EG UK Theory and Practice of Computer Graphics (2007)
Ik Soo Lim, David Duce (Editors)

A Distance Hierarchy to Detect Collisions Between Deformable Objects

F. A. Madera A. M. Day S. D. Laycock

University of East Anglia, School of Computing Sciences
Norwich NR4 7TJ, UK

Abstract

To detect collisions between deformable objects we introduce an algorithm that computes the closest distances between certain feature points defined in their meshes. The strategy is to divide the objects into regions and to define a representative vertex that serves to compute the distance to the regions of the other objects. Having obtained the closest regions between two objects, we proceed to explore these regions by expanding them and detecting the closest sub-regions. We handle a hierarchy of regions and distances where the first level contains n_1 regions, each one is divided into n_2 sub-regions, and so on. A collision is obtained when the distance between two vertices in the last level of the tree is less than a predefined value ε. The advantage of our algorithm is that we can follow the deformation of the surface with the representative vertices defined in the hierarchy.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling

1. Introduction

Real time interaction in virtual environments for computer graphics programs requires rapid and accurate collision detection algorithms. Important areas demanding this process are clothes and surgery simulation, animation, and computer games. In addition to the object's motion, which is basically translation and rotation, a collision detection algorithm is required to follow the deformation in the objects in order to accurately compute distances between them.

This work describes an algorithm to detect collisions between deformable objects by computing the Euclidean distance between the regions defined in the objects. The algorithm exploits Bounding Volume Hierarchy (BVH) and feature-based methods to obtain a simple and fast approach to detect collisions. Due to the intersection tests involved, only one parameter is required to be updated per region during deformation.

The objects are divided into clusters without using a simplification method, that is, the original mesh is held. The object's representations, the trees, are compared in the dynamic simulation to detect their proximity to each other. Even when the object is deforming, representative points follow the surface because they are part of the mesh. Unlike the BVH in which the enclosing primitive should be refitted, our proposal only updates an ε value to guarantee that all the vertices of the cluster are considered.

The algorithm proceeds in three stages. First, in the precomputation stage the cluster hierarchy is constructed by using an octree. Alongside this operation, the representative vertices are created, which are the central vertices of the regions. Other techniques to divide objects are the Voronoi regions used by Mirtich [Mir98], and the clustered hierarchy of progressive meshes used by Yoon et al. [YSLM04].

The second and third stages take place in the running of the simulation. The second part involves the comparison of the nodes in each level of the tree, using the breadth-first-search traversal algorithm. The nodes of level h are expanded if the distance between them is less than ε_h. This ε_h is the greatest distance between the representative vertices, namely the middle vertex and the other vertices in a cluster.

The third stage of the algorithm commences when the leaves of the tree are reached. At this point the distance comparison takes place between the vertices of the regions involved and not between the middle vertices. This procedure

works by looking for the closest distances among the adjacent vertices of the current vertex. Furthermore we could use a contact determination routine to obtain the exact collision in one of the six different contact types as shown in [ESHD05].

The contributions are described as follows: A collision detection method which works with deformable models and follows the mesh deformation. The algorithm works for rigid and deformable bodies, and allows quad and triangular meshes. The most employed computation is the Euclidean distance, a cheap operation. The method can be adapted to achieve more levels of the tree and different values of ε to have faster computations. The algorithm detects collisions in several regions at the same time, it can be utilised in scenes with many complex objects and it is independent of the physical model.

This paper is organised as follows: previous work is described in the next section, Section 3 gives details about the three parts of the method. Section 4 shows the experiments made to test our algorithm, and finally, in Section 5 conclusions and future work are presented.

2. Previous work

Many algorithms have arisen to detect collisions between deformable models, and a recent survey with details about the most important methods can be found in [TKH*05]. Besides the motion considered in rigid bodies, the difficult task in deformable models is to calculate contacts between the objects as they change shape.

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 bounding volume in comparison to testing against the object itself. Spheres have been used in a wide range of applications since they are easier to represent and are rotationally invariant [SBT06, BO04, JP04]. The advantage of working with an Axis Aligned Bounding Box (AABB) is its fast overlap check, which is a simple comparison of its coordinate values [vdB97, LAM01]. An Oriented Bounding Box (OBB) is an AABB with an arbitrary orientation, enabling it to enclose the underlying geometry tightly. However, it does require a more expensive overlap test [GLM96, GLGT98]. Other volumes are Discrete Oriented Polytopes [KHM*98, FF03], Sphere-Swept Volumes [LGLM00, RKL*04], and Convex Hulls [PLM95, EL01].

Feature-based algorithms work directly on the features of an object. For polygonal meshes, the features are the vertices, edges, and faces of the meshes. A well known collision detection algorithm based on the tracking of closest features is the Voronoi-Clip algorithm [Mir98], which operates on a pair of polyhedra and tracks the closest features. Our method does not need to find the initial closest pair of vertices between two objects, and it works on non-convex deformable objects. 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 [Mir98, LC91]. Secondly, the Dobkin-Kirkpatrick algorithm [DK90] starts from the innermost layers of hierarchies and tracks the closest feature-pair from layer to layer. Thirdly, the H-Walks [GHZ99] 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.

Agarwal et al. [ABHZ02] presented an algorithm to fill the space between closest objects in 2D by using pseudo-triangulations, as a tool to detect collisions. This approach is inefficient in 3D because of the number of concave areas presented that would produce slow performance to test objects. However, this is an interesting method to calculate how far apart the other objects are. We consider the distances created in our method as a filling of the free space, indicating the closest regions of the objects and then tracking motion and deformation in every time step.

Some algorithms use a simplification method to decrease the level of detail of the objects, in order to reduce the number of primitives on which the collision detection process is working. Mendoza et al. [MO06] produced multiple approximations of the object's surface using a sphere hierarchy, and Guéziec [Gue01] handled a refinement process, using increasingly accurate closest points to compute distances. Since we are working with the features of the objects, our method does not need to simplify the models to speed up the simulation.

3. The Algorithm

Our approach is motivated by the need to achieve a simple and fast algorithm to detect collisions between deformable objects. Our proposal was inspired by the work of others; from the BVH [JP04] we use the hierarchies to cull away farther vertices and to reduce the number of elements to work with. From the H-Walk [GHZ99] we have a similar algorithm to walk around the surface by using the mesh connectivity, and from the work of Argawal et al. [ABHZ02] we explored the idea of filling the free space to test for close proximity.

The Euclidean distance between two points x and y, $\delta(x,y)$, is our simple but powerful tool to compute the close proximity between objects; this operation requires 3 additions and 3 multiplications to obtain its square value. This distance carries more information than the detection of a collision because it permits prediction, use of coherence, and dynamic motion modification. Let $P = \{O_1, ..., O_n\}$ be a set of n objects in a scene, where object i, O_i, is represented by a triangular mesh, whose features are faces, edges, and vertices. Let F_i be the number of faces, E_i be the number of edges, and V_i be the number of vertices. The approach calculates a collision between two objects using these features, in particular the vertices; and returns the closest pair

of vertices and faces between them. Let $O_i, O_j \in P$ be two objects in motion and $v_i \in O_i$ and $v_j \in O_j$ the closest points between them. We say that O_i and O_j collide with each other if $\delta(v_i, v_j) < \varepsilon(\approx 0^+)$.

3.1. Part I. Preparing the objects

In the first part of the algorithm, the objects' mesh is divided into regions as shown in line 1 of the code segment in Figure 1. Let $\bar{v}^k_{i,j}$ be the representative vertex of a region $R^k_{i,j}$, called the middle vertex of region j, level i, object k. Both convex and concave regions are admissible. The middle vertex is a vertex of the mesh, closest to the average of the vertices of the region and it is calculated in line 3 of the pseudocode as follows.

$$\bar{v}^k_{i,j} = min\{\ \delta(avg(v), v)\ \}, \quad \forall\ v \in R^k_{i,j}. \tag{1}$$

Subdivide ($R^k_{i,j}$)
0. If Num. vertices in $R^k_{i+1,j} > 30$
1. Divide R in 8 equal-sized regions $R^k_{i+1,j}$
2. $\forall\ R^k_{i+1,j}$
3. Compute $\bar{v}^k_{i+1,j}$
4. Compute $\varepsilon^k_{i+1,j}$
5. Subdivide ($R^k_{i+1,j}$)
6. Compute $\varepsilon^k_{i+1} = max\{\varepsilon^k_{i+1,j}\}$
7. Else return

Figure 1: *Pseudocode of the Subdivision algorithm.*

We also need an ε, obtained in line 4, to indicate that an external point is close to a region.

$$\varepsilon^k_{i,j} = max\{\ \delta(\bar{v}^k_{i,j}, v)\ \} \quad \forall\ v \in R^k_{i,j} \tag{2}$$

To be truly effective we generalise a global ε to represent a level, as specified in line 6 of the pseudocode.

$$\varepsilon^k_h = max\{\varepsilon^k_{h,j}\}. \tag{3}$$

Even when the vertices are moving in a region, ε indicates that they are kept within εs threshold. This can be shown by constructing a sphere with centre $\bar{v}$ and radius ε as pictured in Figure 2. Although the moving points can cause $\bar{v}$ to move out of the centre of the region, the condition is satisfied, because the vertices are still in the volume of the sphere, and we do not need to update values. Line 5 follows the subdivision hierarchy, each region $R^k_{1,j}$ of object k in level 1 is divided into eight regions $R^k_{2,j}$ for the second level. This first part of the algorithm takes $O(dlog_d(V_i))$ time for an object i, where $d = 8$; and requires $V_i + (1 - 8^L)/(1 - 8)$ nodes in its tree. Another subdivision method can be used to partition the objects and compute their middle vertices, such as the multiresolution hierarchy using filtered edge collapse utilised by Otaduy et al. [LO05] and the Hierarchical mesh decomposition using fuzzy clustering developed by Katz and Tal [KT03].

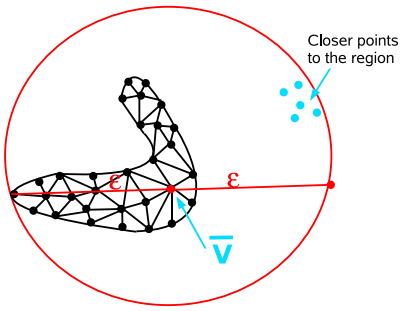

Figure 2: ε *is the greatest distance from the middle vertex to a vertex of a region.*

3.2. Part II. Detecting the closest regions between two objects

Having constructed the subdivision for the objects, we now have a tree for each, with L levels. The second part of the algorithm starts when the running simulation takes place. Since collision detection works between a pair of objects, we should compare the distances between the middle vertices of both objects to decide whether to descend to the next level of the tree for a greater level of detail.

$$Expand \quad R^{k_0}_{i,j}, R^{k_1}_{i,j} \iff \delta(\bar{v}^{k0}_{i,j}, \bar{v}^{k1}_{i,j}) < \varepsilon_i. \tag{4}$$

As a consequence, we could have more than one pair of regions expanded in a level, depending on the motion and deformation of the objects. The traversal method used is the breadth-first search. The maximum number of pairs created in level 1 is 64, but in practice this does not occur, we usually create less than $C_1 = 30$ per level so that the checking in the second level is $64C_1$. The number of checks in the next level is $64C_2$, and so on. In this sense, the time complexity in the second part of the algorithm is $O(C)$ where C is the maximum number of distances created in a level. This second part, that we call *DMV* (*Distance Middle Vertices*), is applied to all levels until the leaves of the tree are reached at which the third part starts. The pseudocode of the *DMV*

DMV (O_{k_0}, O_{k_1})
1. $\forall$ level h expanded in the tree
2. if h is the last level then compute DV($R^{K_0}_h, R^{K_1}_h$)
3. else if $\delta(\bar{v}^{k_0}_h, \bar{v}^{k_1}_h) < \varepsilon_h$
4. expand the child nodes

Figure 3: *Pseudocode of the DMV algorithm.*

algorithm, shown in Figure 3, takes two objects as inputs and works in a loop to track the levels of the tree. Line 2 states that if the current level is the last one, then the *DV* (*Distance Vertices*) algorithm is called, otherwise there is a comparison, in line 3, between the distances formed with the

middle vertices of object k_0 and the middle vertices of object k_1. If this distance is less than ε_h then the children should be expanded. We define two main data structures, *numcd* which has information about the pair of objects, and *cd* for the distance to be created for each pair. The first data structure is tested every time step to determine how far apart the objects are and it needs $n(n-1)/2$ elements for n objects. Alongside this structure, we handle specific information relating to each pair of objects in another data structure, where the distances are stored. In this structure, called *cd*, we keep dynamic information, because the number of distances depends on the proximity, that comes from the object's motion. Figure 4 illustrates two objects, where the distances of the regions are compared. Two pairs of regions are chosen to be expanded for the next level of detail.

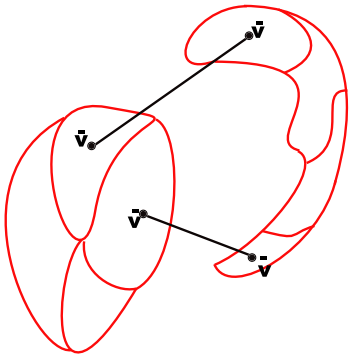

Figure 4: *The algorithm DMV working between two objects.*

3.3. Part III. Returning the closest pair of features between two objects

The third part of the algorithm starts when the process gets to the last level of the tree. Note that in a region $R^k_{i,j}$, there is only one middle vertex $\bar{v}^k_{i,j}$, while in the last level there are several vertices $v^k_{i,j}$, usually less than 32. This algorithm, that we call *DV*, starts with a distance δ^{k_0,k_1}_L in the last level L, and tracks the adjacent vertices in order to get the closest distance. The adjacent vertices of v are its neighbouring vertices, that is, they share a common edge with v. This tracking takes $O(Q)$ time, where Q is the number of neighbouring vertices, normally less than 10. An improvement is made by starting with the closest distance of both regions.

$$\delta^{k_0,k_1}_{i,j} = min\{\ \delta(v^{k0}_{i,j}, v^{k1}_{i,j})\ \}. \tag{5}$$

The *DV* algorithm is called from the *DMV* algorithm, and it is employed to compute the closest distances between the vertices of two regions. The algorithm starts with a pair of vertices which are part of a distance, and computes the distances of their neighbouring vertices in order to get the smallest distance. We do not need additional data structures for this operation, since it can work with the original mesh connectivity. The pseudocode of the *DV* algorithm is illustrated in Figure 5. Contrary to [MDL06b] we do not need to compute the volumes of tetrahedra to fill the free space, instead we compute the close proximity between objects.

DV ($R^{k_0}_{h,i}, R^{k_1}_{h,j}$)
1. *Get the closest pair of vertices to start with:* v_i, v_j
2. $\forall$ *neighbours of* v_i *and* v_j
3. *if* $\delta\ (Neighb(v_i), Neighb(v_j))\ ) < \varepsilon_h$
4. *Return this pair:* $(Neighb(v_i),\ Neighb(v_j))$

Figure 5: *Pseudocode of the DV algorithm.*

The input data for the *DV* algorithm are the regions of both objects, and it begins by computing the closest vertices. There is a cycle in line 2 for the neighbours of the current vertices to make comparisons between their distances. When a distance is less than $\varepsilon_L \approx 0$ then the result is the closest pair of vertices in both objects. In Figure 6, the *DV* algorithm is running between two objects, walking along their surfaces as they move.

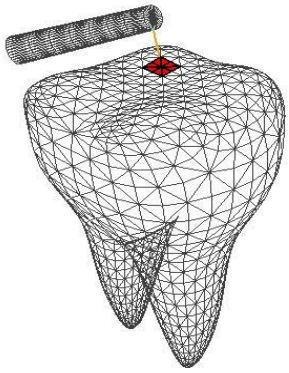

Figure 6: *The algorithm DV working over two objects.*

4. Experiments

We tested our algorithm on five models whose features are described in Table 1. Three different experiments were carried out to investigate the performance characteristics of our proposed method. The program was implemented in C++, using a 3 GHz Pentium 4 CPU, 1 GB RAM. The objects are animated using a physically-based method, the Mass-Spring model described in [MDL06a].

In our first scene, 10 objects are animated during 300 steps, choosing two models, the torus and the pawn. According to eqn. (3) we compute ε to define the closest points to

Object	Vertices	Faces	Edges
Torus	768	1536	2304
Pawn	610	1216	1824
Human	946	1888	40
Cylinder	258	512	768
Tooth	1202	2400	3600

Table 1: *Features of example models.*

a region for each level of hierarchy. The number of collisions increases as objects become closer, and the simulation is stopped to avoid penetrations. The experiment was repeated five times with different ε values shown in Table 2, and the number of collisions reported is illustrated in Figure 9.

Parameters	Level 1	Level 2	Level 3
1	20	14	8
2	22	14	8
3	24	14	8
4	30	14	8
5	30	20	10

Table 2: *Initial* ε *values (integer units) in each level for the first experiment.*

The first two runs show less collisions detected because of a small ε chosen in the first level, which could not detect some closer regions, having an algorithm timing in the range of 30 ms and 37 ms per time step. The two last runs report more collisions than there should have, because of a large ε, having an algorithm timing in the range of 30 ms and 45 ms. Even when the ε value satisfies eqn. (3), there are some regions in which this value can be adjusted because they can not be expanded further, otherwise the object would lose its shape. The optimal value was obtained in the third run, where all the collisions were detected with the minimum number of nodes expanded, being 76 for level 1, 288 for level 2, and 151 for level 3.

Figure 7 shows two objects employed in the first experiment, with the hierarchical lines indicating how close they are. Red, green, and blue lines represent the first, second, and third level respectively. These lines are created in this order, and when objects are being separated, lines are removed in reverse order. In the second experiment we used three humans and one torus, deforming during 200 steps. We chose a human because it is a difficult object with extremities far apart from the central body, which makes it more difficult to compute the ε. This object can be decomposed in its extremities (head and central body) in order to have more control over the regions. The torus is also a complex object, because it has a hole in its centre which needs to be checked for collisions.

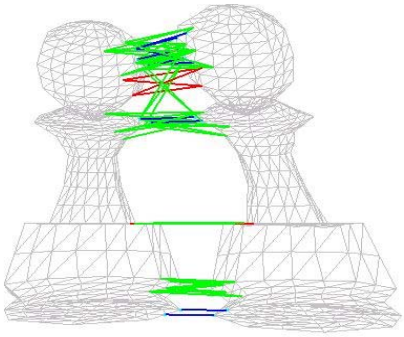

Figure 7: *Two pawns in the close proximity. Red, green, and blue lines represent levels* 1,2, *and* 3 *respectively.*

Parameters	Level 1	Level 2	Level 3
1	20,25	15,20	10,10
2	45,35	30,30	10,10
3	25,35	15,30	10,10
4	40,40	25,30	10,10
5	25,45	15,30	10,10

Table 3: *Initial* ε *values (integer units) in each level for the second experiment. The notation* a,b *indicate the values for the human-human collision and for the human-torus collision.*

In the previous experiment the objects were similar in size, so that we could handle the same values for each level. In this second experiment the values were different and we employed an ε value for each type of collision, human-human and human-torus. From here, we have that the value for a pair in level 1, of objects 0 and 1 is

$$\varepsilon_1 = \varepsilon_1^0 + \varepsilon_1^1. \tag{6}$$

The experiment was repeated five times with different parameters, and the number of collisions reported are illustrated in Figure 10 where the optimal value of ε is represented in the third run, and the algorithm takes about 25 ms per time step. The simulation is pictured in Figure 8. Note that the number of collisions reported using the parameters 2 and 3 are the same, that is where the optimal value is specified. However, the difference lies in the number of expanded nodes, where the third parameter set is the optimal with a lot less nodes expanded, that is 5 ms time less than the second run per frame.

The final experiment used a rigid tooth and a deformable cylinder, running in 100 steps. These objects were chosen to investigate the algorithm's behaviour with convex surfaces.

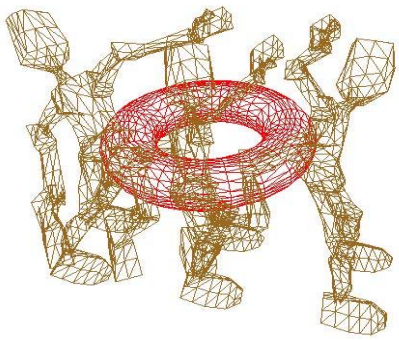

Figure 8: *The second experiment, three humans and a torus deforming.*

The cylinder is a convex object and the tooth is formed by four convex surfaces on its top. The number of collisions detected, illustrated in Figure 11, are almost the same even when the ε values are varied (see Table 4). This is because the algorithm works on convex regions and not because we are dealing with a pair of objects. The optimal value took place in the third set, with 199 collisions reported in 24 ms.

Parameters	Level 1	Level 2	Level 3
1	20	15	10
2	25	12	8
3	25	13	10
4	25	15	8
5	30	15	10

Table 4: *Initial* ε *(integer units) values for the third experiment.*

There are several factors influencing the performance of the algorithm: the number of objects, the objects' starting locations, the object's complexity, and the deformations applied. The most expensive operations are in the deformation routines because a physically-based method is required to calculate ordinary differential equations every time step. The closest features can be calculated easily from the mesh connectivity since the closest vertices were known when the object is colliding.

5. Conclusions

This paper presents an algorithm, which computes the distance between *n* deformable bodies to obtain the closest regions between them. This is made by employing a hierarchical representation of the objects and defining representative vertices to be used in the distance computation. The algorithm is capable of following the shape of the objects during deformation, exploiting the mesh connectivity and the region's hierarchy.

The algorithm utilises the efficient Euclidean distance to test collisions between objects by traversing the trees using the breadth-first-search method. Since we are returning pairs of features, a contact determination method can be used to obtain more detail about the collision. Rather than approximating the geometry of the objects, our proposal exploits the mesh connectivity and the neighbouring features of the object in order to detect collisions between deformable models. While the sphere bounding volume requires two parameters to be defined and updated in the running simulation, a 3D vector for the centre and a float for the radius, this novel method requires two floats to represent the centre and the radius of the region. Since the centre or the middle vertex of the region was computed in the preprocessing stage, the only parameter to be updated is the radius when object is deforming.

From observation the experiments bring an optimal ε to be defined to reduce the number of distances. One drawback is the accuracy of the algorithm but this is expected to be improved by defining the minimum regions in the object, the basic primitives to be dealt with. A future objective involves a comparison between other methods focusing on the intersection test, tight fitting, computation time, and the use of memory; in order to show the comparative performance of the algorithm proposed.

6. Acknowledgments

We wish to thank to the Universidad Autonoma de Yucatan and the Mexican program PROMEP for the funding of madera's PhD program. We are also grateful to the reviewers for their feedback.

References

[ABHZ02] AGARWAL P., BASCH J., HERSHBERGER J., ZHANG L.: Deformable free-space tilings for kinetic collision detection. *International Journal of Robotics Research 21*, 3 (2002), 179–198.

[BO04] BRADSHAW G., O'SULLIVAN C.: Adaptive medial axis approximation for sphere-tree construction. *ACM Transactions on Graphics 23*, 1 (2004), 1–26.

[DK90] DOBKIN D. P., KIRKPATRICK D. G.: Determining the separation of preprocessed polyhedra: a unified approach. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming* (New York, NY, USA, 1990), Springer-Verlag New York, Inc., pp. 400–413.

[EL01] EHMANN S., LIN M.: Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum 20* (2001), 319–328.

[ESHD05] ERLEBEN K., SPORRING J., HENRIKISEN K., DOHLMANN H.: *Physics-Based Animation*. Charles River Media Publisher, San Francisco, 2005.

[FF03] FUNFZIG C., FELLNER D. W.: *Easy Realignment of k-DOP Bounding Volumes*. Tech. rep., University of Technology Muhlenpfordtstr, Institute of Computer Graphics, 2003.

[GHZ99] GUIBAS L. J., HSU D., ZHANG L.: H-walk : Hierarchical distance computation for moving convex bodies. In *Symposium on Computational Geometry* (1999), pp. 265–273.

[GLGT98] GREGORY A., LIN M., GOTTSCHALK S., TAYLOR R.: *H-COLLIDE: A framework for fast and accurate collision detection for haptic interaction*. Tech. Rep. TR98-032, University of North Caroline, Chapell Hill, 1998.

[GLM96] GOTTSCHALK S., LIN M., MANOCHA D.: Obb-tree: A hierarchical structure for rapid interference detection. In *Proceedings on SIGGRAPH 96* (New York, 1996), ACM, pp. 171–180.

[Gue01] GUEZIEC A.: 'meshsweeper': Dynamic point-to-polygonal-mesh distance and applications. *IEEE Transactions on Visualization and Computer Graphics 07*, 1 (2001), 47–61.

[JP04] JAMES D., PAI D.: Bd-tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (SIGGRAPH 2004) 23*, 3 (2004).

[KHM*98] KLOSOWSKI J. T., HELD M., MITCHELL J. S. B., SOWIZRAL H., ZIKAN K.: Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (1998), 21–36.

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts, 2003.

[LAM01] LARSSON T., AKENINE-MÖLLER T.: Collision detection for continuously deforming bodies. In *Eurographics 2001, Short Presentations* (Manchester, September 2001), Eurographics Association, pp. 325–333.

[LC91] LIN M. C., CANNY J. F.: A fast algorithm for incremental distance calculation. In *IEEE International Conference on Robotics and Automation* (1991), pp. 1008–1014.

[LGLM00] LARSEN E., GOTTSCHALK S., LIN M., MANOCHA D.: Fast distance queries using rectangular swept sphere volumes. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (San Francisco,CA, April 2000), vol. 4, pp. 24–48.

[LO05] LIN M. C., OTADUY M. A.: Sensation-preserving haptic rendering. *IEEE Computer Graphics and Applications 25*, 4 (2005), 8–11.

[MDL06a] MADERA F., DAY A., LAYCOCK S.: Collision detection for deformable objects using octrees. In *Theory and Practice of Computer Graphics* (Middlesbrough, UK, 2006), EG.

[MDL06b] MADERA F., DAY A., LAYCOCK S.: The use of tetrahedra to detect collisions. In *Third WorkShop in Virtual Reality and Physical Simulations* (Madrid, Spain, 2006), EG.

[Mir98] MIRTICH B.: V-clip: fast and robust polyhedral collision detection. *ACM Trans. Graph. 17*, 3 (1998), 177–208.

[MO06] MENDOZA C., O'SULLIVAN C.: Interruptible collision detection for deformable objects. *Computers & Graphics 30*, 3 (June 2006), 432–438.

[PLM95] PONAMGI M., LING M., MANOCHA D.: Incremental collision detection for polygonal models. In *Proceedings of the eleventh annual symposium on Computational geometry* (Vancouver, British Columbia, Canada, 1995), pp. 445– 446.

[RKL*04] REDON S., KIM Y., LIN M., MANOCHA D., TEMPLEMAN J.: Interactive and continuous collision detection for avatars in virtual environments. In *IEEE Virtual Reality Conference 2004 (VR'04)* (2004), IEEE, pp. 117–130.

[SBT06] SPILLMANN J., BECKER M., TESCHNNER M.: Efficient updates of bounding sphere hierarchies for geometrically deformable models. In *Third Workshop in Virtual Reality, Interactions and Physical Simulations VRIPHYS'06* (Madrid, Spain, Nov. 2006), EG.

[TKH*05] TESCHNER M., KIMMERLE S., HEIDELBERGER B., ZACHMANN G., RAGHUPATHI L., FUHRMANN A., CANI M., FAURE F., MAGNENAT-THALMANN N., STRASSER W., VOLINO P.: Collision detection for deformable objects. *Computer Graphics Forum 24*, 1 (2005), 61– 81.

[vdB97] VAN DEN BERGEN G.: Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools 2*, 4 (1997), 1– 14.

[YSLM04] YOON S.-E., SALOMON B., LIN M., MANOCHA D.: Fast collision detection between massive models using dynamic simplification. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (New York, NY, USA, 2004), ACM Press, pp. 136–146.

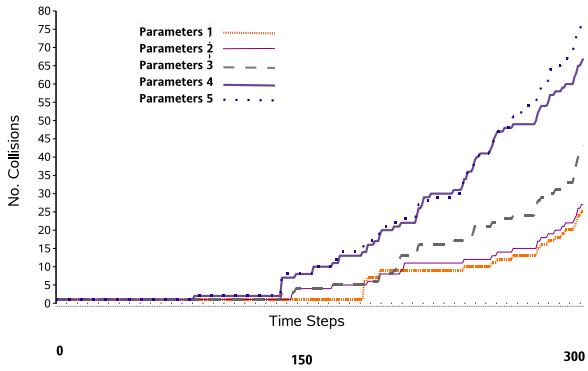

Figure 9: *Number of collisions detected in the first experiment.*

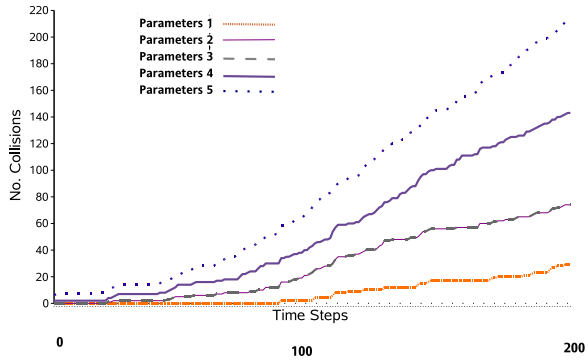

Figure 10: *Number of collisions detected in the second experiment.*

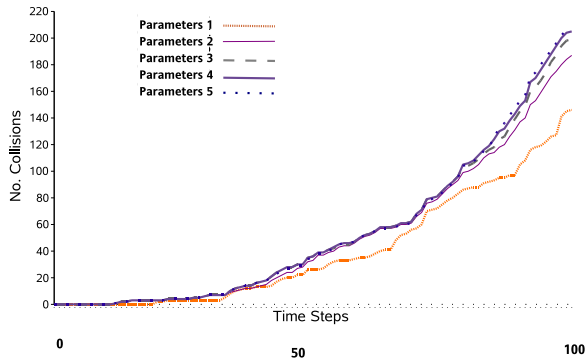

Figure 11: *Number of collisions detected in the third experiment.*