Collision detection: A survey

Conference Paper · November 2007

DOI: 10.1109/ICSMC.2007.4414258 · Source: IEEE Xplore

CITATIONS
96

READS
2,169

5 authors, including:

Sinan Kockara
University of Central Arkansas
38 PUBLICATIONS **343** CITATIONS
SEE PROFILE

Tansel Halic
University of Central Arkansas
59 PUBLICATIONS **401** CITATIONS
SEE PROFILE

Kamran Iqbal
University of Arkansas at Little Rock
136 PUBLICATIONS **1,369** CITATIONS
SEE PROFILE

Coskun Bayrak
Youngstown State University
129 PUBLICATIONS **648** CITATIONS
SEE PROFILE

Collision Detection: A Survey

S. Kockara, T. Halic, K. Iqbal, *Senior Member, IEEE*, C. Bayrak, and Richard Rowe

***Abstract*—A process of determining whether two or more bodies are making contact at one or more points is called collision detection or intersection detection. Collision detection is inseparable part of the computer graphics, surgical simulations, and robotics. There are varieties of methods for collision detection. We will review some of the most common ones. Algorithms for contact determination can be grouped into two general parts: broad-phase and narrow-phase. This paper provides a comprehensive classification of a collision detection literature into the two phases. Moreover, we have attempted to explain some of the existing algorithms which are not easy to interpret. Also, we have tried to keep sections self-explanatory without sacrificing depth of coverage.**

I. Introduction

WHEN given the two models and their placements in the world space, the simplest brute force approach to perform a collision query is to test each of the primitive segments in object A against each of the primitive segments of object B, requiring number of A's primitive segments times number of B's primitive segments overlap tests. We cannot perform exhaustive pair-wise testing on models which have thousands of primitives since a collision query needs to be performed in every simulation step in order to detect colliding objects. Animations can have many objects, all of which may have a complex geometry such as polygonal soups of several thousands facets. It is therefore computationally heavy burden to perform collision detection. Thus, to eliminate this computationally costly pair-wise tests some different algorithms proposed in the literature.

Hubbard [1] was the first who classified the collision detection in terms of broad-phase and narrow-phase. Those concepts of broad-phase and narrow phase collision detection reduce the computational load by performing a coarse test in order to prune an unnecessary pair test. Broad-phase collision detection identifies disjoint groups of possibly intersecting objects. On the contrary, pruning unnecessary primitive-pair test is narrow-phase collision detection. Most of the literature uses Hubbard's broad and narrow phase collision detection scheme to classify collision detection algorithms [2][3]. The same classification technique will also be used throughout this survey. Some of the methods such as bounding volumes are included in both broad and narrow-phase collision detection.

Manuscript received March 16, 2007. This work was supported in part by the University of Arkansas Medical Sciences.

Sinan Kockara is with University of Arkansas at Little Rock, Applied Science Dept., 2801 S. University Ave., Little Rock, Ar 72204 (phone: 501-6837154; e-mail: sxkockara@ualr.edu).

Tansel Halic is with University of Arkansas at Little Rock, Applied Science Dept., 2801 S. University Ave., Little Rock, Ar 72204 (e-mail: txhalic@ualr.edu).

Kamran Iqbal is with University of Arkansas at Little Rock, Systems Engineering Dept., 2801 S. University Ave., Little Rock, Ar 72204 (email: kxiqbal@ualr.edu).

Coskun Bayrak is with University of Arkansas at Little Rock, Computer Science Dept., 2801 S. University Ave., Little Rock, Ar 72204 (e-mail: cxbayrak@ualr.edu).

Richard Rowe is with University of Arkansas Medical Scieneces, Neurosurgery,, 4301 W. Markham, Little Rock, AR 72205 (e-mail: RoweRichard@uams.edu).

II. Broad-Phase Collision Detection

Broad-phase collision detection determines objects which should be tested with during the narrow-phase. Therefore, approximating objects with boxes makes broad-phase collision detection easier. To perform broad-phase collision detection, there are mainly three different kinds of algorithms: All-pair test (Exhaustive Search), sweep and prune (Coordinate Sorting), and hierarchical hash tables (multi-level grids).

An exhaustive search is a brute-force approach which compares each object's bounding volume with others' bounding volumes. If algorithm finds colliding bounding volumes then starts further investigation with narrow-phase collision detection algorithms. Sweep and prune algorithm [5][6] projects every object's bounding volume's starting and ending points onto the coordinate axes. If there is intersection among the entire principal coordinate axes, then there is collision between the objects. Hierarchical hash tables are another approach in broad-phase [7]. This approach divides the entire scene into the same size grids along all the principal axes. All points overlaps with the given grid cell is identified by the algorithm. If there is more than one object sharing the same cell, then those objects are possibly colliding objects.

III. Narrow-Phase Collision Detection

Broad-phase lists pairs of possible colliding objects and narrow-phase inspects further each of these pairs and finally contact determination algorithms determine the exact collisions. Narrow-phase algorithms usually return more detailed information. That information can be later used for the computation of time of impact, collision response and forces, and contact determination. Algorithms in this category can be categorized into the four groups: feature-based, simplex-based, volume-based, and spatial data structures [8].

A. Feature-based Algorithms

This kind of algorithms directly works on the geometric primitives of the objects. Well known examples are polygonal intersection [8], Lin-Canny [9], V-Clip[10], SWIFT[11][12]. Lin-Canny algorithm is the first feature-based algorithm in the literature. There are other feature tracking algorithms proposed based on Lin-Canny such as Voronoi-Clip (V-Clip)

and SWIFT. In real-time simulations, objects tend to change their orientations or rotations by small amounts from one frame to another (coherence). That coherence assumes that the closest points between two non-intersecting objects are located in the near vicinity of the closest points between the same objects located at the previous frame. However, for a polyhedron, even a minute change in orientation can cause a big change on closest points' locations between consecutive frames. Therefore, for polyhedra, Lin et al. [9] proposed using the closest features (vertices, edges, or faces) rather than tracking closest points from one frame to another. A drawback for Lin-Canny is that it does not terminate when presented with penetrating polyhedra.

Another feature based algorithm is V-Clip. It is based on the theorem which defines the closest points between two polyhedra in terms of the closest features of the pair of polyhedra. Figure 1. shows pair of 3D polyhedra satisfying the theorem. F(X) and F(Y) are closest pair of features and P(X) and P(Y) are closest pair of points (not necessarily unique points) between two polyhedra X and Y. Red lines indicate Voronoi region for object X and yellow lines indicate Voronoi region of Y. The V-Clip starts with two features one from X and another from Y. Feature F(X) is edge E and feature F(Y) is vertex V. If P(X) is in Voronoi region of Y and P(Y) is in the Voronoi region of X, the F(X) and F(Y) are closest pair of features. In each iteration, the features are tested to see if they are satisfying the conditions at the theorem. If they satisfy the theorem, algorithm terminates and returns nonintersecting between two polyhedra. If the theorem does not satisfy, one of the features is updated with a neighboring feature. Neighbors of a feature are defined as:

Definition: *The neighbors of a vertex are the edges incident to the vertex, the neighbors of a face are the edges bounding the face, and the neighbors of an edge are the two vertices and the two faces incident to the edge.*

The V-Clip becomes trapped in a local minimum in the vertex-face state where the vertex lies below the supporting plane of the face and at the same time lies inside all of the Voronoi planes of Voronoi region. That can cause objects penetrations before collision detection.

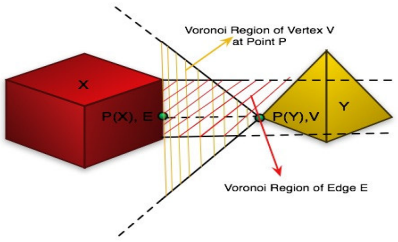

Figure 1. Closest pair of features and closest pair of points (P(X), P(Y)) for Vertex-Edge feature pair

B. *Simplex Based Algorithms*

The simplex is the convex hull of an affinely independent set of points. The GJK (Gilbert-Johnson-Keerthi) [26] is the well known ancestor of this group of algorithms [27]-[30]. GJK takes two sets of vertices as input and finds the Euclidean distance and closest points between the convex hulls. Thanks to Gilbert et al. [31], GJK was generalized to be applied to arbitrary convex point sets, not just to polyhedra. An important fact in GJK is that: it does not operate on the two input objects; however, operates on the Minkowski difference between the objects. Minkowski difference provides transformation of the problem from finding the distance between two convex sets to that of finding the distance between the origin and a single convex set. The GJK searches a sub-volume of the Minkowski difference object iteratively (each sub-volume being a simplex). We take a cue from the work of Ericson et al. [3] and clarify the GJK algorithm.

Let A and B two convex point sets and x and y two position vectors corresponding to pairs of points in A and B respectively. The Minkowski difference is defined as $A\Theta B = \{x - y : x \in A, y \in B\}$. The GJK algorithm based on the fact that separation distance between two convex polyhedra A and B is equal to the distance between Minkowski sum and the origin as shown in **Fig. 2**. [3] below.

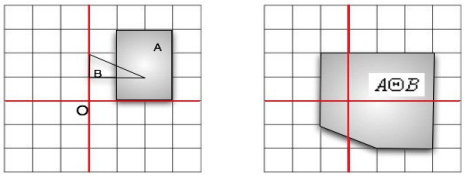

Fig. 2. Minkowski Difference

Two convex objects collide if and only if their Minkowski difference contains the origin. **Fig. 3**. [3] illustrates how GJK algorithm finds a point closest to origin *O*. In this case, the distance of closest point to the origin is equal to the minimum distance between two convex polyhedra due to the Minkowski difference.

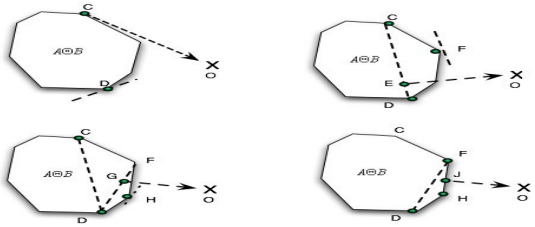

Fig. 3. GJK Algorithm

The algorithm arbitrarily begins with vertex C as the initial simplex set *Q={C}*. For a single-vertex simplex, vertex itself is the closest point to the origin *X*. Searching in the direction (from vertex C to the origin) results vertex D as a supporting point or extreme point at this direction. So, D is added to the simplex set *Q={C,D}*. The point in convex hull Q closest to the origin is now E. Since both C and D are needed to express point E we keep these vertices in the simplex *Q={C,D}*. Now F is the extreme point in the direction from E to the origin. That results new convex hull Q, *Q= {C,D,F}*. The closest point to the origin from convex hull Q is now point G. Since representing point G is possible with only D and F, C is removed from the simplex, *Q={D,F}*. Now supporting vector is from point G to the origin and new extreme point is H. H is added to the simplex Q, *Q={D,F,H}*. The point on Q closest to the origin is now J. Since F and H are smallest set of vertices to represent J, D is removed from Q, *Q={F,H}*. After this point,

there is no vertex closer to the origin in direction from J to the origin. Therefore, J must be the closest point to the origin and the algorithm terminates.

C. Image-Space Based Algorithms

Image space base (ISB) techniques are computed by image-space occlusion queries which are convenient to implement on the graphics hardware (GPU). Therefore, ISB techniques are recently more preferred. Opposite to common belief, they can also be employed on the CPU rather than GPU such as [13]. Occlusion queries have lower bandwidth than buffer read-backs and thus more convenient for GPU implementations [14]. Frontiers of ISB methods include [15]-[18] and [19]-[23]. All these methods have several common drawbacks. They are much slower than hierarchical approaches. They usually have O(n) complexity. During the rendering, objects are discretized to the image space which causes erroneous representations. These errors depend on the size of the viewport, the internal representation of numbers, and the number of bits per pixel in the z-buffer. Therefore, the size of the viewport has significant impact on the performance.

Cinder [21] is well known example of the ISB algorithms. It is based on 3D version of Jordan Curve Theorem **[4]**. It is a theorem in computational geometry that a semi infinite ray originating within a solid will intersect the boundary of the solid odd number of times as in **Fig. 4**. Cinder is handling both convex and non-convex geometries. The tests for collisions are performed in image space. The algorithm does not require any pre-processing or special data structures. It uses frame buffer operations to implement a virtual ray casting algorithm for every pixels that detect interference between objects. The edges of the objects are written to the depth buffer and the objects they penetrated each other are detected by using a virtual ray-casting algorithm. The number of polygons that the ray passes through is counted in such a way that if summation result for one ray is even, then the point is outside the object. In contrast, if the summation results un-even, then the point is inside the object and there is a collision. The algorithm uses a stencil buffer for counting the number of front and back facing polygons that the rays pass through. The values in the stencil buffer are increased for front-facing polygons and decreased for back-facing polygons. If at the end there is non-zero value in the stencil buffer, then edge in the specific pixel is inside an object. Colliding objects' identifications' numbers are kept in color-buffer. The algorithm's running time is linear in the number of objects and the number of polygons existing in the objects. With this algorithm, collisions that are about to happen or have already occurred will not be detected. This occurs when objects' very thin parts pass through each other in space of one frame. The object must be closed to get a correct result. The **Fig. 5**. shows an example of CINDER.

CULLIDE [24] uses occlusion queries and one of the prominent examples of ISB methods. The graphics hardware is used only to detect potentially colliding objects, while triangle-triangle intersections are performed in the CPU. CULLIDE uses clever but simple lemma to prune the non-colliding objects from possibly colliding objects' set. The lemma is: "An object A does not collide with a set of objects S, when A is fully visible with respect to S." CULLIDE keeps potentially colliding objects in a set which initially includes each and every object in the scene. Then it prunes the primitives from a potentially colliding set by rendering in a two-pass algorithm; first rendering front and then reverse order. Throughout the rendering, visibility (occlusion) queries remove objects from potentially colliding list if the object is not visible. This strategy continues iteratively until no more changes are made in potentially colliding set (PCS). The primitives in the final PCS are then made for exact collision detection. Boldt et al. [25] extended CULLIDE to handle self collision tests. Even though this approach alleviates Cinder's restrictions on object topology; CULLIDE's effectiveness degrades dramatically when the density of the environment increases.

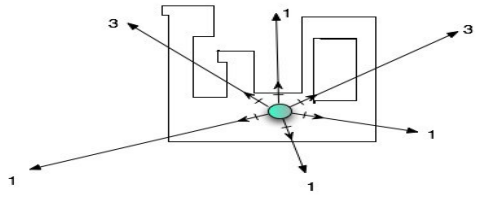

Fig. 4. Jordan Curve Theorem

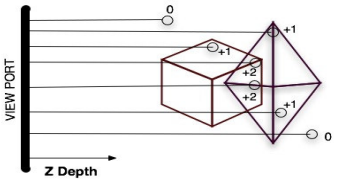

Fig. 5. Cinder with Virtual semi-infinite ray casting

D. Volume Based Algorithms

Most of the volume based algorithms conceptually based on the same idea of the ISB techniques; however, they use different methods to compute layered depth images [51] and distance fields etc. These groups of algorithms are also suitable for GPU implementations. Gundelman et al. [32] is one of the volume based collision detection algorithm which assumes that A and B are objects in the scene and we are searching whether they are colliding or not. This algorithm works by taking vertices of A and looks them up in the signed distance function of B. After that, the vertices of B are looked up in the signed distance function of A. Each object is represented by a triangular mesh and the signed distance map. Thus, both the triangular mesh and signed distance grid are stored in object space. This means that when vertices of A looked up in B, then they must be transformed from object space A to objects space B. The drawback of this algorithm is that it is not tailored for detecting edge-edge intersections.

E. Bounding Volume Hierarchies

There are two types of spatial data structures for collision detection: spatial division and bounding volume hierarchies (BVH). Spatial partitioning recursively divides the space. On

the other hand, BVH recursively or iteratively partitions the object itself. With spatial partitioning, splitting of polygons is unavoidable. This causes increase of depth of the tree and performance lost. In addition, since cell size of the spatial partitioning cannot cover objects' primitives tightly, when objects are close, determining contact status is difficult. On the contrary, BVs provide smaller and tighter hierarchies than spatial partitioning. In addition, BVHs are more applicable for general shapes than simplex based and feature based algorithms. BVH can be called as discrete representation of level of details of objects. At the first level, hierarchy includes one bounding volume which is very coarse representation of an object. Further levels include more detail representation of the object. The leaf level or finest level of the hierarchy generally includes the object primitives (lines, triangles, or tetrahedra). There is a parent-child relationship between succeeding levels with the topology of the tree. Bounding volume (BV) does not necessarily enclose its children's bounding volume; instead it must enclose the geometry of an object included in the children BVs.

Even though objects are not colliding, their BVs can collide. Therefore, we must look further down of the BVH to answer the question of whether objects are colliding or not. We do this by changing one of the root volumes by its children. Determining which root volume to descend is called traversal rule. Generally largest volume is chosen to descend to lower the chance of finding overlapping. If two volumes are equal then random choice is made. Non-overlapping BVs are discarded from further consideration (pruning). At last in the traversal, if we reach to two leaf nodes from two distinct volumes, then we have two choices; whether testing two primitives are colliding (pairwise test) or testing one primitive with the other's leaf bounding volume (primitive-volume test). If objects' primitives are colliding, we have to test two primitives anyway. Thus, there is a tradeoff between the number of iterations and the complexity in the overlap tests.

Gottschalk et al. [33] states that recursively traversing BVHs is often a bad choice since the number of primitives and the hierarchies can be quite large. Therefore, number of recursive calls would be huge; that causes memory stack overflows. This problem can be solved by using iterative traversal technique with first-in-first-out queue. The idea of using queue to escape from disadvantage of recursive nature of the algorithm is taken one step further by [34] and [35] with introducing a priority on the pairwise BV tests. This is useful for time critical collision detections. In this algorithm, all pairs of root passed from broad-phase are pushed to the queue and given a priority. Up to the certain threshold time, priority queue based traversal algorithm runs. When time is up, objects are determined as colliding if they are not pruned yet.

When we perform intersection tests between different objects, we need to bring those hierarchies to the common representation ground. For this purpose, there are two methods: the first is transforming one object's hierarchy to the other object's frame (model space update) which provides a performance and a fitting advantageous. The second is transforming both objects' hierarchy into the world coordinate system (WCS) which is costly.

With movement or rotations of an object, BVH needs to be realigned (refitting). Some BVHs do not need alignment such as spheres and oriented bounding boxes (OBB). Spheres have very good property that they are completely independent of the orientation. In addition, with Welzl algorithm presented in [42]-[44], finding better fitted spheres is easier and that makes spheres preferable over other topological BVs. Update of BVs to movements involves coordinate transformation and realignment. This problem can be solved by refitting the BV for the moved object. Notwithstanding, even though refitting provides tighter fit and early pruning capability, it is not preferred for computationally expensive simulations.

An ideal BVH preferred to be small size, to have small height, to have good pruning capabilities, and to be balanced. Small heights and balanced trees result performance gains. Balanced trees provide a good overall worst-case seeking complexity. Good pruning capabilities imply good performance of a collision query. BVs comparisons should have a capability of early rejection as much as possible. This early rejection criterion depends on the topology choice of the hierarchy (spheres, OBB, AABB etc.) and tighter fitting. In overall, there is a tradeoff between BVH complexity and performance [33]. Complex topology choices establish tighter fitted BVs and so fewer overlap tests but causes performance lost. On the other hand, less complex hierarchies provide faster overlap test but less tight BVs. Using a simpler geometry with cheaper overlap test is preferable for highly dynamic environments even though this will cause unnecessary overlap tests. The measurement of the tightness of the BV is presented by [36].

Usually balanced binary trees are wanted since they have good search properties. Zachmann et al. [37] proved that quad-trees and/or octrees significantly faster than binary trees for collision queries. Gundelman et al. [33] proved that OBBs are superior over spheres and AABBs for surface based BVHs. AABBs are preferred choice of deformable objects [29], [38] since AABBs are cost effective for frequent refitting operations in deformations. Thus, AABBs over perform OBBs even though OBBs have better pruning capabilities. However, for volume based methods, spheres are the most suitable choice [34], [39],[52], and [40]. To increase the pruning capability of the BVH, BV should be short and fat rather than being long and thin. The long and thing topology increases the chance of overlapping BV with the other BVs which causes performance lost. In this sense, spherical or cubic BVs will reduce possible overlapping with other BVs which implies a better chance for pruning. Klosowski et al. [41] introduces some criteria of partitioning of BVs to provide better pruning capabilities. He proposed to choose splitting plane orthogonal to local x, y, or z coordinate axis. This is very simple and fast approach. Other splitting axis methods are longest side method is another approach for finding splitting axis by selecting the axis along which BV is longest. Another simple method is using already existed axis of bounding volume as splitting axis. In this, case BV cannot be sphere since spheres do not have any associated axis. The last and the most accurate method is statistic based method which aligns the splitting axis along the axis where the covariance is

the largest. Subsequent to splitting axis determination split point needs to be chosen since there are infinitely many numbers of possible splitting points along the splitting axis. Again the choice for the point must be restricted to a small set of points with the following strategies: object median, object mean, spatial median, BV projection, and centroid coordinates. Object median method is splitting at the object's median where calculated from primitives' centers. Cormen et al. [45] reduced the cost of median finding by using a sophisticated method. Klosowski et al. [41] reports that using the object mean is superior to using the object median since resulting smaller volume trees with a lower number of operations performed and with a better query times. Splitting at the spatial median is another method which is equal to splitting the volume to two equal parts. In this technique, split point is found in constant time because this method only deal with the BV not the data included in the BV. This approach is used when the axis is selected from the parent volume such as used in longest side rule. BV projection method splits BV projection into evenly spaced points and instead of spending the time for intelligent point guessing, it spends the time for finding best splitting point among those evenly spaced points by using brute force methodology. The centroid coordinates method finds splitting point between random subset of the centroid coordinates. Fig. 6. [3] shows some of these methods.

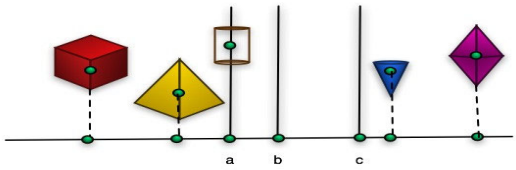

Fig. 6. Splitting at (a) the object median, (b) the object mean, (c) the spatial median

Deformable objects are very challenging for BVH. Deformations can make representation of the object's geometry with BVH useless. Since it is impossible to reconstruct the BVH for each time step from scratch for complex objects and deformations, it is necessary to find better approaches. [46]-[50], and [37] investigate faster approach to refit the currently misaligned BVH tree by using bottom-up update scheme. There are some drawbacks of these works. One is coming from the nature of the bottom-up scheme that update requires one to traverse the entire BVH. The second problem with the bottom-up update is that not all BV types can be updated very fast. The third problem with bottom-up scheme is that pruning capabilities of BV can be damaged during the update due to a possible increasing among sibling BVs. Instead of bottom-up scheme top-down scheme can be used in some certain situations; however, both have deficiencies at certain situations. Therefore, [38] proposed hybrid approach to benefit from both scheme advantageous. Recently, deformable spanners [53] have been proposed to encode all proximity information which may turn out to be very useful for deformable and rigid body collision detections. Deformable spanners are geometry independent (means that the geometry can be open, close, convex or concave) proximity queries with different resolutions that makes the deformable spanners very useful for surgical simulations. However, it may be difficult to get real time performance out of the deformable spanners without further algorithmic improvements.

Ultimate goal for collision detection researchers is development of new algorithms that handle rigid and deformable bodies, self collisions, convex and non-convex or open and close geometries and process all in real-time. To that end, there is still a long way to go.

IV. Conclusion

We have classified collision detection into two groups broad and narrow phase. Spatial partitioning and bounding volumes are well-known examples of the broad-phase. Feature based (FB), simplex based (SB), ISB, volume based, and bounding volume based algorithms are group of algorithms in narrow-phase. FB approaches only works for closed objects and it is not known how algorithm behaves to the degenerate conditions. Problem with FB algorithms is that it does not terminate when presented with penetrating polyhedra. SB methods are only for convex objects. ISB techniques require close objects in the scene and cannot detect self collisions. Also, collisions that are about to happen or have already occurred will not be detected in ISB. Therefore, FB, SB, and ISB are not considerable for dynamic simulations and deformations e.g. surgical simulations. Those methods do not allow open objects which occur during the surgical procedures such as cutting. With spatial partitioning, splitting of polygons is unavoidable and determination of cell size is very difficult. Therefore, when objects are close, determining contact status is difficult. BVHs are generally most suitable for collision detections but they are highly dependent to the topology choice of the hierarchy. There is a trade off between complexity of the topology of BV and construction cost. For deformable objects, BVs require different kind of update involving rebuilding, rebalancing, and refitting of BVH. There is still no well-informed hierarchy update scheme for deformations in that deformable spanners may introduce solutions.

References

[1] P. M. Hubbard, "Interactive Collision Detection", In Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality, 1993, pp.24-32.

[2] Physics-Based Animation, Kenny Erleben et al. Charles River Media, 2005

[3] Real-time Collision Detection, Christer Ericson, Morgan Kaufman, 2005.

[4] Computational Geometry, M. de Berg, 1998, Computational Geometry in C, O'Rourke, 2005.

[5] D. Baraff, A. Witkin, J. Anderson, and M. Kass, "Physical Based Modelling", SIGGRAPH Course Notes, 2003.

[6] M. C. Lin, and D. Manocha, "Efficient Contact Determination between Geometric Models", Technical Report TR94-024, The University of North Carolina at Chapel Hill, Dept. of Computer Science, 1994.

[7] B. Mirtich, "Impulse Based Dynamic Simulation of Rigid Body Systems", Phd. Thesis, University of California, Berkley, 1996.

[8] M. Moore, and J. Williams, "Collision Detection and Response for Computer Animation", In Computer Graphics, vol. 22, pp. 289-298, 1988.

[9] M. Lin and J. Canny, “A fast Algorithm for Incremental Distance Calculation”, Proc. of the 1991 IEEE International Conference on Robotics and Automation, pp. 1008-1014, 1991.
[10] B. Mirtich, “V-Clip: Fast and Robust Polyhedral Collision Detection”, ACM Transactions on Graphics, vol. 17(3), pp. 177-208, 1998.
[11] S. A. Ehmann, and M. Lin, “Accelerated Proximity Queries between Convex Polyhedra by Multi-level Voronoi Marching”, IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2000, vol. 3, pp. 2101-2106, 2000.
[12] S. A. Ehmann and M.C. Lin, “Swift: Accelerated Proximity Queries between Convex Polyhedra by Multi-level Voronoi Marching”, Technical Report, Computer Science Dept., University of North Carolina at Chapel Hill, http://www.cs.unc.edu/~geom/SWIFT/, 2000.
[13] B. Benes and N. G. Villanueva, “GI_COLLIDE- Collision Detection with Geometry Images”, In SCCG 2005, Proc. of the Spring Conference on Computer Graphics, pp.95-102, 2005
[14] oss.sgi.com/projects/ogl-example/registry/ARB/occlusion_query.txt, SGI 2005
[15] M. Shinya and M.C. Forgue, “Interference detection through rasterization”, Journal of Visualization and Computer Animation 2, pp. 132-134, 1991
[16] K. Myszkowski, O.G. Okunev, and T. L. Kunii, “Fast Collision Detection Between complex Solids using Rasterizing graphics Hardware”, Visual Comput. 11, pp. 497-512, 1995
[17] J.-C. Lombardo, M.-P. Cani, and F. Neyret, “Real-time Collision Detection for Virtual Surgery”, in proc. of Computer Animation, Geneva, Switzerland, pp. 82-90, 1999
[18] G. Baciu and W.S.-K. Wong, “Hardware Assisted self collision for deformable surfaces”, in Proc. of the ACM Symposium on Virtual Reality Software and Technology (VRST), ACM press, pp. 129-136, 2002
[19] G. Baciu and W.S.-K. Wong, “Image based techniques in a hybrid collision detector”, IEEE Trans. On Visualization and Computer Graphics 9, pp. 254-271, 2003
[20] K. E Hoff, A. Zaferakis, M. Lin, and D. Manocha, “Fast and Simple 2D Geometric Proximity Queries Using Graphics Hardware”, In Proc. Of ACM Symposium on Interactive 3D Graphics, pp. 145-148, 2001.
[21] D. Knott and D. Pai, “Cinder: Collision and Interference Detection in Real–time Using Graphics Hardware”, In Proc. of Graphics Interface ’03, 2003.
[22] B. Heidelberger, M. Teschner, and M. Gross, “Volumetric Collision Detection for Deformable Objects”, Technical Report #395, Computer Science Dept., ETH Zurich, 2003.
[23] B. Heidelberger, M. Teschner, and M. Gross, “Detection of Collisions and Self-collisions Using Image-space Technique”, In Proc. WSCG, pp. 145-152, Plzen, Czech Republic, 2004
[24] N. Govindraju, S. Redon, M. Lin, and D. Manocha, “CULLIDE: Interactive Collision Detection between Complex Models in Large Environments Using Graphics Hardware”, ACM SIGGRAPH/Eurographics Graphics Hardware, 2003.
[25] N. Boldt and J. Meyer, “Self-intersections with CULLIDE”, DIKU project # 04-02-19, the department of Computer Science at the University of Copenhagen, 2004.
[26] E. Gilbert, D. Johnson, and S. Keerthi, “A Fast Procedure for Computing the Distance Between Complex Objects in Three-dimensional Space”, IEEE Journal of Robotics and Automation, vol. 4, pp. 193-203, 1988.
[27] S. Cameron, “Enhancing GJK: Computing Minimum and Penetration Distances Between Convex Polyhedra”, IEEE Int. Conf. Robotics and Automation, vol. 4, pp. 3112-3117, Albuquerque, NM, USA,1997.
[28] G. v. d. Bergen, “A Fast and Robust GJK Implementation for Collision Detection of Convex Objects”, Journal of Graphics Tools, vol. 4(2), pp. 7-25, 1999.
[29] G. v. d. Bergen, “Proximity Queries and Penetration Depth Computation on 3D Game Objects”, Proc. Game Developers Conf., 2001.
[30] G. v. d. Bergen, “Collision Detection in Interactive 3D Environments”, Interactive 3D Technology Series, Morgan Kaufmann, 2003.
[31] E. Gilbert and Chek-P. Foo, “Computing the Distance between General Convex Objects in Three-dimensional Space”, IEEE Transactions on Robotics and Automation, vol. 6, no. 1, pp.53-61, 1990.
[32] E. Gundelman, R. Bridson, and R. Fedkiw, “Nonconvex Rigid Bodieswith Stacking”, ACM Transaction on Graphics, Proc. of ACM SIGGRAPH, 2003.
[33] S. Gottschalk, “Collision Queries Using Oriented Bounding Boxes”, Phd. Thesis, Dept. of Computer Science, University of N. Carolina at Chapel Hill, 2000.
[34] C. O’Sullivan, and J. Dinglina, “Real-time Collision Detection and Response Using Sphere-trees”, 1999.
[35] J. Dingliana and C. O’Sullivan, “Graceful Degradation of Collision Handling in Physically Based Animation”, Computer Graphics Forum Proc. Eurographics, vol. 19, no. 3 , pp 239-247, 2000.
[36] G. Zachmann and E. Langetepe, “Geometric Data Structures for Computer Graphics”, SIGGRAPH 2003 Course Notes, 2003.
[37] J. Mezger, S. Kimmerle, and O. Etzmuss, “Hierarchical Techniques in Collision Detection for Cloth Animation”, Journal of Winter School of Computer Graphics (WSCG) vol. 11, no. 2, pp. 322-329, 2003.
[38] T. Larsson and T. Akenine-Moller, “Collision detection for continuously deforming bodies”, in Proc. of Eurographics, pp. 325-333, 2001
[39] P. M. Hubbard, “Approximating Polyhedra with Spheres for Time-critical Collision Detection”, ACM Transactions on Graphics, vol. 15, no. 3, pp.179-210, 1996.
[40] G. Bradshaw and C. O’Sullivan, “Adaptive Medial-xis Approximation for Sphere-tree Construction”, ACM Transactions on Graphics, vol. 23, no. 1, pp. 1-26, 2004.
[41] J. T. Klosowski, “Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments”, PhD thesis, State Univ. of New York at Stony Brook, 1998.
[42] E. Welzl, “Smallest Enclosing Disks (balls and ellipsoids)”, In H. Maurer, editor, New Results and New Trends in Computer Science, LNCS, Springer 1991.
[43] M. D. Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, “Computational Geometry: Algorithms and Applications”, Springer-Verlag, 1997.
[44] K. Fischer and B. Gartner, “The Smallest Enclosing Ball of Balls: Combinatorial Structure and Algorithms”, Proc. of 19th Annual Symposium on Computational Geometry (SCG), pp. 291-301, 2003.
[45] T. Cormen, C. Leiserson, and R. Rivest, “Introduction to Algorithms”, MIT Press, 1990.
[46] G. v. d. Bergen, “Efficient Collision detection of Complex Deformable Models Using AABB Trees”1997,
[47] P. Volino, and N. M. Thalmann, “Collision and Self-collision Detection: Efficient and Robust Solutions for Highly Deformable Surfaces”, Technical Report, MIRALab, 1998.
[48] P. Volino and N. M. Thalmann, “Virtual Clothing: Theory and Practice”, Springer-Verlag Berlin Heidalbarg, 2000.
[49] R. Bridson, R. Fedkiw, and J. Anderson, “Robust Treatment of Collisions, Contact and Friction for Cloth Animation”, Proc. of ACM SIGGRAPH, vol. 21, no. 3, pp. 594-603, 2002.
[50] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure, N. Magnetat-Thalmann, W. Strasser, "Collision Detection for Deformable Objects", Eurographics State-of-the-Art Report (EG-STAR), 2004, pp 119-139.
[51] B. Heidelberger, M. Teschner, and M. Gross, "Detection of collisions and self-collisions using image-space techniques". Journal of WSCG, vol. 12, no. 1-3, 2004.
[52] T. Larsson and T. Akenine-Moller , “A dynamic bounding volume hierarchy for generalized collision detection”, Computers & Graphics, vol 30, no 3, pp 451-460, Elsevier Ltd, 2006.
[53] J. Gao, L. J. Guibas, and A. Nguyen, “Deformable spanners and applications”, Computational Geometry: theory and applications, vol. 35, issue 1, pp. 2-19, 2006.