Abstraction & Application **5** (2011) 7-18 UADY

# An Introduction to the Collision Detection Algorithms

Francisco Madera

Facultad de Matemáticas, UADY

mramirez@uady.mx

**Abstract**

The problem of determining the close proximity of objects in a virtual environment has been investigated in several fields such as computational geometry, CAD, robotics and computer graphics. Collision detection algorithms focus on the geometric intersections among objects in the scene, performing different queries such as detecting objects in close proximity or finding exact points of contact. The rapid evolution of collision detection algorithms is due to the demand for advanced simulations, in addition to the significant developments of graphics hardware. Recent algorithms consider the combination of traditional methods in a continuous interval, and sometimes take advantage of graphics hardware. The state of the art of the collision detection process is presented, discussing several object representations, the different phases in the process, and the factors of an efficient algorithm.

## 1 Introduction

The widespread use of animation has resulted in a strong demand for plausible and believable collision detection. A major part of such animations will always be the physical model of the world, typically constructed from the laws of physics and solved numerically. Whilst some collision detection algorithms take advantage of the motion and deformations, others assume no knowledge of the ensuing motions and deformations of the objects involved.

The contact points between two objects can usually be approximated at a small number of locations on a subset of the object's primitives. When testing a virtual environment comprised of many objects, an efficient algorithm should first cull away whole objects which are deemed too distant to be colliding and then cull away distant triangles contained within two objects.

With regard to objects and motions, the collision detection algorithm addresses its design on specific tools. Avatars, cloth, fluids, rigid bodies have different characteristics and therefore require different methods to compute the collisions. Forward and inverse kinematics, the Finite Element Method, wave equations, and rigid body dynamics present specific properties that can be exploited in order to obtain an optimum algorithm. While some approaches are focused on speed, others consider the accuracy as the main issue. Nevertheless, the performance of the algorithms can be determined regarding the complexity and the difficulty of implementation.

The contribution of this paper is the discussion of the research into collision detection which has taken place in the last five years, since a previous collision detection survey [1] was published. The properties of the

1998 *ACM Computing Classification System.* I.3.5 [*Computer Graphics*] *Computational Geometry and Object Modeling.*
*Keywords and phrases* : Computationl Geometry, Computer Graphics, Collision Detection.

different approaches are investigated with the advantages and disadvantages being detailed to aid comparison. At a broad level, the field of collision detection can be categorised depending on the nature of the input models (rigid versus deformable, linear versus curved, convex versus concave or surface versus volumetric), the existence of motion (static versus dynamic), the type of collision query (discrete or continuous), and the type of computing resources that the collision query utilises (CPUs versus GPUs).

## 2 The Problem

Collision detection is the problem of checking for possible interferences between geometric models moving in space. Given an environment of $n$ objects $\{O_1, O_2, ..., O_n\}$, each represented as a collection of triangles, $O_i = \{t_1^i, t_2^i, ...\}$, the objective is to find out which objects overlap, $O_i \cap O_j \neq \emptyset$ for $i \neq j$, and compute the overlapping triangles, $t_\alpha^i \cap t_\beta^j \neq \emptyset$.

Irrespective of the animation method, collision detection is a problem where the object representation is mapped into geometric information in order to test its proximity with respect to other objects. However, in some applications specific algorithms exploit information provided by the deformation model such as [2] which uses linear blend skinning to detect collisions in skeletally deformable models, and [3] where the vertex positions are convex combinations of sets of reference meshes to detect collisions between models deformed by mesh morphing.

Design factors when constructing a collision detection algorithm involve the geometric representation, different types of queries, the relationship with the animation engine, the ease of implementation and use. Collision detection algorithms strive for fast and accurate results. To reduce the computation, several areas of the algorithm can be improved to cull primitives away. For deformable objects the data structures employed may also need to be recomputed to ensure efficient intersection queries are still possible even after the geometry has deformed. The accuracy is determined by the contacts of the collision, such as the exact pair of primitives in collision.

Different types of queries can be considered, with the simplest being to determine if two objects are colliding. For more detail, information concerning the parts of these objects in collision is required, whilst at other times the separation distance is enough. For some kinds of objects and applications, the penetration distance or the minimum translational distance is required to keep them apart. Finally, for each animation frame, a task to determine the closest objects and the closest primitives between such objects should be performed. To reach this goal, several methods are available in the literature as can be seen in the following sections.

## 3 The Shape Representation

Earlier work on collision detection focused on the study of basic primitives, such as spheres, boxes and cylinders. Determining the intersection among these basic primitives is cheap to compute, and complex objects can be created from such primitives using, for example, Constructive Solid Geometry [4]. Furthermore, these geometries can be decomposed into small pieces such as points, lines, and planes to allow detailed analysis to take place. Mostly, interference between objects is detected in a discrete manner, synchronised with the time steps of the animation. This is in contrast to the continuous approach which considers a time interval instead [5].

To reduce the number of polygons in the input models, some algorithms employ multiresolution techniques [6, 7], that aim to approximate the shape of the objects with lower resolution versions. This is an interesting idea, but care must be taken when handling several levels of resolution because of the cost of computation and the memory consumption which can cause inefficient performance. Otaduy and Lin [8] implemented a data structure that serves both as a BVH (Bounding Volume Hierarchy) for accelerating collision queries and as a multiresolution representation of the original model for computing contact information. This approach was tested in a rigid body simulation.

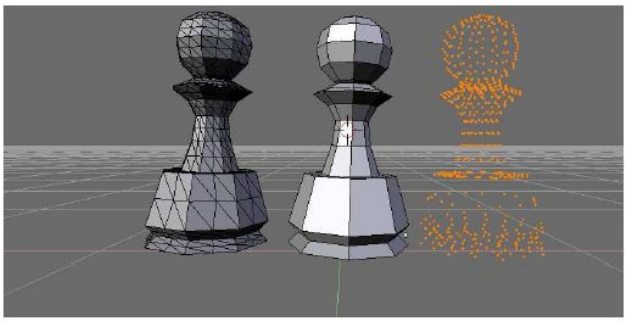

Figure 1: Three types of geometric representation of a pawn. From left to right: triangles, quads, and points.

A point-based representation focuses on the vertices of the objects, alleviating the need to store the mesh connectivity. Klein and Zachmann [9] presented an approach for time-critical collision detection of point clouds. Based solely on the point representation, it can detect intersections of the underlying implicit surfaces. They construct a point hierarchy where each node stores a sufficient sample of the points in addition to a sphere covering part of the surface. Spillmann et al. [10] employed a point-based object representation which undergoes geometric deformation. The geometry was bounded in a sphere hierarchy. Keiser et al. [11] showed a collision detection and response algorithm for point-based animation. The point-based surface is deformed during collisions and forces are exerted on the points representing the volume yielding plausible simulations.

When a collision response is required, some objects are represented with inner meshes using tetrahedra or hexahedra to simulate a range of material effects including elasticity, plasticity, melting and fracture [12, 13]. The most interesting simulations involve pairs of objects that touch each other at multiple points of collision. When the collision is detected between two objects, another check is performed using the normals of the contact points to determine the directions of the objects' movements relative to each other. In Figure 1 a pawn is represented by three methods, triangle mesh, quad mesh, and points.

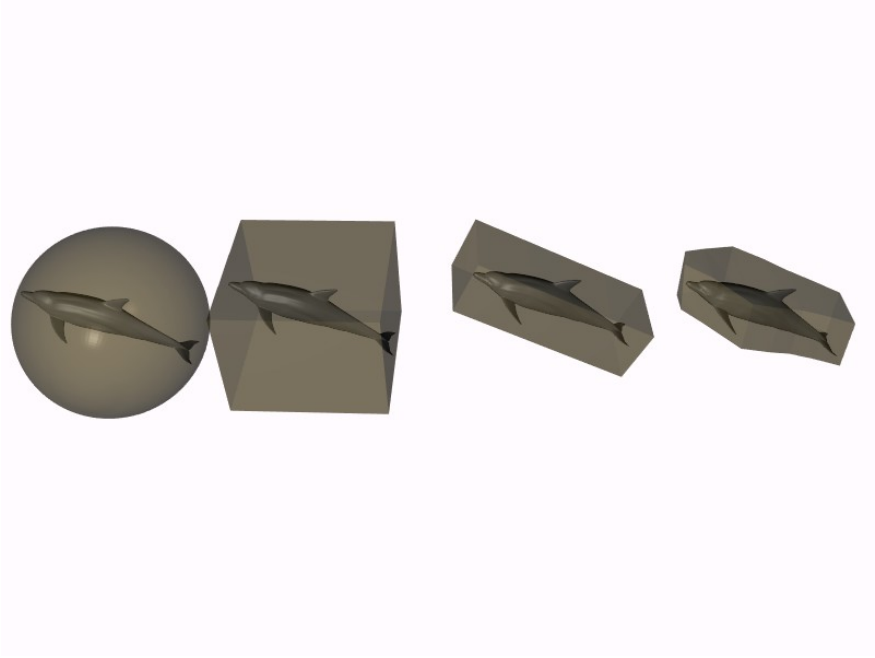

Figure 2: A dolphin object enclosed by different types of bounding volumes: sphere, AABB, OBB, convex hull.

# 4 Collision Detection Methods

Valuable surveys detailing collision detection algorithms have been presented. Lin et al. [14] classify the methods based on the type of the geometric model used, while Jiménez et al. [15] consider their interference process. Teschner et al. [16] surveyed collision detection for deformable objects where five methods are described and applications are shown.

A variety of methods from computational geometry have been adopted by the computer graphics community. Firstly, efforts were made for rigid bodies in robotics [17, 18, 19] and computational geometry [20, 21], in particular employing convex objects. Later, algorithms were considered for deformable models, including the common collision detection methods such as bounding volume hierarchies, spatial partitioning, distance fields, and GPU-based techniques.

**Bounding Volume Hierarchy (BVH)**. Efficient collision detection algorithms are accelerated by spatial data structures such as the BVH or by spatial partitioning. Such object representations are commonly built in a pre-processing stage and perform very well for rigid and deformable objects. The use of BVHs as a way to represent virtual objects in a scene dates back to 1976 when Clark [22] suggested an algorithm in a hierarchical fashion to render objects quickly. However, time-consuming pre-processing is required to build these structures, rendering them unsuitable for dynamic settings. Bounding volumes are utilised in many applications because of their ability to represent the shape of objects and due to the reduced cost of testing against a BV rather than comparing against the object itself. Spheres have been used in a wide range of applications since they are easy to represent, have a fast overlap, and are rotationally invariant [10, 23, 24]. The AABB (Axis Aligned Bounding Box) also has a fast overlap check, which is accomplished via a simple comparison of its coordinate values [25, 26, 27]. The Oriented Bounding Box (OBB) can bound the object tighter than the AABB because it is oriented to best align with the underlying geometry [28, 29], however, it does require a more expensive overlap test. Other volumes are discrete oriented polytopes, sphere-swept volumes, and convex hulls. In Fig. 2 four bounding volumes are employed to enclose a 3D dolphin.

**Spatial Partitioning**. The spatial partitioning algorithms are independent of changes to the topology of the objects they contain. Rather than divide the object into regions, the spatial partitioning approach divides the scene into regions and it tests if objects overlap the same region of space. There are three types of structures; grids, trees, and spatial sorting. Grids are often uniform, but a problem appears when they are large because of the memory requirements. To reduce the memory requirements, hashed storage can be used to allow the grid to be of infinite extent.

Dynamic objects can be located in a cell by tracing the tree using a top-down approach, obtaining more accuracy when the leaves are reached. This approach is suitable to cull away parts of the scene which are empty, however, when accuracy is required another strategy should be utilised to locate the specific region in the object. Figure 3 illustrates the spatial partitioning method by subdividing a scene into ten cells.

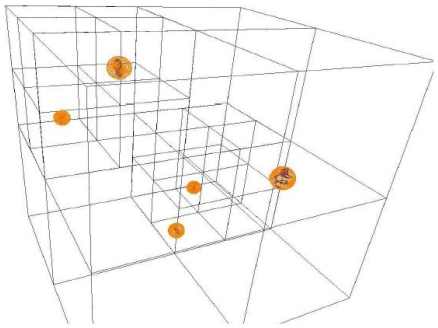

Figure 3: The scene partitioned with an octree contains five objects moving around.

**Feature-based algorithms**. Feature-based algorithms, which as the name suggests, work directly on the features of the objects. For polygonal meshes, these features are the vertices, edges, and faces. A well known collision detection algorithm based on the tracking of closest features is the Voronoi-Clip algorithm

[30], which operates on a pair of polyhedra and tracks the closest features. Three representative algorithms that work with rigid convex polyhedra are as follows: firstly, a feature-based incremental method that walks on the boundary of polyhedra [30, 31]. Secondly, the Dobkin-Kirkpatrick algorithm [32] starts from the innermost layers of hierarchies and tracks the closest feature-pair from layer to layer. Thirdly, the H-Walks [33] which starts on the boundaries of polyhedra, walks on the same layers for a few steps, and then drops to inner layers to take shortcuts.

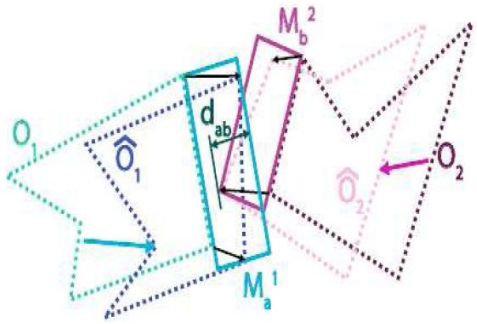


Figure 4: Continuous Collision Detection for two polygons $O_1$ and $O_2$: Two polygons, $O_1$ and $O_2$, move to positions $\widehat{O_1}$ and $\widehat{O_2}$ at time $t+\Delta t$. The volume swept by a pair of features is bounded by the prisms,$M_a^1$ and $M_b^2$, respectively. A conservative CCD (Continuous Collision Detection) check is performed by volumetric collision detection between $M_a^1$ and $M_b^2$. $d_{ab}$ is the signed distance between the prisms. The N-body distance query is used to compute the signed distance functions for all the prisms [34].

The feature-based method is used in the contact determination phase to calculate the exact points of contact from the geometric features ( illustrated in Figure 4 ). To reduce the number of tests between the features of the triangulated meshes, Curtis et al. [35] introduced the concept of Representative-Triangles (R-Triangles) which are standard triangles, augmented with mesh feature information.

**GPU-based algorithms**. Collision detection algorithms based on GPUs can be classified into two different categories: image space [36, 37] and object space [38, 39] approaches. The former approach exploits the powerful rasterization capability available in modern GPUs to perform intersection tests between object primitives in image space. The effectiveness of the approach is often limited by the image space resolution. The latter approach utilises the high floating point bandwidth and programmability of GPUs and all the computations are performed in object space and, thus, are limited by the floating point precision of GPUs.

Fig. 5 illustrates a Layer Depth Image method requirign three stages. Stage 1 computes the AABB intersection for a pair of objects, Stage 2 computes two LDIs, one for each object, and Stage 3 performs the actual collision detection. Also, the GPUs can be utilised as a co-processor of the CPU. This meehod is called GPGPU (General Purpose GPU) and the most important tool to construct programs is the CUDA (Computer Unified Device Arhitecture).

**Other Methods**. Most of the collision detection algorithms utilise two or more of the methods previously presented in the quest for good performance. Certainly, it is difficult to classify the algorithms in one of the methods metioned above.

Most evolving physical systems follow known physical laws, it is also frequently the case that collision events occur that alter the motion of one or more of the objects. Because of such discrete events, algorithms for modelling motion must be able to adapt in a dynamic way. The occurrence of these events can be handled with Kinetic Data Structures (KDS) [41, 42]. These often form a collection of simple geometric relations that certify the combinatorial structure of the attribute, as well as a set of rules for repairing the attribute and its certifying relations when one relation fails.

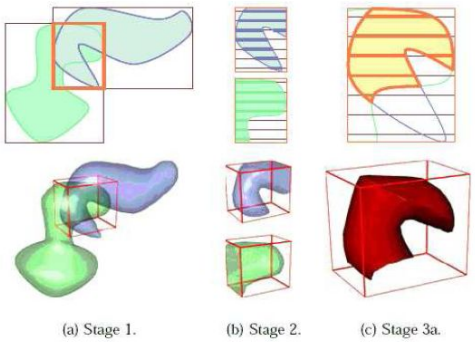


Figure 5: Algorithm overview in 2D and 3D. (a) AABB intersection. (b) LDI (Layer Depth Image) generation within the Volume Image. (c) Computation of the intersection volume[40]

## 5 Self-Collisions

Some models suffer large deformations that cause self-collisions. The mesh is deformed in such a way that polygons can touch other polygons during deformation. There are many 3D objects with this property: snake-like objects (Fig. 6), humans (Fig. 7) and some organs used in surgical simulators. Common 2D-objects used in 3D-simulations are cloth (Fig. 8) and hair since they are easily deformed in several directions and therefore self-collisions occur frequently.

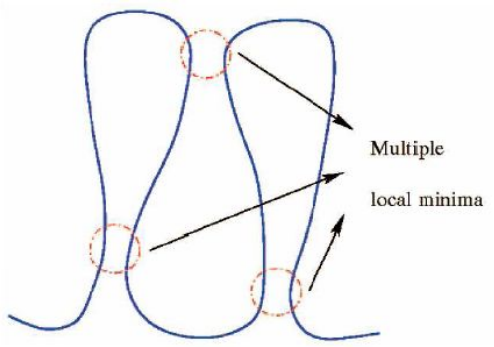


Figure 6: Tracking the local minima of distance between non-neighbouring segments [49].

Volino et al. [44] developed an algorithm to detect self-collisions in cloth, stating the curvature criteria as a tool to determine the existence of collisions. The normals are used in a hierarchy to determine the parent's normal and to decide whether further operations between the regions should be considered. Provot [45] detects self-collisions in cloth by applying another curvature criterion. When a given zone has a sufficiently low curvature, it cannot self-intersect and all the zones included do not intersect with each other. The curvature of a zone is evaluated by the set of normals of the triangle belonging to the zone. Mezger et al. [46] worked with object hierarchies for collision detection for cloth models utilising k-DOPs (K-Discrete Oriented Polytope). They also showed how normal cones can be incorporated into k-DOP hierarchies and they represented movements using velocity cones. The use of the angles formed between the polygons gives information about the nearness among the features of the mesh; the apex angle $\alpha$ of the cone represents the curvature of the region, indicating self-proximity intersections if $\alpha \geq \epsilon$ for $\epsilon \leq \pi$.

Computing self-collisions is a more expensive process than detecting collisions between objects, since all the polygons or regions in a mesh must be checked. Taking advantage of the renderization process to detect overlapped regions, GPUs are often used to render parts of the mesh from different ponts of view without a preprocessing phase. Some applications require the self-collision and collision detection processes to be run at the same time to determine all the contacts in the scene.

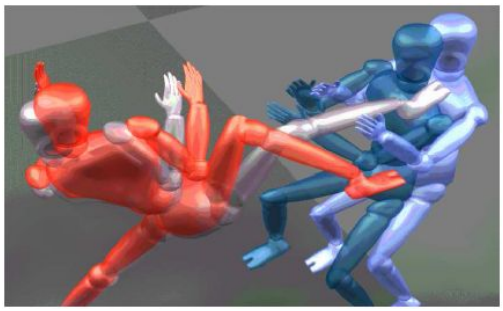

Figure 7: Initial (dark red and blue) and final (light red and blue) configurations of two moving mannequin models consisting of 15 links and 20K triangles each [48].

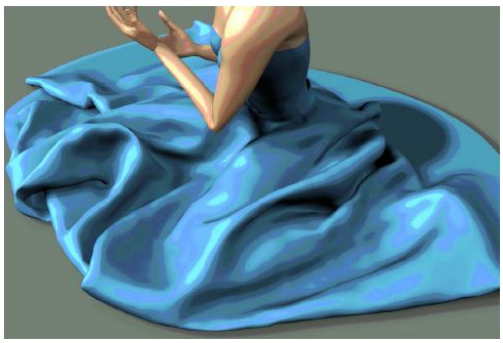

Figure 8: This simulation generates complex folds and wrinkles on the skirt. The cloth is modelled as a mesh with 13K triangles. The collision detection algorithm accurately checks for all self-collisions within 400-500ms during each step of the simulation [43].

# 6 Design Issues

Aspects affecting the performance of the collision detection algorithms are described in this section. Regarding several components of the simulation environment, the CD (Collision Detection) algorithm can be focused on specific types of strategies to obtain a better efficiency during the running time. The object's representation helps to define how to track the mesh or the points involved. The object's motion is considered to determine if the time steps represent the real movement or if an interpolation method is neccesary to obtain continuous information.

## 6.1 Rigid and Deformable Models

Rigid bodies are manipulated in an environment via affine transformations whilst deformable models change their shape by applying a transformation to a subset of the primitives. From the geometric point of view, rigid bodies are described by position, orientation and their derivatives. Deformable models can be described as a set of particles where the mapping function applied is different for each set of particles.

In the object's representation, data structures are the same in both rigid and deformable objects, but some methods employ additional data structures to hold some data about the deformation. As deformable models vary their vertices many times, more memory consumption is used in the processing of the collision detection algorithm. This causes an increase in time which depends on the number of vertices moved in a time step.

## 6.2 Continuous Approach

Most of the work in CD has focused on discrete algorithms, which check for interferences at fixed time instants only. In such cases it is possible to miss a collision between two successive instances. The continuous approach presupposes a specified motion of bodies over some time interval [5, 47]. A major issue in the design of such algorithms is modelling the continuous motion between the two successive positions and orientations of the object. It is important that the motion formulation is generic enough to interpolate any two given instances of an object, as well as simple enough so that it can be frequently and efficiently evaluated by the underlying CCD algorithm. Furthermore, it is relatively more expensive to check for collisions along a continuous path as opposed to a discrete instance.

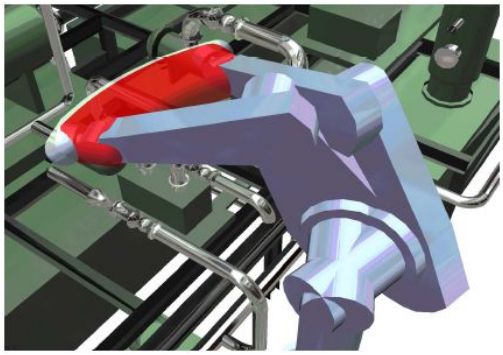

Figure 9: Dynamic SV culling based on graphics hardware (i.e., CULLIDE) applied to the last two links of a Puma robot model. The volume swept by the LSS which bounds the last link does not collide with the environment, and thus the link is culled away [50].

## 6.3 Objects Partitioning

Objects can change their shape by deformation or by decomposition. The decomposition of objects into sub-objects can be caused by cuts or breaks. This property should be dealt with in a particular way to preserve the essence of the original objects. The ability to cut the objects is fundamental to many real applications, especially virtual surgery. This problem has a deep impact in both modelling the physics of the object and in detecting the possible collisions, because it requires discarding the assumption that at least the description of the surface never changes.

In surgical interventions, cutting is one of the most important tasks. The correct and precise execution of intersections is the basis for a successful surgical outcome. Figure. 10 presents the opening of skin tissue with a surgical scalpel, which is a task in open surgery. In this example, Bielser et al. [51] simulated the skin tissue as a pre-stressed deformable model.

Figure 10: Simulation of open surgery [51].

# 7 Conclusions

The problem of detecting collisions between objects has been significantly well researched, often motivated by the demand for improved performance. While some researchers have focused on the computational efficiency, others focus on the accuracy of the results. In some cases this research has led to the release of open source software libraries and tools, enabling other researchers to develop further contributions to the field. Benchmarking of the collision detection algorithms developed is a particularly challenging endeavour due to the varied factors which can significantly affect the performance of the algorithms. The types of objects involved and their relative transformations and sizes can all influence the computation times of queries. Therefore, the type of application is one of the determining factors in choosing an appropriate collision detection strategy. In this paper the main collision detection algorithms have been described.

One approach which performs well in general is to decompose the scene into separate spatial areas to enable an efficient strategy for reducing the problem. However, this strategy typically requires a pre-processing stage and efficient techniques for tracking dynamic objects. A bounding volume hierarchy may be constructed for each three dimensional object to facilitate the acceleration of proximity queries between each object pair. Determining intersections between bounding volumes such as spheres, axis-aligned bounding boxes and oriented bounding boxes is a relatively inexpensive procedure when compared to testing the individual polygons which might be contained within them. To determine the exact points of contact and surface normals, the individual polygons within the bounding volumes forming the leaves of the hierarchy may be considered.

An open question stated in [52] concerns how the best return from the speed-accuracy trade-off can be obtained. The potential future of collision detection research could entail the further use of continuous collisions and the exploitation of parallel architectures. The ever increasing performance of GPUs is ensuring that they are becoming a powerful tool not just for rendering but for other uses such as collision detection. However, care must be taken to ensure the balance between the CPU and GPU is achieved to prevent undesirable delays on either processor.

# References

[1] M. Teschner and S. Kimmerle and B. Heidelberger and G. Zachmann and L. Raghupathi and A. Fuhrmann and M.P.P Cani and F. Faure and N. Magnenat-Thalmann and W. Strasser and P. Volino. *Collision Detection for Deformable Objects*. Computer Graphics Forum, vol. 24, No. 1, pp. 61-81, 2005.

[2] Ladislav Kavan and Jiri Zara, *Fast Collision Detection for Skeletally Deformable Models*. Computer Graphics Forum, vol. 24, No. 3, pp. 363-372, 2005.

[3] Thomas Larsson, Tomas Akenine-Möller. *Efficient collision detection for models deformed by morphing*. The Visual Computer, vol. 19, No. 2, pp 164 - 174.

[4] Aristides G. Requicha. *Representations for Rigid Solids: Theory, Methods, and Systems*. ACM Computing Surveys, vol. 12, No. 4, pp 437-464, 1980.

[5] Naga K. Govindaraju and Ilknur Kabul and Ming C. Lin and Dinesh Manocha. *Fast continuous collision detection among deformable models using graphics processorss*. Computer Graphics, vol. 31, No. 1, pp 5-14, 2005.

[6] F. Ganovelli. *Multiresolution and Cuts in Deformable Objects Modelling*. PhD Thesis, Pisa University, Italy, Computer Science Department, 2000.

[7] C. Mendoza and C. O'Sullivan. *Interruptible collision detection for deformable objects*. Computers & Graphics, vol. 9, No. 3, pp 432-438, 2006.

[8] Miguel A. Otaduy and Ming C. Lin. *CLODs: dual hierarchies for multiresolution collision detection*. Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing, Aire-la-Ville, Switzerland, pp 94-101, 2003.

[9] J. Klein and G. Zachmann. *Point cloud collision detection*. Computer Graphics Forum , vol. 23, No. 3, pp 567-576, 2004.

[10] Jonas Spillmann and Markus Becker and Matthias Teschner. *Efficient Updates of Bounding Sphere Hierarchies for Geometrically Deformable Models*. Journal of Vis. Comun., vol. 18, No. 2, pp 1047-3203, 2007.

[11] Richard Keiser and Matthias Müller and Bruno Heidelberger and Matthias Teschner and Markus H. Gross. *Contact Handling for Deformable Point-Based Objects*. Proc. VMV., pp 315-322, 2004.

[12] Matthias Müller and Markus Gross. *Interactive virtual materials*. Proceedings of Graphics Interface, Ontario, Canada, pp 239-246, 2004.

[13] M. Müller and J. Dorsey and L. McMillan and R. Jagnow. *Real-Time Simulation of Deformation and Fracture of Stiff Materials*. Eurographics Workshop on Animation and Simulation, 2001.

[14] Ming C. Lin and S. Gottschalk. *Collision detection between geometric models: A survey*. Proc. of IMA Conference on Mathematics of Surfaces, pp 25-32, 1998.

[15] P. Jiménez and F. Thomas and C. Torras. *3D Collision Detection: A Survey*. Journal of Computers and Graphics, vol. 25, No. 2, pp 269-285, 2001.

[16] M. Teschner and S. Kimmerle and B. Heidelberger and G. Zachmann and L. Raghupathi and A. Fuhrmann and M.P.P Cani and F. Faure and N. Magnenat-Thalmann and W. Strasser and P. Volino. *Collision Detection for Deformable Objects*. Computer Graphics Forum, vol. 24, No. 1, pp 61-81, 2005.

[17] S. A. Cameron and R. K. Culley. *Determining the minimum translational distance between two convex polyhedra*. IEEE International Conference on Robotics and Automation, San Francisco, USA, pp 591-596, 1986.

[18] J. Canny. *Collision Detection for Moving Polyhedra*. IEEE Transactions on PAMI, vol. 8, No. 8, pp 200-209, 1984.

[19] Ming C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD Thesis, University of California, Berkeley, 1993.

[20] B. Chazelle and D. P. Dobkin, *Intersection of convex objects in two and three dimensions*. Journal of the ACM, JACM, vol. 34, No. 1, pp. 1-27, 1987.

[21] David P. Dobkin and David G. Kirkpatrick, *A Linear Algorithm for Determining the Separation of Convex Polyhedra*. Tech. Report, University of British Columbia, 1983.

[22] James H. Clark, *Hierarchical geometric models for visible surface algorithms*. Communications of the ACM, vol. 19, No. 10, pp. 547-554, 1976.

[23] G. Bradshaw and C. O'Sullivan, *Adaptive Medial Axis Approximation for Sphere-Tree Construction*. ACM Transactions on Graphics, vol. 23, No. 1, pp. 1-26, 2004.

[24] D. James and D. Pai, *BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models*. ACM Transactions on Graphics , vol. 23, No. 3, 2004.

[25] G. van den Bergen, *Efficient Collision Detection of Complex Deformable Models using AABB Trees*. Journal of Graphics Tools , vol. 2, No. 4, pp. 1-14, 1997.

[26] Thomas Larsson and Tomas Akenine-Möller, *Collision Detection for Continuously Deforming Bodies*. Eurographics, Manchester, UK, pp. 325-333, 2001.

[27] Denis Steinemann and Miguel A. Otaduy and Markus Gross, *Special Section: Point-Based Graphics: Tight and efficient surface bounds in meshless animation*. Computer and Graphics, Manchester, vol. 32, No. 2, pp. 235-245, 2008.

[28] S. Gottschalk and M.C. Lin and D. Manocha, *OBB-Tree: A Hierarchical Structure for Rapid Interference Detection*. Proceedings on SIGGRAPH, New York, pp. 171-180, 1996.

[29] Konstantinos Moustakas and Dimitrios Tzovaras, *Fellow-Michael Gerassimos Strintzis*. Computer and Graphics, Manchester, vol. 13, No. 1, pp. 80-93, 2007.

[30] Brian Mirtich, *V-Clip: fast and robust polyhedral collision detection*. ACM Trans. Graph., vol. 17, No. 3, pp. 177-208, 1998.

[31] M. C. Lin and J. F. Canny, *A Fast Algorithm for Incremental Distance Calculation*. IEEE International Conference on Robotics and Automation, pp. 1008-1014, 1991.

[32] David P. Dobkin and David G. Kirkpatrick, *Determining the separation of preprocessed polyhedra: a unified approach*. Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming, Warwick University, England, pp. 400-414, 1990.

[33] L. J. Guibas and D. Hsu and L. Zhang, *H-Walk : Hierarchical Distance Computation for Moving Convex Bodies*. PSCG '99: Proceedings of the 15th Annual Symposium on Computational Geometry, Miami, Florida, USA, pp. 265-273, 1999.

[34] Avneesh Sud and Naga Govindaraju and Russell Gayle and Ilknur Kabul and Dinesh Manocha, *Fast proximity computation among deformable models using discrete Voronoi diagrams*. ACM Trans. Graph, vol. 25, No. 3, pp. 1144-1153, 2006.

[35] Sean Curtis and Rasmus Tamstorf and Dinesh Manocha, *Fast collision detection for deformable models using representative-triangles*. I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games, Redwood City, California, pp. 61–69, 2008.

[36] G. Baciau and W. Wong, *Image-Based Collision Detection for Deformable Cloth Models*. IEEE Transactions on Visualization and Computer Graphics, vol. 10, No. 6, pp. 649-663, 2004.

[37] Naga K. Govindaraju and Sthepane Redon and Ming C. Lin, *CULLIDE: Interactive Collision Detection Between Complex Models in Large Environments using Graphics Hardware*. EUROGRAPHICS 03, Germany, 2003.

[38] Alexander Gress and Gabriel Zachmann, *Object-Space Interference Detection on Programmable Graphics Hardware*. SIAM Conf. on Geometric Design and Computing, Seattle, Washington, pp. 311-328, 2003.

[39] Yoo-Joo Choi and Young J. Kim and Myoung-Hee Kim, *self-CD: Interactive Self-collision Detection for Deformable Body Simulation Using GPUs*. AsiaSiM, pp. 187-196, 2004.

[40] B. Heidelberger and M. Teschner and M. Gross, *Volumetric Collision Detection for Deformable Objects*. Proc. VMV 03, pp. 461– 468, 2003.

[41] Leonidas J. Guibas, *Kinetic data structures: a state of the art report*. WAFR '98: Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective, Houston, Texas, pp. 191–209, 1998

[42] L. Guibas and L. Feng and L. Zhang, *Kinetic collision detection: algorithms and experiments*. Robotics and Automation, 2001. Proceedings 2001 ICRA, MASSACHUSETTS, USA, pp. 2903- 2910, 2001

[43] Naga K. Govindaraju and David Knott and Nitin Jain and Ilknur Kabul and Rasmus Tamstorf and Russell Gayle and Ming C. Lin and Dinesh Manocha, *Interactive collision detection between deformable models using chromatic decomposition*. RACM Trans. Graph. vol. 24, no. 6, pp. 991–999, 2005

[44] Pascal Volino and Nadia Magnenat-Thalmann, *Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity*. EUROGRAPHICS, Germany, 1994.

[45] Xavier Provot, *Collision and self-collision handling in cloth model dedicated to design garments*. Proc. Graphics Interface , pp. 177- 189, 1997.

[46] Johannes Mezger and Stefan Kimmerle and Olaf Etzmuß, *Hierarchical Techniques in Collision Detection for Cloth Animation*. Journal of WSCG, vol. 11, No. 2, pp. 322–329, 2003.

[47] Wingo Sai-Keung Wong and George Baciau, *Robust continuous collision detection for interactive deformable surfaces*. Computer Animation and Virtual Worlds, vol. 18. No. 3, pp. 179-192, 2007.

[48] Xinyu Zhang and Stephane Redon and Minkyoung Lee and Young J. Kim, *Continuous Collision Detection for Articulated Models using Taylor Models and Temporal Culling*. ACM Transactions on Graphics , vol. 26, No. 3,2007.

[49] Laks Raghupathi and Laurent Grisoni and Francois Faure and Damien Marchal and Marie-Paule Cani and Christophe Chaillou, *An Intestinal Surgery Simulator: Real-Time Collision Processing and Visualization*. IEEE Transactions on Visualization and Computer Graphics , vol. 10, No. 6, pp. 708-718, 2004.

[50] S. Redon and Y. Kim and M.C. Lin and D. Manocha and J. Templeman, *Interactive and Continuous Collision Detection for Avatars in Virtual Environments*. IIEEE Virtual Reality Conference , pp. 117–130, 2004.

[51] D. Bielser and P. Glardon and M. Teschner and M. Gross, *A state machine for real-time cutting of tetrahedral meshes*. IIEEE Virtual Reality Conference, Graphical Models, vol. 66. No. 6, pp. 398–417, 2004.

[52] C. O'Sullivan and J. Dingliana, *Real-time Collision Detection and Response Using Sphere-trees*. 15th Spring Conference on Computer Graphics, Budmerice, Slovakia, pp. 83–92, 1999.