become obsolete.

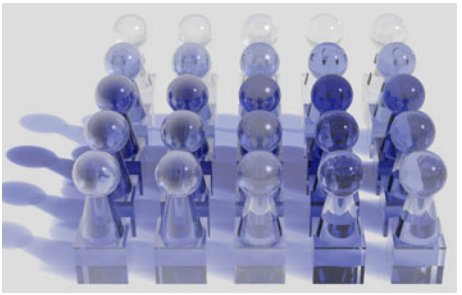

Fig. 2.8 Ray tracing supports a wide variety of optical effects like reflections, refractions, and shadows

2.6 Related Fields

Of course, data structures for the acceleration of geometric queries are not restricted to collision detection. They are also widely used in ray tracing (see Sect. 2.6.1), object recognition [199], 3D audio rendering [223, 234] or occlusion [242, 254], view frustum [31] and backface culling [253]. Moreover, they accelerate visibility queries including hierarchical z-Buffers [74] and back-to-front [55] or front-to-back [69] rendering via BSP-Trees. Geometric hierarchies help to index [79, 201] and search [180] geometric databases efficiently, and they improve hardware tessellation [161].

This small selection of very different applications and the large number of data structures already presented just for the field of collision detection in the previous sections suggests that there is available an almost uncountable number of different approaches. A perfect geometric data structure would be one that can process every imaginable geometric search query optimally. Unfortunately, such a data structure does not—and maybe cannot—exist. Quite to the contrary, much research is concerned with finding optimal data structures for each small sub-problem. However, maintaining dozens of different optimized data structures in a simple virtual environment with ray tracing, sound rendering and collision detection could also be very inefficient due to memory waste and the computational cost of hierarchy updates. Consequently, there is also a counter movement that proposes the use of more general data structures [78].

2.6.1 Excursus: Ray Tracing

Basically, ray tracing is a rendering technique that realizes global illumination for perfect reflections (see Fig. 2.8). Instead of scan converting all polygons in the scene, as traditional renderers like OpenGL and DirectX do, a ray of light is traced backward from the eye through the scene. If the ray hits an object, an additional ray is shot to the light sources and moreover, reflected and refracted rays are further traced recursively [238]. Consequently, the main challenge on tracing rays is to find intersections between these rays and the scene. This problem is closely related to

collision detection where two objects are checked for intersection. Therefore, also the geometric acceleration data structures are very similar.

A complete overview of all existing data structures for ray tracing is far beyond the scope of this excursus. As a starting point we would like to refer the interested reader to the books and surveys of Hanrahan [83], Arvo and Kirk [5], Shirley and Morley [200], and Suffern [209]. In the following, we will briefly point out similarities and differences between ray tracing and collision detection and dwell on the open challenges.

Almost all data structures that were proposed for collision detection had been earlier applied to ray tracing. This includes non-hierarchical data structures like uniform grids [3, 57], as well as bounding volume hierarchies [104, 192]. However, a ray has to be tested for intersection with the whole scene, whereas during the collision detection process objects are checked for collision with other objects in the same scene. Therefore, the data structures for ray tracing are usually used at a *scene* level, while collision detection uses them on an *object* level. Consequently, other spatial subdivision data structures that are rarely used in collision detection, like octrees [196, 237] and kd-trees [59], which were originally developed for associative searches [16], became more popular for ray tracing [233].

However, these data structures are primarily designed for static scenes. If objects in the scene move or deform, the data structures have to be updated or rebuilt. As in collision detection for deformable objects, it is still a challenge to find the right updating strategy and a lot of recent work has been done on this problem recently [4, 244]. Moreover, even when using fast acceleration data structures, ray tracing is computational very expensive and is not applicable for real-time rendering on consumer hardware. However, the first GPU implementations that support parallel tracing of rays seem to be very promising [40, 91, 150, 181, 221].

References

1. Agarwal, P. K., Guibas, L. J., Har-Peled, S., Rabinovitch, A., & Sharir, M. (2000). Penetration depth of two convex polytopes in 3d. *Nordic Journal of Computing*, *7*(3), 227–240. URL http://dl.acm.org/citation.cfm?id=642992.642999.
2. Albocher, D., Sarel, U., Choi, Y.-K., Elber, G., & Wang, W. (2006). Efficient continuous collision detection for bounding boxes under rational motion. In *ICRA* (pp. 3017–3022). New York: IEEE. URL http://dblp.uni-trier.de/db/conf/icra/icra2006.html.
3. Amanatides, J., & Woo, A. (1987). A fast voxel traversal algorithm for ray tracing. In *Eurographics 1987* (pp. 3–10).
4. Andrysco, N., & Tricoche, X. (2011). Implicit and dynamic trees for high performance rendering. In *Proceedings of graphics interface 2011, GI '11*, School of Computer Science, University of Waterloo, Waterloo, Ontario (pp. 143–150). Waterloo: Canadian Human-Computer Communications Society. ISBN 978-1-4503-0693-5. URL http://dl.acm.org/citation.cfm?id=1992917.1992941.
5. Arvo, J., & Kirk, D. (1989). A survey of ray tracing acceleration techniques. In A. S. Glassner (Ed.), *An introduction to ray tracing* (pp. 201–262). London: Academic Press Ltd. ISBN 0-12-286160-4. URL http://dl.acm.org/citation.cfm?id=94788.94794.

6. Atay, N., Lockwood, J. W., & Bayazit, B. (2005). A collision detection chip on reconfigurable hardware (Technical report). In *Proceedings of pacific conference on computer graphics and applications (pacific graphics)*.
7. Avila, R. S., & Sobierajski, L. M. (1996). A haptic interaction method for volume visualization. In *Proceedings of the 7th conference on visualization '96, VIS '96* (pp. 197-ff). Los Alamitos: IEEE Computer Society Press. ISBN 0-89791-864-9. URL http://dl.acm.org/citation.cfm?id=244979.245054.
8. Avril, Q., Gouranton, V., & Arnaldi, B. (2009). New trends in collision detection performance. In S. Richir & A. Shirai (Eds.), *Laval virtual VRIC'09 proceedings*, BP 0119, 53001 Laval Cedex, France, April 2009 (pp. 53–62).
9. Avril, Q., Gouranton, V., & Arnaldi, B. (2010). A broad phase collision detection algorithm adapted to multi-cores architectures. In S. Richir & A. Shirai (Eds.), *VRIC'10 proceedings*, April 2010.
10. Avril, Q., Gouranton, V., & Arnaldi, B. (2010). Synchronization-free parallel collision detection pipeline. In *ICAT 2010*, December 2010.
11. Avril, Q., Gouranton, V., & Arnaldi, B. (2011). Dynamic adaptation of broad phase collision detection algorithms. In *IEEE international symposium on virtual reality innovations*, March 2011.
12. Bandi, S., & Thalmann, D. (1995). An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies. *Computer Graphics Forum*, *14*(3), 259–270. URL http://dblp.uni-trier.de/db/journals/cgf/cgf14.html#BandiT95.
13. Barbič, J., & James, D. L. (2007). Time-critical distributed contact for 6-dof haptic rendering of adaptively sampled reduced deformable models. In *2007 ACM SIGGRAPH / eurographics symposium on computer animation*, August 2007.
14. Barbič, J., & James, D. L. (2008). Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transactions on Haptics*, *1*(1), 39–52.
15. Barequet, G., Chazelle, B., Guibas, L. J., Mitchell, J. S. B., & Tal, A. (1996). Boxtree: a hierarchical representation for surfaces in 3d. *Computer Graphics Forum*, *15*(3), 387–396. URL http://dblp.uni-trier.de/db/journals/cgf/cgf15.html#BarequetCGMT96.
16. Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517. doi:10.1145/361002.361007. URL http://doi.acm.org/10.1145/361002.361007.
17. Bentley, J. L., & Friedman, J. H. (1979). Data structures for range searching. *ACM Computing Surveys*, *11*(4), 397–409. doi:10.1145/356789.356797. URL http://doi.acm.org/10.1145/356789.356797.
18. Bicchi, A., Buss, M., Ernst, M. O., & Peer, A. (Eds.) (2008). *Springer tracts in advanced robotics (STAR): Vol. 45. The sense of touch and its rendering: progresses in haptics research*. Berlin: Springer.
19. Bonner, S., & Kelley, R. B. (1988). A representation scheme for rapid 3-d collision detection. In *IEEE international symposium on intelligent control* (pp. 320–325).
20. Bradshaw, G., & O'Sullivan, C. (2004). Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics*, *23*(1), 1–26. doi:10.1145/966131.966132. URL http://doi.acm.org/10.1145/966131.966132.
21. Brown, J., Sorkin, S., Bruyns, C., Latombe, J.-C., Montgomery, K., & Stephanides, M. (2001). Real-time simulation of deformable objects: tools and application. In *COMP. ANIMATION*.
22. Cameron, S. (1997). Enhancing gjk: computing minimum and penetration distances between convex polyhedra. In *Proceedings of international conference on robotics and automation* (pp. 3112–3117).
23. Canny, J. (1984). *Collision detection for moving polyhedra* (Technical report). Massachusetts Institute of Technology, Cambridge, MA, USA.
24. Chang, C.-T., Gorissen, B., & Melchior, S. (2011). Fast oriented bounding box optimization on the rotation group so(3,ℝ). *ACM Transactions on Graphics*, *30*(5), 122:1–122:16. doi:10.1145/2019627.2019641. URL http://doi.acm.org/10.1145/2019627.2019641.

25. Chang, J.-W., & Kim, M.-S. (2009). Technical section: efficient triangle-triangle intersection test for obb-based collision detection. *Computer Graphics*, *33*(3), 235–240. doi:10.1016/j.cag.2009.03.009.
26. Chazelle, B. (1984). Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM Journal on Computing*, *13*(3), 488–507. doi:10.1137/0213031.
27. Chen, J.-S., & Li, T.-Y. (1999). *Incremental 3D collision detection with hierarchical data structures*. November 22. URL http://citeseer.ist.psu.edu/356263.html; http://bittern.cs.nccu.edu.tw/li/Publication/pdf/vrst98.pdf.
28. Chen, W., Wan, H., Zhang, H., Bao, H., & Peng, Q. (2004). Interactive collision detection for complex and deformable models using programmable graphics hardware. In *Proceedings of the ACM symposium on virtual reality software and technology, VRST '04* (pp. 10–15). New York: ACM. ISBN 1-58113-907-1. doi:10.1145/1077534.1077539. URL http://doi.acm.org/10.1145/1077534.1077539.
29. Chin, F., & Wang, C. A. (1983). Optimal algorithms for the intersection and the minimum distance problems between planar polygons. *IEEE Transactions on Computers*, *32*(12), 1203–1207. doi:10.1109/TC.1983.1676186.
30. Choi, Y.-K., Chang, J.-W., Wang, W., Kim, M.-S., & Elber, G. (2009). Continuous collision detection for ellipsoids. *IEEE Transactions on Visualization and Computer Graphics*, *15*(2), 311–324. URL http://www.ncbi.nlm.nih.gov/pubmed/19147893.
31. Clark, J. H. (1976). Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, *19*(10), 547–554. doi:10.1145/360349.360354. URL http://doi.acm.org/10.1145/360349.360354.
32. Cohen, J. D., Lin, M. C., Manocha, D., & Ponamgi, M. (1995). I-collide: an interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 symposium on interactive 3D graphics, I3D '95* (pp. 189-ff). New York: ACM. ISBN 0-89791-736-7. doi:10.1145/199404.199437. URL http://doi.acm.org/10.1145/199404.199437.
33. Coming, D. S., & Staadt, O. G. (2007). Stride scheduling for time-critical collision detection. In *Proceedings of the 2007 ACM symposium on virtual reality software and technology, VRST '07* (pp. 241–242). New York: ACM. ISBN 978-1-59593-863-3. doi:10.1145/1315184.1315240. URL http://doi.acm.org/10.1145/1315184.1315240.
34. Coming, D. S., & Staadt, O. G. (2008). Velocity-aligned discrete oriented polytopes for dynamic collision detection. *IEEE Transactions on Visualization and Computer Graphics*, *14*(1), 1–12. doi:10.1109/TVCG.2007.70405.
35. Coumans, E. (2005). *Continuous collision detection and physics* (Technical report). Sony Computer Entertainment. August.
36. Coumans, E. (2012). *Bullet physics library*. http://bulletphysics.com.
37. Coutinho, M. G. (2001). *Dynamic simulations of multibody systems*. London: Springer. ISBN 0-387-95192-X.
38. Davis, C., Hegde, M., Schmid, O. A., Maher, M., & Bordes, J. P. (2003). System incorporating physics processing unit 1.
39. Dingliana, J., & O'Sullivan, C. (2000). Graceful degradation of collision handling in physically based animation. *Computer Graphics Forum*, *19*(3), 239–247 (Proc. of EUROGRAPHICS 2000).
40. Djeu, P., Hunt, W., Wang, R., Elhassan, I., Stoll, G., & Razor, W. R. M. (2011). An architecture for dynamic multiresolution ray tracing. *ACM Transactions on Graphics*, *30*(5), 115:1–115:26. doi:10.1145/2019627.2019634. URL http://doi.acm.org/10.1145/2019627.2019634.
41. Dobkin, D. P., Hershberger, J., Kirkpatrick, D. G., & Suri, S. (1993). Computing the intersection-depth of polyhedra. *Algorithmica*, *9*(6), 518–533.
42. Dong, Z., Chen, W., Bao, H., Zhang, H., & Peng, Q. (2004). Real-time voxelization for complex polygonal models. In *Proceedings of the computer graphics and applications, 12th pacific conference, PG '04* (pp. 43–50). Washington: IEEE Computer Society. ISBN 0-7695-2234-3. URL http://dl.acm.org/citation.cfm?id=1025128.1026026.

43. Eberly, D. H. (2003). *Game physics*. New York: Elsevier Science Inc. ISBN 1558607404.
44. Eckstein, J., & Schömer, E. (1999). Dynamic collision detection in virtual reality applications. In V. Skala (Ed.), *WSCG'99 conference proceedings*. URL citeseer.ist.psu.edu/eckstein99dynamic.html.
45. Edelsbrunner, H., & Maurer, H. A. (1981). On the intersection of orthogonal objects. *Information Processing Letters*, *13*(4/5), 177–181. URL http://dblp.uni-trier.de/db/journals/ipl/ipl13.html#EdelsbrunnerM81.
46. Ehmann, S. A., & Lin, M. C. (2001). Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum*, *20*(3), 500–510 (Proc. of EUROGRAPHICS 2001).
47. El-Far, N. R., Georganas, N. D., & El Saddik, A. (2007). Collision detection and force response in highly-detailed point-based hapto-visual virtual environments. In *Proceedings of the 11th IEEE international symposium on distributed simulation and real-time applications, DS-RT '07* (pp. 15–22). Washington: IEEE Computer Society. ISBN 0-7695-3011-7. doi:10.1109/DS-RT.2007.17.
48. Elsen, E., Houston, M., Vishal, V., Darve, E., Hanrahan, P., & Pande, V. (2006). N-body simulation on gpus. In *Proceedings of the 2006 ACM/IEEE conference on supercomputing, SC '06*, New York: ACM. ISBN 0-7695-2700-0. doi:10.1145/1188455.1188649. URL http://doi.acm.org/10.1145/1188455.1188649.
49. Ericson, C. (2004). *The Morgan Kaufmann series in interactive 3-D technology: Real-time collision detection*. San Francisco: Morgan Kaufmann Publishers Inc. ISBN 1558607323.
50. Fahn, C.-S., & Wang, J.-L. (1999). Efficient time-interupted and time-continuous collision detection among polyhedral. *Journal of Information Science and Engineering*, *15*(6), 769–799.
51. Fares, C., & Hamam, A. (2005). Collision detection for rigid bodies: a state of the art review. In *GraphiCon*.
52. Faure, F., Barbier, S., Allard, J., & Falipou, F. (2008). Image-based collision detection and response between arbitrary volumetric objects. In *ACM siggraph/eurographics symposium on computer animation, SCA*, Dublin, Irlande. July 2008.
53. Figueiredo, M., Oliveira, J., Araujo, B., & Madeiras, J. (2010). An efficient collision detection algorithm for point cloud models. In *Proceedings of graphicon*.
54. Fisher, S., & Lin, M. C. (2001). Deformed distance fields for simulation of non-penetrating flexible bodies. In *Proceedings of the eurographic workshop on computer animation and simulation* (pp. 99–111). New York: Springer. ISBN 3-211-83711-6. URL http://dl.acm.org/citation.cfm?id=776350.776360.
55. Fuchs, H., Kedem, Z. M., & Naylor, B. F. (1980). On visible surface generation by a priori tree structures. *SIGGRAPH Computer Graphics*, *14*(3), 124–133. doi:10.1145/965105.807481. URL http://doi.acm.org/10.1145/965105.807481.
56. Fuhrmann, A., Sobotka, G., & Groß, C. (2003). Distance fields for rapid collision detection in physically based modeling. In *Proceedings of GraphiCon 2003* (pp. 58–65). URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.4043&rep=rep1&type=pdf.
57. Fujimoto, A., Tanaka, T., & Iwata, K. (1986). Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, *6*(4), 16–26.
58. Funfzig, C., Ullrich, T., & Fellner, D. W. (2006). Hierarchical spherical distance fields for collision detection. *IEEE Computer Graphics and Applications*, *26*(1), 64–74. doi:10.1109/MCG.2006.17.
59. Fussell, D. S., & Subramanian, K. R. (1988). *Fast ray tracing using k-d trees* (Technical report). University of Texas at Austin, Austin, TX, USA.
60. Ganjugunte, S. K. (2007). A survey on techniques for computing penetration depth.
61. Ganovelli, F., & Dingliana, J. (2000). Buckettree: improving collision detection between deformable objects. In *Proceedings of SCCG2000: spring conference on computer graphics*, Budmerice (pp. 4–6).
62. Giang, T., & O'Sullivan, C. (2005). Closest feature maps for time-critical collision handling. In *International workshop on virtual reality and physical simulation (VRIPHYS'05)*, Novem-

ber (pp. 65–72). URL http://isg.cs.tcd.ie/cosulliv/Pubs/GiangVriphys.pdf.
63. Giang, T., & O'Sullivan, C. (2006). Virtual reality interaction and physical simulation: approximate collision response using closest feature maps. *Computer Graphics*, *30*(3), 423–431. doi:10.1016/j.cag.2006.02.019.
64. Gibson, S. F. F. (1995). Beyond volume rendering: visualization, haptic exploration, and physical modeling of voxel-based objects. In *Proc. eurographics workshop on visualization in scientific computing* (pp. 10–24). Berlin: Springer.
65. Gilbert, E. G., Johnson, D. W., & Keerthi, S. S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, *4*(2), 193–203.
66. Gilbert, E. G., & Ong, C. J. (1994). New distances for the separation and penetration of objects. In *ICRA* (pp. 579–586).
67. Glondu, L., Marchal, M., & Dumont, G. (2010). A new coupling scheme for haptic rendering of rigid bodies interactions based on a haptic sub-world using a contact graph. In *Proceedings of the 2010 international conference on haptics: generating and perceiving tangible sensations, part I, EuroHaptics'10* (pp. 51–56). Berlin: Springer. ISBN 3-642-14063-7, 978-3-642-14063-1. URL http://dl.acm.org/citation.cfm?id=1884164.1884173.
68. Goldsmith, J., & Salmon, J. (1987). Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, *7*(5), 14–20. doi:10.1109/MCG.1987.276983.
69. Gordon, D., & Chen, S. (1991). Front-to-back display of bsp trees. *IEEE Computer Graphics and Applications*, *11*(5), 79–85. doi:10.1109/38.90569.
70. Gottschalk, S., Lin, M. C., & Manocha, D. (1996). Obbtree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on computer graphics and interactive techniques, SIGGRAPH '96* (pp. 171–180). New York: ACM. ISBN 0-89791-746-4. doi:10.1145/237170.237244. URL http://doi.acm.org/10.1145/237170.237244.
71. Gottschalk, S. (1997). Collision detection techniques for 3d models.
72. Gottschalk, S. A. (2000). *Collision queries using oriented bounding boxes*. PhD thesis, The University of North Carolina at Chapel Hill. AAI9993311.
73. Govindaraju, N. K., Knott, D., Jain, N., Kabul, I., Tamstorf, R., Gayle, R., Lin, M. C., & Manocha, D. (2005). Interactive collision detection between deformable models using chromatic decomposition. *ACM Transactions on Graphics*, *24*(3), 991–999. URL http://dblp.uni-trier.de/db/journals/tog/tog24.html#GovindarajuKJKTGLM05.
74. Greene, N., Kass, M., & Miller, G. (1993). Hierarchical z-buffer visibility. In *Proceedings of the 20th annual conference on computer graphics and interactive techniques, SIGGRAPH '93* (pp. 231–238). New York: ACM. ISBN 0-89791-601-8. doi:10.1145/166117.166147. URL http://doi.acm.org/10.1145/166117.166147.
75. Gregory, A., Lin, M. C., Gottschalk, S., & Taylor, R. (1999). A framework for fast and accurate collision detection for haptic interaction. In *Proceedings of the IEEE virtual reality, VR '99* (p. 38). Washington: IEEE Computer Society. ISBN 0-7695-0093-5. URL http://dl.acm.org/citation.cfm?id=554230.835691.
76. Greß, A., Guthe, M., & Klein, R. (2006). Gpu-based collision detection for deformable parameterized surfaces. *Computer Graphics Forum*, *25*(3), 497–506.
77. Guibas, L., Nguyen, A., Russel, D., & Zhang, L. (2002). Collision detection for deforming necklaces. In *Proceedings of the eighteenth annual symposium on computational geometry, SCG '02* (pp. 33–42). New York: ACM. ISBN 1-58113-504-1. doi:10.1145/513400.513405. URL http://doi.acm.org/10.1145/513400.513405.
78. Günther, J., Mannuß, F., & Hinkenjann, A. (2009). Centralized spatial data structures for interactive environments. In *Proceedings of workshop on software engineering and architectures for realtime interactive systems, in conjunction with IEEE virtual reality*. URL http://cg.inf.fh-bonn-rhein-sieg.de/basilic/Publications/2009/GMH09.
79. Günther, O. (1989). The design of the cell tree: an object-oriented index structure for geometric databases. In *ICDE* (pp. 598–605).
80. Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. *SIGMOD Record*, *14*(2), 47–57. doi:10.1145/971697.602266. URL http://doi.acm.org/10.1145/

971697.602266.
81. Hachenberger, P. (2007). Exact Minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra in convex pieces. In *Proceedings of the 15th annual European conference on algorithms, ESA'07* (pp. 669–680). Berlin: Springer. ISBN 3-540-75519-5. URL http://dl.acm.org/citation.cfm?id=1778580.1778642.
82. Haines, E. (1988). Spline surface rendering, and what's wrong with octrees. *Ray Tracing News*, *1*.
83. Hanrahan, P. (1989). A survey of ray-surface intersection algorithms. In A. S. Glassner (Ed.), *An introduction to ray tracing* (pp. 79–119). London: Academic Press Ltd. ISBN 0-12-286160-4. URL http://dl.acm.org/citation.cfm?id=94788.94791.
84. Hasegawa, S., & Sato, M. (2004). Real-time rigid body simulation for haptic interactions based on contact volume of polygonal objects. *Computer Graphics Forum*, *23*(3), 529–538.
85. He, T. (1999). Fast collision detection using quospo trees. In *Proceedings of the 1999 symposium on interactive 3D graphics, I3D '99* (pp. 55–62). New York: ACM. ISBN 1-58113-082-1. doi:10.1145/300523.300529. URL http://doi.acm.org/10.1145/300523.300529.
86. Heidelberger, B., Teschner, M., & Gross, M. (2004). Detection of collisions and self-collisions using image-space techniques. In *Proceedings of the 12th international conference in central Europe on computer graphics, visualization and computer vision'2004 (WSCG'2004)*, University of West Bohemia, Czech Republic, February (pp. 145–152).
87. Held, M. (1998). Erit: a collection of efficient and reliable intersection tests. *Journal of Graphics Tools*, *2*(4), 25–44. URL http://dl.acm.org/citation.cfm?id=763345.763348.
88. Held, M., Klosowski, J. T., & Mitchell, J. S. B. (1996). Collision detection for fly-throughs in virtual environments. In *Proceedings of the twelfth annual symposium on computational geometry, SCG '96* (pp. 513–514). New York: ACM. ISBN 0-89791-804-5. doi:10.1145/237218.237428. URL http://doi.acm.org/10.1145/237218.237428.
89. Ho, C.-H., Basdogan, C., & Srinivasan, M. A. (1999). Efficient point-based rendering techniques for haptic display of virtual objects. *Presence: Teleoperators & Virtual Environments*, *8*(5), 477–491. doi:10.1162/105474699566413.
90. Hoang, T. (1998). *Convex analysis and global optimization. Nonconvex optimization and its applications*. Dordrecht: Kluwer Academic Publishers. ISBN 9780792348184. URL http://books.google.co.uk/books?id=hVkJc2IRDdcC.
91. Horn, D. R., Sugerman, J., Houston, M., & Hanrahan, P. (2007). Interactive k-d tree gpu raytracing. In *Proceedings of the 2007 symposium on interactive 3D graphics and games, I3D '07* (pp. 167–174). New York: ACM. ISBN 978-1-59593-628-8. doi:10.1145/1230100.1230129. URL http://doi.acm.org/10.1145/1230100.1230129.
92. Hubbard, P. M. (1996). Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, *15*(3), 179–210.
93. Hutter, M. (2007). Optimized continuous collision detection for deformable triangle meshes. *Computer*, *15*(1–3), 25–32. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.85.1140&rep=rep1&type=pdf.
94. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., & Fitzgibbon, A. (2011). Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on user interface software and technology, UIST '11* (pp. 559–568). New York: ACM. ISBN 978-1-4503-0716-1. doi:10.1145/2047196.2047270. URL http://doi.acm.org/10.1145/2047196.2047270.
95. James, D. L., & Pai, D. K. (2004). Bd-tree: output-sensitive collision detection for reduced deformable models. In *ACM SIGGRAPH 2004 papers, SIGGRAPH '04* (pp. 393–398). New York: ACM. doi:10.1145/1186562.1015735. URL http://doi.acm.org/10.1145/1186562.1015735.
96. Je, C., Tang, M., Lee, Y., Lee, M., & Kim, Y. J. (2012). Polydepth: real-time penetration depth computation using iterative contact-space projection. *ACM Transactions on Graphics*, *31*(1), 5:1–5:14. doi:10.1145/2077341.2077346. URL http://doi.acm.org/10.1145/2077341.2077346.

97. Jimenez, P., Thomas, F., & Torras, C. (2000). 3d collision detection: a survey. *Computers & Graphics*, *25*, 269–285.
98. Johnson, D. E., & Cohen, E. (1998). A framework for efficient minimum distance computations. In *Proc. IEEE intl. conf. robotics and automation* (pp. 3678–3684).
99. Johnson, D. E., Willemsen, P., & Cohen, E. (2005). 6-dof haptic rendering using spatialized normal cone search. In *Transactions on visualization and computer graphics* (p. 2005).
100. Jolliffe, I. T. (2002). *Principal component analysis*. Berlin: Springer. ISBN 0387954422.
101. Ju, M.-Y., Liu, J.-S., Shiang, S.-P., Chien, Y.-R., Hwang, K.-S., & Lee, W.-C. (2001). Fast and accurate collision detection based on enclosed ellipsoid. *Robotica*, *19*(4), 381–394. doi:10.1017/S0263574700003295.
102. Kallay, M. (1984). The complexity of incremental convex hull algorithms in r^d. *Information Processing Letters*, *19*(4), 197.
103. Kamat, V. V. (1993). A survey of techniques for simulation of dynamic collision detection and response. *Computers & Graphics*, *17*(4), 379–385.
104. Kay, T. L., & Kajiya, J. T. (1986). Ray tracing complex scenes. *SIGGRAPH Computer Graphics*, *20*(4), 269–278. doi:10.1145/15886.15916. URL http://doi.acm.org/10.1145/15886.15916.
105. Kim, B., & Rossignac, J. (2003). Collision prediction for polyhedra under screw motions. In *ACM symposium in solid modeling and applications* (pp. 4–10). New York: ACM Press.
106. Kim, D., Heo, J.-P., & Yoon, S.-e. (2009). Pccd: parallel continuous collision detection. In *SIGGRAPH '09: posters, SIGGRAPH '09* (pp. 50:1–50:1). New York: ACM. doi:10.1145/1599301.1599351. URL http://doi.acm.org/10.1145/1599301.1599351.
107. Kim, J.-P., Lee, B.-C., Kim, H., Kim, J., & Ryu, J. (2009). Accurate and efficient cpu/gpu-based 3-dof haptic rendering of complex static virtual environments. *Presence: Teleoperators & Virtual Environments*, *18*(5), 340–360. doi:10.1162/pres.18.5.340.
108. Kim, Y.-J., Oh, Y.-T., Yoon, S.-H., Kim, M.-S., & Elber, G. (2011). Coons bvh for freeform geometric models. In *Proceedings of the 2011 SIGGRAPH Asia conference, SA '11* (pp. 169:1–169:8). New York: ACM. ISBN 978-1-4503-0807-6. doi:10.1145/2024156.2024203. URL http://doi.acm.org/10.1145/2024156.2024203.
109. Kim, Y. J., Lin, M. C., & Manocha, D. (2002). DEEP: dual-space expansion for estimating penetration depth between convex polytopes. In *ICRA* (pp. 921–926). New York: IEEE. ISBN 0-7803-7273-5.
110. Kim, Y. J., Otaduy, M. A., Lin, M. C., & Manocha, D. (2002). *Fast penetration depth computation using rasterization hardware and hierarchical refinement* (Technical report). Department of Computer Science, University of North Carolina. URL ftp://ftp.cs.unc.edu/pub/publications/techreports/02-014.pdf.
111. Kim, Y. J., Otaduy, M. A., Lin, M. C., & Manocha, D. (2002). Fast penetration depth computation for physically-based animation. In *Proceedings of the 2002 ACM SIGGRAPH/eurographics symposium on computer animation, SCA '02* (pp. 23–31). New York: ACM. ISBN 1-58113-573-4. doi:10.1145/545261.545266. URL http://doi.acm.org/10.1145/545261.545266.
112. Kim, Y. J., Otaduy, M. A., Lin, M. C., & Manocha, D. (2003). Six-degree-of-freedom haptic rendering using incremental and localized computations. *Presence: Teleoperators & Virtual Environments*, *12*(3), 277–295. doi:10.1162/105474603765879530.
113. Kim, Y. J., Lin, M. C., & Manocha, D. (2004). Incremental penetration depth estimation between convex polytopes using dual-space expansion. *IEEE Transactions on Visualization and Computer Graphics*, *10*(2), 152–163. doi:10.1109/TVCG.2004.1260767.
114. Kipfer, P. (2007). LCP algorithms for collision detection using CUDA. In H. Nguyen (Ed.), *GPUGems 3* (pp. 723–739). Reading: Addison-Wesley.
115. Klein, J., & Zachmann, G. (2003). Adb-trees: controlling the error of time-critical collision detection. In T. Ertl, B. Girod, G. Greiner, H. Niemann, H.-P. Seidel, E. Steinbach, & R. Westermann (Eds.), *Vision, modeling and visualisation 2003* (pp. 37–46). Berlin: Akademische Verlagsgesellschaft Aka GmbH. ISBN 3-89838-048-3.

116. Klein, J., & Zachmann, G. (2004). Point cloud collision detection. In M.-P. Cani & M. Slater (Eds.), *Computer graphics forum (Proc. EUROGRAPHICS)*, Grenoble, France, Aug. 30–Sep. 3 (Vol. 23, pp. 567–576). URL http://www.gabrielzachmann.org/.
117. Klein, J., & Zachmann, G. (2005). Interpolation search for point cloud intersection. In *Proc. of WSCG 2005*, University of West Bohemia, Plzen, Czech Republic, January 31–February 7 (pp. 163–170). ISBN 80-903100-7-9. URL http://www.gabrielzachmann.org/.
118. Klosowski, J. T., Held, M., Mitchell, J. S. B., Sowizral, H., & Zikan, K. (1998). Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, *4*(1), 21–36. doi:10.1109/2945.675649.
119. Knott, D., & Pai, D. (2003). Cinder: collision and interference detection in real–time using graphics hardware. URL citeseer.ist.psu.edu/knott03cinder.html.
120. Kobbelt, L., & Botsch, M. (2004). A survey of point-based techniques in computer graphics. *Computers & Graphics*, *28*(6), 801–814.
121. Kockara, S., Halic, T., Iqbal, K., Bayrak, C., & Rowe, R. (2007). Collision detection: a survey. In *SMC* (pp. 4046–4051). New York: IEEE.
122. Kolb, A., Latta, L., & Rezk-Salama, C. (2004). Hardware-based simulation and collision detection for large particle systems. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on graphics hardware, HWWS '04* (pp. 123–131). New York: ACM. ISBN 3-905673-15-0. doi:10.1145/1058129.1058147. URL http://doi.acm.org/10.1145/1058129.1058147.
123. Kolesnikov, M., & Zefran, M. (2007). Energy-based 6-dof penetration depth computation for penalty-based haptic rendering algorithms. In *IROS* (pp. 2120–2125).
124. Krishnan, S., Gopi, M., Lin, M., Manocha, D., & Pattekar, A. (1998). Rapid and accurate contact determination between spline models using shelltrees.
125. Krishnan, S., Pattekar, A., Lin, M. C., & Manocha, D. (1998). Spherical shell: a higher order bounding volume for fast proximity queries. In *Proceedings of the third workshop on the algorithmic foundations of robotics on robotics: the algorithmic perspective, WAFR '98* (pp. 177–190). Natick: A. K. Peters, Ltd. ISBN 1-56881-081-4. URL http://dl.acm.org/citation.cfm?id=298960.299006.
126. Larsen, E., Gottschalk, S., Lin, M. C., & Manocha, D. (1999). *Fast proximity queries with swept sphere volumes*, November 14. URL http://citeseer.ist.psu.edu/408975.html; ftp://ftp.cs.unc.edu/pub/users/manocha/PAPERS/COLLISION/ssv.ps.
127. Larsson, T., & Akenine-Möller, T. (2001). Collision detection for continuously deforming bodies. In *Eurographics 2001, short presentations* (pp. 325–333). Geneve: Eurographics Association. URL http://www.mrtc.mdh.se/index.php?choice=publications&id=0354.
128. Larsson, T., & Akenine-Möller, T. (2003). Efficient collision detection for models deformed by morphing. *The Visual Computer*, *19*(2–3), 164–174. URL http://www.mrtc.mdh.se/index.phtml?choice=publications&id=0551.
129. Larsson, T., & Akenine-Möller, T. (2006). A dynamic bounding volume hierarchy for generalized collision detection. *Computer Graphics*, *30*(3), 450–459. doi:10.1016/j.cag.2006.02.011.
130. Larsson, T., & Akenine-Möller, T. (2009). Bounding volume hierarchies of slab cut balls. *Computer Graphics Forum*, *28*(8), 2379–2395. URL http://dblp.uni-trier.de/db/journals/cgf/cgf28.html#LarssonA09.
131. Lau, R. W. H., Chan, O., Luk, M., & Li, F. W. B. (2002). Large a collision detection framework for deformable objects. In *Proceedings of the ACM symposium on virtual reality software and technology, VRST '02* (pp. 113–120). New York: ACM. ISBN 1-58113-530-0. doi:10.1145/585740.585760. URL http://doi.acm.org/10.1145/585740.585760.
132. Lauterbach, C., Garland, M., Sengupta, S., Luebke, D. P., & Manocha, D. (2009). Fast bvh construction on gpus. *Computer Graphics Forum*, *28*(2), 375–384. URL http://dblp.uni-trier.de/db/journals/cgf/cgf28.html#LauterbachGSLM09.
133. Lauterbach, C., Mo, Q., & Manocha, D. (2010). gproximity: hierarchical gpu-based operations for collision and distance queries. *Computer Graphics Forum*, *29*(2), 419–428. URL http://dblp.uni-trier.de/db/journals/cgf/cgf29.html#LauterbachMM10.

134. LaValle, S. M. (2006). *Planning algorithms*. Cambridge: Cambridge University Press. Available at http://planning.cs.uiuc.edu/.
135. Le Grand, S. (2008). Broad-phase collision detection with CUDA. In *GPU gems 3* (pp. 697–721). URL http://http.developer.nvidia.com/GPUGems3/gpugems3_ch32.html.
136. Leon, C. A. D., Eliuk, S., & Gomez, H. T. (2010). Simulating soft tissues using a gpu approach of the mass-spring model. In B. Lok, G. Klinker, & R. Nakatsu (Eds.), *VR* (pp. 261–262). New York: IEEE. ISBN 978-1-4244-6258-2.
137. Leutenegger, S. T., Edgington, J. M., & Lopez, M. A. (1997). *Str: a simple and efficient algorithm for r-tree packing* (Technical report). Institute for Computer Applications in Science and Engineering (ICASE).
138. Lien, J.-M., & Amato, N. M. (2008). Approximate convex decomposition of polyhedra and its applications. *Computer Aided Geometric Design*, *25*(7), 503–522. doi:10.1016/j.cagd.2008.05.003.
139. Lin, M. C., & Canny, J. F. (1991). A fast algorithm for incremental distance calculation. In *IEEE international conference on robotics and automation* (pp. 1008–1014).
140. Lin, M. C., & Gottschalk, S. (1998). Collision detection between geometric models: a survey. In *Proc. of IMA conference on mathematics of surfaces* (pp. 37–56).
141. Lin, M. C., Otaduy, M., Lin, M. C., & Otaduy, M. (2008). *Haptic rendering: foundations, algorithms and applications*. Natick: A. K. Peters, Ltd. ISBN 1568813325.
142. Lin, Y.-T., & Li, T.-Y. (2006). A time-budgeted collision detection method. In *ICRA* (pp. 3029–3034). New York: IEEE.
143. Liu, F., Harada, T., Lee, Y., & Kim, Y. J. (2010). Real-time collision culling of a million bodies on graphics processing units. *ACM Transactions on Graphics*, *29*(6), 154:1–154:8. doi:10.1145/1882261.1866180. URL http://doi.acm.org/10.1145/1882261.1866180.
144. Liu, J.-S., Kao, J.-I., & Chang, Y.-Z. (2006). Collision detection of deformable polyhedral objects via inner-outer ellipsoids. In *IROS* (pp. 5600–5605). New York: IEEE. URL http://dblp.uni-trier.de/db/conf/iros/iros2006.html#LiuKC06.
145. Liu, M., Wang, D., & Zhang, Y. (2010). A novel haptic rendering algorithm for stable and precise 6-dof virtual assembly. In *Proceedings of the ASME 2010 world conference on innovative virtual reality, WINVR2010* (pp. 1–7).
146. Luebke, D. (2003). *The Morgan Kaufmann series in computer graphics and geometric modeling. Level of detail for 3D graphics*. San Francisco: Morgan Kaufmann. ISBN 9781558608382. URL http://books.google.de/books?id=CB1N1aaoMloC.
147. Luque, R. G., Comba, J. L. D., & Freitas, C. M. D. S. (2005). Broad-phase collision detection using semi-adjusting bsp-trees. In *Proceedings of the 2005 symposium on interactive 3D graphics and games, I3D '05* (pp. 179–186). New York: ACM. ISBN 1-59593-013-2. doi:10.1145/1053427.1053457. URL http://doi.acm.org/10.1145/1053427.1053457.
148. Massie, T. H., & Salisbury, K. J. (1994). Phantom haptic interface: a device for probing virtual objects. *American Society of Mechanical Engineers, Dynamic Systems and Control Division (Publication) DSC*, *55*(1), 295–299.
149. Mazhar, H. (2009). Gpu collision detection using spatial subdivision with applications in contact dynamics. In *ASME IDETC conference*.
150. McGuire, M., & Luebke, D. (2009). Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the conference on high performance graphics 2009, HPG '09* (pp. 77–89). New York: ACM. ISBN 978-1-60558-603-8. doi:10.1145/1572769.1572783. URL http://doi.acm.org/10.1145/1572769.1572783.
151. McNeely, W. A., Puterbaugh, K. D., & Troy, J. J. (1999). Six degrees-of-freedom haptic rendering using voxel sampling. *ACM Transactions on Graphics*, *18*(3), 401–408 (SIGGRAPH 1999).
152. McNeely, W. A., Puterbaugh, K. D., & Troy, J. J. (2005). Advances in voxel-based 6-dof haptic rendering. In *ACM SIGGRAPH 2005 courses, SIGGRAPH '05*. New York: ACM. doi:10.1145/1198555.1198606. URL http://doi.acm.org/10.1145/1198555.1198606.
153. McNeely, W. A., Puterbaugh, K. D., & Troy, J. J. (2006). Voxel-based 6-dof haptic rendering improvements. *Hapticse: The Electronic Journal of Haptics Research*, *3*(7).

154. Mendoza, C., & O'Sullivan, C. (2006). Interruptible collision detection for deformable objects. *Computer Graphics*, *30*(3), 432–438. doi:10.1016/j.cag.2006.02.018.
155. Mezger, J., Kimmerle, S., & Etzmuß, O. (2003). Hierarchical techniques in collision detection for cloth animation. *Journal of WSCG*, *11*(2), 322–329.
156. Mirtich, B. (1998). Efficient algorithms for two-phase collision detection. In K. Gupta & A. P. del Pobil (Eds.), *Practical motion planning in robotics: current approaches and future directions* (pp. 203–223). New York: Wiley.
157. Mirtich, B. (1998). V-clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, *17*(3), 177–208. doi:10.1145/285857.285860. URL http://doi.acm.org/10.1145/285857.285860.
158. Mirtich, B. (2000). Timewarp rigid body simulation. In *Proceedings of the 27th annual conference on computer graphics and interactive techniques, SIGGRAPH '00* (pp. 193–200). New York: ACM Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5. doi:10.1145/344779.344866.
159. Möller, T. (1997). A fast triangle-triangle intersection test. *Journal of Graphics Tools*, *2*(2), 25–30. URL http://dl.acm.org/citation.cfm?id=272317.272320.
160. Mosegaard, J., Herborg, P., & Sørensen, T. S. (2005). A GPU accelerated spring mass system for surgical simulation. *Studies in Health Technology and Informatics*, *111*, 342–348. URL http://view.ncbi.nlm.nih.gov/pubmed/15718756.
161. Munkberg, J., Hasselgren, J., Toth, R., & Akenine-Möller, T. (2010). Efficient bounding of displaced Bezier patches. In *Proceedings of the conference on high performance graphics, HPG '10* (pp. 153–162). Aire-la-Ville: Eurographics Association. URL http://dl.acm.org/citation.cfm?id=1921479.1921503.
162. Naylor, B. F. (1992). Interactive solid geometry via partitioning trees. In *Proceedings of the conference on graphics interface '92* (pp. 11–18). San Francisco: Morgan Kaufmann Publishers. ISBN 0-9695338-1-0. URL http://dl.acm.org/citation.cfm?id=155294.155296.
163. NVIDIA (2012). *Nvidia physx*. http://www.nvidia.com/object/nvidia_physx.html.
164. O'Brien, J. F., & Hodgins, J. K. (1999). Graphical modeling and animation of brittle fracture. In *Proceedings of the 26th annual conference on computer graphics and interactive techniques, SIGGRAPH '99* (pp. 137–146). New York: ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-48560-5. doi:10.1145/311535.311550.
165. Ong, C. J., Huang, E., & Hong, S.-M. (2000). A fast growth distance algorithm for incremental motions. *IEEE Transactions on Robotics*, *16*(6), 880–890.
166. O'Rourke, J. (1984). *Finding minimal enclosing boxes* (Technical Report). Johns Hopkins Univ., Baltimore, MD.
167. Ortega, M., Redon, S., & Coquillart, S. (2007). A six degree-of-freedom god-object method for haptic display of rigid bodies with surface properties. *IEEE Transactions on Visualization and Computer Graphics*, *13*(3), 458–469. doi:10.1109/TVCG.2007.1028.
168. O'Sullivan, C., & Dingliana, J. (1999). Real-time collision detection and response using sphere-trees.
169. O'Sullivan, C., & Dingliana, J. (2001). Collisions and perception. *ACM Transactions on Graphics*, *20*(3), 151–168. doi:10.1145/501786.501788. URL http://doi.acm.org/10.1145/501786.501788.
170. Otaduy, M. A., & Lin, M. C. (2006). *A modular haptic rendering algorithm for stable and transparent 6-dof manipulation*. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1668258.
171. Otaduy, M. A., & Lin, M. C. (2003). CLODs: Dual hierarchies for multiresolution collision detection. In *Symposium on geometry processing* (pp. 94–101).
172. Otaduy, M. A., & Lin, M. C. (2005). Sensation preserving simplification for haptic rendering. In *ACM SIGGRAPH 2005 courses, SIGGRAPH '05*. New York: ACM. doi:10.1145/1198555.1198607. URL http://doi.acm.org/10.1145/1198555.1198607.
173. Otaduy, M. A., Jain, N., Sud, A., & Lin, M. C. (2004). *Haptic rendering of interaction between textured models* (Technical report). University of North Carolina

Chapel Hill, April 13. URL http://citeseer.ist.psu.edu/638785.html; ftp://ftp.cs.unc.edu/pub/publications/techreports/04-007.pdf.
174. Otaduy, M. A., Chassot, O., Steinemann, D., & Gross, M. (2007). Balanced hierarchies for collision detection between fracturing objects. In *Virtual reality conference, IEEE* (pp. 83–90). New York: IEEE. URL http://doi.ieeecomputersociety.org/10.1109/VR.2007.352467.
175. Page, F., & Guibault, F. (2003). Collision detection algorithm for nurbs surfaces in interactive applications. In *Canadian conference on electrical and computer engineering, 2003. IEEE CCECE 2003*, May 2003 (Vol. 2, pp. 1417–1420). doi:10.1109/CCECE.2003.1226166.
176. Pan, J., & Manocha, D. (2012). Gpu-based parallel collision detection for fast motion planning. *The International Journal of Robotics Research*, *31*(2), 187–200. doi:10.1177/0278364911429335.
177. Pan, J., Chitta, S., & Manocha, D. (2011). Probabilistic collision detection between noisy point clouds using robust classification. In *International symposium on robotics research*, Flagstaff, Arizona, 08/2011. URL http://www.isrr-2011.org/ISRR-2011//Program_files/Papers/Pan-ISRR-2011.pdf.
178. Pan, J., Chitta, S., & Manocha, D. (2012). Proximity computations between noisy point clouds using robust classification. In *RGB-D: advanced reasoning with depth cameras*, Los Angeles, California, 06/2012. URL http://www.cs.washington.edu/ai/Mobile_Robotics/rgbd-workshop-2011/.
179. Pantaleoni, J. (2011). Voxelpipe: a programmable pipeline for 3d voxelization. In *Proceedings of the ACM SIGGRAPH symposium on high performance graphics, HPG '11* (pp. 99–106). New York: ACM. ISBN 978-1-4503-0896-0. doi:10.1145/2018323.2018339. URL http://doi.acm.org/10.1145/2018323.2018339.
180. Park, S.-H., & Ryu, K. (2004). Fast similarity search for protein 3d structure databases using spatial topological patterns. In F. Galindo, M. Takizawa, & R. Traunmüller (Eds.), *Lecture notes in computer science: Vol. 3180. Database and expert systems applications* (pp. 771–780). Berlin: Springer. doi:10.1007/978-3-540-30075-5_74.
181. Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., & Stich, M. (2010). Optix: a general purpose ray tracing engine. *ACM Transactions on Graphics*, *29*(4), 66:1–66:13. doi:10.1145/1778765.1778803. URL http://doi.acm.org/10.1145/1778765.1778803.
182. Ponamgi, M., Manocha, D., & Lin, M. C. (1995). Incremental algorithms for collision detection between solid models. In *Proceedings of the third ACM symposium on solid modeling and applications, SMA '95* (pp. 293–304). New York: ACM. ISBN 0-89791-672-7. doi:10.1145/218013.218076. URL http://doi.acm.org/10.1145/218013.218076.
183. Prior, A., & Haines, K. (2005). The use of a proximity agent in a collaborative virtual environment with 6 degrees-of-freedom voxel-based haptic rendering. In *Proceedings of the first joint eurohaptics conference and symposium on haptic interfaces for virtual environment and teleoperator systems, WHC '05* (pp. 631–632). Washington: IEEE Computer Society. ISBN 0-7695-2310-2. doi:10.1109/WHC.2005.137.
184. Provot, X. (1997). Collision and self-collision handling in cloth model dedicated to design garments. In *Proc. graphics interface '97* (pp. 177–189).
185. Quinlan, S. (1994). Efficient distance computation between non-convex objects. In *Proceedings of international conference on robotics and automation* (pp. 3324–3329).
186. Raabe, A., Bartyzel, B., Anlauf, J. K., & Zachmann, G. (2005). Hardware accelerated collision detection—an architecture and simulation results. In *Proceedings of the conference on design, automation and test in Europe, DATE '05* (Vol. 3, pp. 130–135). Washington: IEEE Computer Society. ISBN 0-7695-2288-2. doi:10.1109/DATE.2005.167.
187. Redon, S., Kheddar, A., & Coquillart, S. (2000). An algebraic solution to the problem of collision detection for rigid polyhedral objects. In *Proceedings 2000 ICRA millennium conference IEEE international conference on robotics and automation symposia proceedings cat No00CH37065*, April (Vol. 4, pp. 3733–3738). URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=845313.

188. Redon, S., & Lin, C. M. (2006). A fast method for local penetration depth computation. *Journal of Graphics Tools*. URL http://hal.inria.fr/inria-00390349.
189. Redon, S., Kheddar, A., & Coquillart, S. (2002). Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, *21*(3), 279–287. URL http://dblp.uni-trier.de/db/journals/cgf/cgf21.html#RedonKC02.
190. Renz, M., Preusche, C., Potke, M., Kriegel, H.-P., & Hirzinger, G. (2001). Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm. In *Proc. eurohaptics* (pp. 149–154).
191. Roussopoulos, N., & Leifker, D. (1985). Direct spatial search on pictorial databases using packed r-trees. In *Proceedings of the 1985 ACM SIGMOD international conference on management of data, SIGMOD '85* (pp. 17–31). New York: ACM. ISBN 0-89791-160-1. doi:10.1145/318898.318900. URL http://doi.acm.org/10.1145/318898.318900.
192. Rubin, S. M., & Whitted, T. (1980). A 3-dimensional representation for fast rendering of complex scenes. In *Proceedings of the 7th annual conference on computer graphics and interactive techniques, SIGGRAPH '80* (pp. 110–116). New York: ACM. ISBN 0-89791-021-4. doi:10.1145/800250.807479. URL http://doi.acm.org/10.1145/800250.807479.
193. Ruffaldi, E., Morris, D., Barbagli, F., Salisbury, K., & Bergamasco, M. (2008). Voxel-based haptic rendering using implicit sphere trees. In *Proceedings of the 2008 symposium on haptic interfaces for virtual environment and teleoperator systems, HAPTICS '08* (pp. 319–325). Washington: IEEE Computer Society. ISBN 978-1-4244-2005-6. doi:10.1109/HAPTICS.2008.4479964.
194. Ruspini, D. C., Kolarov, K., & Khatib, O. (1997). The haptic display of complex graphical environments. In *Proceedings of the 24th annual conference on computer graphics and interactive techniques, SIGGRAPH '97* (pp. 345–352). New York: ACM Press/Addison-Wesley Publishing Co. ISBN 0-89791-896-7. doi:10.1145/258734.258878.
195. Salisbury, K., Conti, F., & Barbagli, F. (2004). Haptic rendering: introductory concepts. *IEEE Computer Graphics and Applications*, *24*, 24–32. URL http://doi.ieeecomputersociety.org/10.1109/MCG.2004.10030.
196. Samet, H. (1989). Implementing ray tracing with octrees and neighbor finding. *Computers & Graphics*, *13*, 445–460.
197. Schneider, P. J., & Eberly, D. (2002). *Geometric tools for computer graphics*. New York: Elsevier Science Inc. ISBN 1558605940.
198. Schwarz, M., & Seidel, H.-P. (2010). Fast parallel surface and solid voxelization on gpus. In *ACM SIGGRAPH Asia 2010 papers, SIGGRAPH ASIA '10* (pp. 179:1–179:10). New York: ACM. ISBN 978-1-4503-0439-9. doi:10.1145/1866158.1866201. URL http://doi.acm.org/10.1145/1866158.1866201.
199. Selinger, A., & Nelson, R. C. (1999). A perceptual grouping hierarchy for appearance-based 3d object recognition. *Computer Vision and Image Understanding*, *76*(1), 83–92. doi:10.1006/cviu.1999.0788.
200. Shirley, P., & Morley, R. K. (2003). *Realistic ray tracing* (2nd ed.). Natick: A. K. Peters, Ltd. ISBN 1568811985.
201. Six, H.-W., & Widmayer, P. (1992). Spatial access structures for geometric databases. In B. Monien & Th. Ottmann (Eds.), *Lecture notes in computer science: Vol. 594. Data structures and efficient algorithms* (pp. 214–232). Berlin: Springer. ISBN 978-3-540-55488-2. doi:10.1007/3-540-55488-2_29.
202. Smith, A., Kitamura, Y., Takemura, H., & Kishino, F. (1995). A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. In *Proceedings of the virtual reality annual international symposium VRAIS'95* (p. 136). Washington: IEEE Computer Society. ISBN 0-8186-7084-3. URL http://dl.acm.org/citation.cfm?id=527216.836015.
203. Smith, R. (2012). *Open dynamics engine*. http://www.ode.org.
204. Sobottka, G., & Weber, A. (2005). Efficient bounding volume hierarchies for hair simulation. In *The 2nd workshop in virtual reality interactions and physical simulations (VRIPHYS '05)*, November.

205. Sobottka, G., Varnik, E., & Weber, A. (2005). Collision detection in densely packed fiber assemblies with application to hair modeling. In H. R. Arabnia (Ed.), *The 2005 international conference on imaging science, systems, and technology: computer graphics (CISST'05)* (pp. 244–250). Athens: CSREA Press. ISBN 1-932415-64-5.
206. Spillmann, J., Becker, M., & Teschner, M. (2007). Efficient updates of bounding sphere hierarchies for geometrically deformable models. *Journal of Visual Communication and Image Representation*, *18*(2), 101–108. doi:10.1016/j.jvcir.2007.01.001.
207. Stewart, D., & Trinkle, J. C. (1996). An implicit time-stepping scheme for rigid body dynamics with coulomb friction. *International Journal for Numerical Methods in Biomedical Engineering*, *39*, 2673–2691.
208. Su, C.-J., Lin, F., & Ye, L. (1999). A new collision detection method for csg-represented objects in virtual manufacturing. *Computers in Industry*, *40*(1), 1–13. doi:10.1016/S0166-3615(99)00010-X.
209. Suffern, K. (2007). *Ray tracing from the ground up*. Natick: A. K. Peters, Ltd. ISBN 1568812728.
210. Taeubig, H., & Frese, U. (2012). A new library for real-time continuous collision detection. In *Proceedings of the 7th German conference on robotics (ROBOTIK-2012)*, May 21–22. Munich, Germany. Frankfurt am Main: VDE.
211. Tang, C., Li, S., & Wang, G. (2011). Fast continuous collision detection using parallel filter in subspace. In *Symposium on interactive 3D graphics and games, I3D '11* (pp. 71–80). New York: ACM. ISBN 978-1-4503-0565-5. doi:10.1145/1944745.1944757. URL http://doi.acm.org/10.1145/1944745.1944757.
212. Tang, M., Kim, Y. J., & Manocha, D. (2009). C2a: controlled conservative advancement for continuous collision detection of polygonal models. In *Proceedings of international conference on robotics and automation*.
213. Tang, M., Lee, M., & Kim, Y. J. (2009). Interactive Hausdorff distance computation for general polygonal models. In *ACM SIGGRAPH 2009 papers, SIGGRAPH '09* (pp. 74:1–74:9). New York: ACM. ISBN 978-1-60558-726-4. doi:10.1145/1576246.1531380. URL http://doi.acm.org/10.1145/1576246.1531380.
214. Tang, M., Manocha, D., Otaduy, M. A., & Tong, R. (2012). Continuous penalty forces. *ACM Transactions on Graphics*, *31*(4) (Proc. of ACM SIGGRAPH). URL http://www.gmrv.es/Publications/2012/TMOT12.
215. Tang, M., Tang, M., Curtis, S., Yoon, S.-E., Yoon, S.-E., & Manocha, D. (2008). *Iccd: interactive continuous collision detection between deformable models using connectivity-based culling*. URL http://www.ncbi.nlm.nih.gov/pubmed/19423880.
216. Tasora, A., Negrut, D., & Anitescu, M. (2009). Gpu-based parallel computing for the simulation of complex multibody systems with unilateral and bilateral constraints: an overview.
217. Tavares, D. L. M., & Comba, J. L. D. (2007). *Broad-phase collision detection using Delaunay triangulation* (Technical report). Universidade Federal do Rio Grande do Sul (UFRGS).
218. Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnenat-Thalmann, N., Strasser, W., & Volino, P. (2005). Collision detection for deformable objects. *Computer Graphics Forum*, *24*(1), 61–81. doi:10.1111/j.1467-8659.2005.00829.x.
219. Teschner, M., Heidelberger, B., Müller, M., Pomerantes, D., & Gross, M. H. (2003). Optimized spatial hashing for collision detection of deformable objects. In *Proc. 8th international fall workshop vision, modeling, and visualization (VMV 2003)* (pp. 47–54).
220. Thompson, T. V. II., Johnson, D. E., & Cohen, E. (1997). Direct haptic rendering of sculptured models. In *Proceedings of the 1997 symposium on interactive 3D graphics, I3D '97* (pp. 167–176). New York: ACM. ISBN 0-89791-884-3. doi:10.1145/253284.253336. URL http://doi.acm.org/10.1145/253284.253336.
221. Torres, R., Martín, P. J., & Gavilanes, A. (2009). Ray casting using a roped bvh with cuda. In *Proceedings of the 2009 spring conference on computer graphics, SCCG '09* (pp. 95–102). New York: ACM. ISBN 978-1-4503-0769-7. doi:10.1145/1980462.1980483. URL http://doi.acm.org/10.1145/1980462.1980483.

222. Tropp, O., Tal, A., & Shimshoni, I. (2006). A fast triangle to triangle intersection test for collision detection. *Computer Animation and Virtual Worlds*, *17*(5), 527–535. URL http://doi.wiley.com/10.1002/cav.115.
223. Tsingos, N., Dachsbacher, C., Lefebvre, S., & Dellepiane, M. (2007). Instant sound scattering. In *Rendering techniques (Proceedings of the eurographics symposium on rendering)*. URL http://www-sop.inria.fr/reves/Basilic/2007/TDLD07.
224. Turk, G. (1989). *Interactive collision detection for molecular graphics* (Technical report). University of North Carolina at Chapel Hill. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.4927.
225. van den Bergen, G. (1998). Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, *2*(4), 1–13. URL http://dl.acm.org/citation.cfm?id=763345.763346.
226. Van den Bergen, G. (1999). A fast and robust gjk implementation for collision detection of convex objects. *Journal of Graphics Tools*, *4*(2), 7–25. URL http://dl.acm.org/citation.cfm?id=334709.334711.
227. van den Bergen, G. (2001). Proximity queries and penetration depth computation on 3D game objects. In *Proceedings of game developers conference 2001*, San Jose, CA, March.
228. Van Den Bergen, G. (2004). *The Morgan Kaufmann series in interactive 3D technology*. *Collision detection in interactive 3D environments*. San Francisco: Morgan Kaufman Publishers. ISBN 9781558608016. URL http://books.google.com/books?id=E-9AsqZCTSEC.
229. Volino, P., & Magnenat Thalmann, N. M. (1995). Collision and self-collision detection: efficient and robust solutions for highly deformable surfaces. In *Computer animation and simulation '95* (pp. 55–65). Berlin: Springer.
230. Von Herzen, B., Barr, A. H., & Zatz, H. R. (1990). Geometric collisions for time-dependent parametric surfaces. In *Proceedings of the 17th annual conference on computer graphics and interactive techniques, SIGGRAPH '90* (pp. 39–48). New York: ACM. ISBN 0-89791-344-2. doi:10.1145/97879.97883. URL http://doi.acm.org/10.1145/97879.97883.
231. Voronoi, G. (1908). Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs. *Journal für die Reine und Angewandte Mathematik (Crelles Journal)*, *134*, 198–287. doi:10.1515/crll.1908.134.198.
232. Wald, I. (2007). On fast construction of sah-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE symposium on interactive ray tracing, RT '07* (pp. 33–40). Washington: IEEE Computer Society. ISBN 978-1-4244-1629-5. doi:10.1109/RT.2007.4342588.
233. Wald, I., & Havran, V. (2006). On building fast kd-trees for ray tracing, and on doing that in o(n log n). In *Symposium on interactive ray tracing* (pp. 61–69). URL http://doi.ieeecomputersociety.org/10.1109/RT.2006.280216.
234. Wand, M. (2004). *Point-based multi-resolution rendering*. PhD thesis, Department of computer science and cognitive science, University of Tübingen.
235. Weghorst, H., Hooper, G., & Greenberg, D. P. (1984). Improved computational methods for ray tracing. *ACM Transactions on Graphics*, *3*(1), 52–69.
236. Welzl, E. (1991). Smallest enclosing disks (balls and ellipsoids). In *Results and new trends in computer science* (pp. 359–370). Berlin: Springer.
237. Whang, K.-Y., Song, J.-W., Chang, J.-W., Kim, J.-Y., Cho, W.-S., Park, C.-M., & Song, I.-Y. (1995). Octree-r: an adaptive octree for efficient ray tracing. *IEEE Transactions on Visualization and Computer Graphics*, *1*, 343–349. URL http://doi.ieeecomputersociety.org/10.1109/2945.485621.
238. Whitted, T. (1980). An improved illumination model for shaded display. *Communications of the ACM*, *23*(6), 343–349. doi:10.1145/358876.358882. URL http://doi.acm.org/10.1145/358876.358882.
239. Wong, S.-K. (2011). Adaptive continuous collision detection for cloth models using a skipping frame session. *Journal of Information Science and Engineering*, *27*(5), 1545–1559.
240. Wong, W. S.-K., & Baciu, G. (2005). Gpu-based intrinsic collision detection for deformable surfaces. *Computer Animation and Virtual Worlds*, *16*(3–4), 153–161. doi:10.1002/cav.104.

241. Woulfe, M., Dingliana, J., & Manzke, M. (2007). Hardware accelerated broad phase collision detection for realtime simulations. In J. Dingliana & F. Ganovelli (Eds.), *Proceedings of the 4th workshop on virtual reality interaction and physical simulation (VRIPHYS 2007)* (pp. 79–88). Aire-la-Ville: Eurographics Association. URL https://www.cs.tcd.ie/~woulfem/publications/paper2007/.
242. Yilmaz, T., & Gudukbay, U. (2007). Conservative occlusion culling for urban visualization using a slice-wise data structure. *Graphical Models*, *69*(3–4), 191–210. doi:10.1016/j.gmod.2007.01.002.
243. Yoon, S.-E., Salomon, B., Lin, M., & Manocha, D. (2004). Fast collision detection between massive models using dynamic simplification. In *Proceedings of the 2004 eurographics/ACM SIGGRAPH symposium on geometry processing, SGP '04* (pp. 136–146). New York: ACM. ISBN 3-905673-13-4. doi:10.1145/1057432.1057450. URL http://doi.acm.org/10.1145/1057432.1057450.
244. Yoon, S.-E., Curtis, S., & Manocha, D. (2007). Ray tracing dynamic scenes using selective restructuring. In *ACM SIGGRAPH 2007 sketches, SIGGRAPH '07*. New York: ACM. doi:10.1145/1278780.1278847. URL http://doi.acm.org/10.1145/1278780.1278847.
245. Zachmann, G. (1998). Rapid collision detection by dynamically aligned dop-trees. In *Proceedings of the virtual reality annual international symposium, VRAIS '98* (p. 90). Washington: IEEE Computer Society. ISBN 0-8186-8362-7. URL http://dl.acm.org/citation.cfm?id=522258.836122.
246. Zachmann, G. (2000). *Virtual reality in assembly simulation—collision detection, simulation algorithms, and interaction techniques*. Dissertation, Darmstadt University of Technology, Germany, May.
247. Zachmann, G. (2001). Optimizing the collision detection pipeline. In *Proc. of the first international game technology conference (GTEC)*, January.
248. Zachmann, G. (2002). Minimal hierarchical collision detection. In *Proceedings of the ACM symposium on virtual reality software and technology, VRST '02* (pp. 121–128). New York: ACM. ISBN 1-58113-530-0. doi:10.1145/585740.585761. URL http://doi.acm.org/10.1145/585740.585761.
249. Zachmann, G., & Langetepe, E. (2003). Geometric data structures for computer graphics. In *Proc. of ACM SIGGRAPH. ACM transactions of graphics*, 27–31 July. URL http://www.gabrielzachmann.org/.
250. Zachmann, G., Teschner, M., Kimmerle, S., Heidelberger, B., Raghupathi, L., & Fuhrmann, A. (2005). Real-time collision detection for dynamic virtual environments. In *Tutorial #4, IEEE VR*, Bonn, Germany, 12–16 March. Washington: IEEE Computer Society.
251. Zeiller, M. (1993). Collision detection for objects modelled by csg. In T. K. S. Murthy, J. J. Conner, S. Hernandez, & H. Power (Eds.), *Visualization and intelligent design in engineering and architecture*, April. Amsterdam: Elsevier Science Publishers. ISBN 1853122270. URL http://www.cg.tuwien.ac.at/research/publications/1993/zeiller-1993-coll/.
252. Zhang, D., & Yuen, M. M. F. (2000). Collision detection for clothed human animation. In *Pacific conference on computer graphics and applications* (p. 328). URL http://doi.ieeecomputersociety.org/10.1109/PCCGA.2000.883956.
253. Zhang, H., & Hoff, K. E. III. (1997). Fast backface culling using normal masks. In *Proceedings of the 1997 symposium on interactive 3D graphics, I3D '97* (pp. 103-ff). New York: ACM. ISBN 0-89791-884-3. doi:10.1145/253284.253314. URL http://doi.acm.org/10.1145/253284.253314.
254. Zhang, H., Manocha, D., Hudson, T., & Hoff, K. E. III. (1997). Visibility culling using hierarchical occlusion maps. In *Proceedings of the 24th annual conference on computer graphics and interactive techniques, SIGGRAPH '97* (pp. 77–88). New York: ACM Press/Addison-Wesley Publishing Co. ISBN 0-89791-896-7. doi:10.1145/258734.258781.
255. Zhang, L., Kim, Y. J., & Manocha, D. (2007). A fast and practical algorithm for generalized penetration depth computation. In *Robotics: science and systems conference (RSS07)*.
256. Zhang, L., Kim, Y. J., & Manocha, D. (2007). C-dist: efficient distance computation for rigid and articulated models in configuration space. In *Proceedings of the*

2007 ACM symposium on solid and physical modeling, SPM '07 (pp. 159–169). New York: ACM. ISBN 978-1-59593-666-0. doi:10.1145/1236246.1236270. URL http://doi.acm.org/10.1145/1236246.1236270.

257. Zhang, L., Kim, Y. J., Varadhan, G., & Manocha, D. (2007). Generalized penetration depth computation. *Computer Aided Design*, *39*(8), 625–638. doi:10.1016/j.cad.2007.05.012.
258. Zhang, X., & Kim, Y. J. (2007). Interactive collision detection for deformable models using streaming aabbs. *IEEE Transactions on Visualization and Computer Graphics*, *13*(2), 318–329. doi:10.1109/TVCG.2007.42.
259. Zhang, X., Lee, M., & Kim, Y. J. (2006). Interactive continuous collision detection for non-convex polyhedra. *The Visual Computer*, *22*(9), 749–760. doi:10.1007/s00371-006-0060-0.
260. Zhang, X., Redon, S., Lee, M., & Kim, Y. J. (2007). Continuous collision detection for articulated models using Taylor models and temporal culling. In *ACM SIGGRAPH 2007 papers, SIGGRAPH '07*. New York: ACM. doi:10.1145/1275808.1276396. URL http://doi.acm.org/10.1145/1275808.1276396.
261. Zhu, X., Ding, H., & Tso, S. K. (2004). A pseudodistance function and its applications. *IEEE Transactions on Robotics*, *20*(2), 344–352.
262. Zilles, C. B., & Salisbury, J. K. (1995). A constraint-based god-object method for haptic display. In *Proceedings of the international conference on intelligent robots and systems, IROS '95* (Vol. 3, p. 3146). Washington: IEEE Computer Society. ISBN 0-8186-7108-4. URL http://dl.acm.org/citation.cfm?id=846238.849727.

http://www.springer.com/978-3-319-01019-9

New Geometric Data Structures for Collision Detection and Haptics

Weller, R.

2013, XVI, 240 p., Hardcover
ISBN: 978-3-319-01019-9