A Combination of Hierarchical Structures and Particle Systems for Self-Collision Detection of Deforming Objects

Masamichi Sugihara
Department of Computer Science,
University of Victoria, BC, Canada
sugihara@cs.uvic.ca

Vladimir Savchenko
Faculty of Computer and Information Sciences,
Hosei University, Tokyo, Japan
vsavchen@k.hosei.ac.jp

Abstract

In this paper, we propose an approach which combines hierarchical structures and particle systems for self-collision detection occurring in a deformable object. Numerous algorithms for collision detection have been proposed in computer graphics applications. Our algorithm exploits the efficiency of hierarchical structures to deal with many polygons, and particle systems because they can be used to extract colliding polygons. We have extended these two algorithms to deal with self-collision detection. The approach is split into two stages. Particles are distributed on the surface of a deformable object. Then, if the particles detect a possibility of a self-collision, hierarchical self-collision detection is started. The algorithm has been implemented on a square cloth model as an example of a deformable object. We show that the algorithm efficiently reduces self-collision detection redundancy, and yet precisely detects self-collision events.

Keywords: *Computer animation, Self-collision detection, Deformable objects.*

1. Introduction

For many years, collision detection has been a complex problem for contemporary computer animation. Collision detection is crucial for computer animation in order to prevent collided objects from penetrating each other; however, this procedure is very time consuming. Spatial pre-processed data structures are commonly used for rigid objects, however they are often too inefficient to be usable in interactive applications using deforming objects.

Self-collision detection becomes very important when highly flexible deformable objects are used. Self-collision detection is generally more difficult than detecting collisions between separate rigid bodies and different heuristics have been presented. We examine only the algorithms which efficiently detect self-collision detection.

1.1. Previous work

There are many effective algorithms for collision detection. Collision detection algorithms have been accelerated by various approaches based on bounding volume hierarchies, distance fields, image-space techniques, etc. One of the collision detection approaches is spatial partitioning proposed in [1], [2]. This approach splits 3D space into many regions in a pre-processing step. There are several methods for splitting space, such as using a grid, a tree, and spatial sorting methods. If there are more than two objects in the same region, a polygon-polygon collision check is called. This procedure leads to reducing calculations. Algorithms for optimal spatial partitioning are also discussed in Ray tracing [3]. Another example is a particle based collision detection algorithm proposed in [4]. In this approach, particles serve as a sensor which detects a collision. Two different type particles are distributed on each object. If different type particles close to each other, a polygon-polygon collision check can be called. Bounding volume hierarchies [5], [6], [7], [8] are the most efficient data structures for collision detection. This approach composes tree structures while every node has bounding volumes. There are many kinds of bounding volumes, such as spheres, axis-aligned bounding boxes (AABBs), oriented bounding boxes (OBBs), k-DOPs, and convex hulls. Those bounding volumes are used to construct bounding volume hierarchies. In these algorithms, if the traversal reaches a leaf node, a polygon-polygon collision check is called.

Those algorithms are mainly for collision detection between rigid objects. They don't work well for self-collision detection because they still call many redundant polygon-polygon collision checks. Therefore, several algorithms for self-collision detection are also proposed. Papers [9] and [10] present an approach based on geometrical shape regularity properties. By taking advantage of geometrical shape regularity properties and hierarchical structures, the computation time is reduced. The approach proposed in [11] uses chromatic decomposition. In a pre-processing step, the mesh of a deforming object is divided into several groups. Each group doesn't have primitives which are adjacent. By using these groups, the approach significantly reduces redundant polygon-polygon collision checks.

In this paper, we present work in progress and report our preliminary results devoted to self-collision detection of flexible objects such as cloth. We propose a method to find a self-collision event for deformable geometry objects. In order to achieve interactive speeds we combine hierarchical structures and particle systems, and present a new model of particle interaction.

1.2. Our approach

We combine hierarchical structures and particle systems for efficient self-collision detection. Our approach is as follows:

1. In a pre-processing step, bounding volume hierarchies of a deformable object are pre-constructed; then a deformable object is animated.

2. Particles are distributed on the deformable object and are used as an initial guess for self-collision detection.

3. If the particles detect a possibility of a self-collision, hierarchical self-collision detection is started.

Because our approach adopts hierarchical structures and particle systems, we combine the advantages of these two algorithms:

- Intelligent particles are able to detect a possible self-collision. As a result, the time consuming self-collision detection procedure is called far fewer times.

- A hierarchical structure is particularly beneficial for speeding up self-collision detection when there are a large number of objects. It also has the advantage that various other data can be stored in each node.

The rest of the paper is as follows. In Section 2 and Section 3, we present our approach. Section 4 contains implementation details. In Section 5, we show examples of animation and speed benchmarks. Section 6 presents our conclusions and future work.

2. Particle systems

We improve the particle-based approach of [4] to obtain an initial guess for precise self-collision detection. The main idea of the particle-based approach is that particles move around a deformable object where they are distributed according to a physical law. If the particles don't detect a possibility of a self-collision, the detection terminates.

Let us note that the approach discussed in [4] does not provide correct collision detection for objects with highly deformable geometry, critical for garment animation.

2.1. Particle definition

Our method differs from [4] in that we define a single type of particle. These particles move around a deformable object, similar to the flow of electric charges.

Additionally, for efficient self-collision detection the number of particles is defined as equal to the number of convex parts of the object.

2.2. Original distance and direction

An original distance and direction introduced here are important for calculating particle interactions. The distance between two particles is the length of the path connecting them. In an analogy to the shortest path or geodesic, we define an original distance as the distance along polygons of a deformable object. In other words, for a plane, it is the distance on a deformable object instead of a straight line as shown in Figure 1 (the blue line represents the original distance between particles). The straight line (red line) in Figure 1 is the distance between particles simply given by the Pythagorean Theorem. In Figure 1, arrows show a direction. The reason why an original distance is necessary is that self-collision more often occurs for the polygons that are close on the straight (Red line) yet far on the distance line along a deformable object. Particles search a deformation object for such polygons. However, we need to impose some rules on particles so that particles can find such polygons. The rules adapted to particles are discussed in Section 2.3.

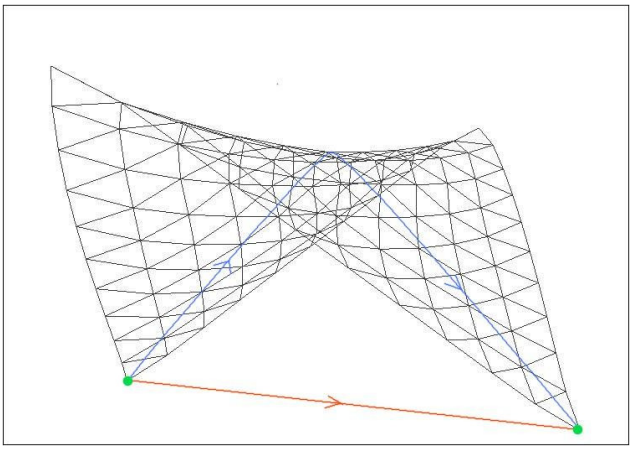

Figure 1: Illustration of the original distance and direction

2.3. Interaction

Particles are affected by two types of forces, attractive forces and repulsive forces. Attractive forces move particles close to each other in a straight line (SL) yet far on an original distance line (ODL). That is, attractive forces will be strong when particles are close in an SL. In addition, attractive forces are proportional to the distance difference between an SL and an ODL. If the distance between particles for the ODL is equal to the distance on the SL, attractive forces don't work, however, if particles are influenced by only attractive forces they immediately gather in the one place on a deformable object. Therefore, we add repulsive forces to affect particles. Repulsive forces will be strong when particles are close to each other on an ODL. As a result, these particles allow us to simulate self-collision detection efficiently, detecting polygons close on an SL yet far on an ODL. The total force $\vec{F}_i$ which affects *i*-th particle is as follows:

$$\vec{F_i} = \sum_{j=0}^{n} \left\{ t \cdot fa(rs_{ij})\overrightarrow{rs_{ij}} + fr(ro_{ij})\overrightarrow{ro_{ij}} \right\},$$

$$t = \frac{ro_{ij} - rs_{ij}}{ro_{ij}}, \; (ro_{ij} \neq 0),$$

$$fa(r) = \frac{a}{r^2},$$

$$fr(r) = -\frac{b}{r^2},$$

where *fa* and *fr* denote functions for attraction and repulsion, *rs* denotes the SL between particles, *ro* denotes the ODL between particles, *t* is a variable, and *a* and *b* are constants. We define *t* as a variable to move particles close to each other on an SL yet far on an ODL. The denominator of *t* doesn't become zero, because one particle isn't on the top of another. In addition, we substitute appropriate numbers for *a* and *b* for smooth motion of particles. In our experiments, *a* is 20 and b is 10. The constant substituted for *a* is bigger than the constant substituted for *b* because the forces which find self-collision are attractive forces. Repulsive forces are mere forces to evenly spread particles over a

deformable object. The initial positions of particles don't need to be placed in particular positions.

2.4. Self-collision detection in particle systems

When particles start to move around a deformable object, we have to wait for relaxation of particles. In theory, we must wait until all particles stop moving, but it is crucial to simulate self-collision detection in real-time. Therefore, to simulate self-collision detection by a system of particles we define two values *Nmax* and *Nmin* to control the relaxation of particles. *Nmax* is the maximum number of particle update steps. *Nmin* is the minimum number of moving particles. The algorithm is as follows:

- The particle positions are updated until either the number of moving particles is less than *Nmin*, or the number of particle update steps is equal to *Nmax*.
- Then, if the distance between closest particles is less than some tolerance distance, hierarchical self-collision detection is called.

3. Hierarchical structures

We use bounding volume hierarchies as hierarchical structures. The main principle of bounding volume hierarchies is that if the parent volume of a polygon doesn't collide with another volume during collision detection, we don't need to check whether the polygon collides with another polygon. However, even if we use bounding volume hierarchies, self-collision isn't detected efficiently because bounding volumes will always find contacts between adjacent polygons. Therefore, we will waste much time calling the polygon-polygon collision check for all the adjacent polygons. To resolve this problem, we include the algorithm based on geometrical shape regularity properties (discussed in [9]) to hierarchical structures. This algorithm requires adding information (positive vector) to every node in a hierarchical structure. If the algorithm detects a self-collision possibility, self-collision detection in bounding volume hierarchies is called. Details of the algorithm are discussed in Section 3.3. To make this paper almost self contained let us briefly discuss hierarchical structures.

3.1. Node

Various kinds of information are assigned to every node of a hierarchical structure. Firstly, we define the type of every node, LEAF or NODE. According to the type, information contained in each node is different:

- Object (Only LEAF node). This part has a polygon which is judged whether it collides with another polygon.
- Left and Right (Only NODE node). We define that a parent node has two child nodes. Left and Right store one child node of the parent node respectively.
- Bounding Volume. This part has a bounding volume which is used when whether collision detection descends to child nodes or not are judged. We adapt AABB as a bounding volume.
- Positive Vector. Positive Vector is used only in the algorithm based on geometrical shape regularity properties**.**

3.2. Construction of bounding volume hierarchies

There are many methods to construct bounding volume hierarchies, such as a top-down method and a bottom-up method. We construct bounding volume hierarchies by using the bottom-up method [12]. This method can construct a better tree than another method, but implementation of this method takes much time. However, since we construct bounding volume hierarchies before animation, this method does not require a real-time implementation. Although node information needs to be updated, the structure of a bounding volume hierarchy doesn't need to. Thus, it doesn't take much time to update node information.

To construct bounding volume hierarchies by using a bottom-up method, firstly, every polygon is covered under a bounding volume. The volume becomes the LEAF node in a hierarchical structure. Next, we group two nodes of a given level into a new node of above level and repeat this process until constructing the root node which has the volume containing all polygons. In this time, we choose two nodes in order that the volume of a new node is the minimum.

3.3. Self-collision detection in the algorithm based on geometrical shape regularity properties

In this section, we summarize the algorithm discussed in [9]. The main principles of the algorithm are as follows:

(A) A surface is curved enough for making a "loop" and hitting another part of the surface.

(B) The contour of a surface has such a shape that a minimal fold will bring superposition and self-collision of the surface.

However, in most cases, a deformable object doesn't self-collide in the (B) case. Therefore, we consider only the (A) case. To test (A), we search for a vector which has positive dot product with the normal vectors of all polygons of a deformable object. If the vector exists, we conclude that the condition of (A) isn't satisfied and then there are no self-collisions on the deformable object. We define the vector as a "positive vector." In addition, even if the condition of (A) is satisfied, if there exists a positive vector in two nodes on the deformable object surface which are adjacent (connected by at least one vertex), there are no self-collisions of the nodes on the deformable object. Therefore, self-collision detection in bounding volume hierarchies is called on the surface where a positive vector doesn't exist.

3.4. Self-collision detection in bounding volume hierarchies

When two volumes collide with each other, a search has to be made through the hierarchical structure. There are some methods, such as breadth-first search (BFS) and depth-first search (DFS). We use DFS in which collision detection descends through the child nodes until a leaf node is reached. DFS requires a rule to define a particular child node. The search descends to the node defined. According to the rule used, child nodes are dynamically searched and collision detection descends to the child nodes of the parent node which has a bigger volume than another parent node. This rule can reduce the sum of volumes which are used in subsequent collision detection. Therefore, the algorithm for self-collision detection in bounding volume hierarchies is as follows:

- When two volumes collide with each other, self-collision detection descends to the child nodes of the parent node which has a bigger volume than another parent node.
- If nodes of two collided volumes are LEAF nodes, a polygon-polygon collision check is called.

4. Software implementation

The proposed algorithm has been implemented in C++, and a deformable object and particles are visualized by OpenGL. To implement the proposed algorithm, we have modeled a cloth as a deformable object and simulated collision response.

4.1. Cloth modeling

We model the cloth, adopting the algorithm discussed in [13]. This algorithm is based on a mass-spring system. The cloth is calculated by implicit integration. Implicit integration allows us to simulate collision detection and modify polygons in which collisions occur. This modification is repeated until all result points are satisfied with desired positions. Additionally, we consider stretch forces and damping forces as internal forces, and gravity as external forces in the cloth. Under those environments, we have implemented our algorithm.

4.2. Collision response

Once a collision is detected, we have to force the cloth not to penetrate the obstacle. The approach of [14], is used, the part of the cloth in collision is constrained against the obstacle and the cloth motion is enforced. This is done by manipulating the forces as follows. Forces can be ignored by assigning a polygon infinite mass ($1/m = 0$) its acceleration becomes zero regardless of the values of all forces exerted on it.

We apply the approach as follows. To block a polygon's acceleration, while the value of a mass is changed (for more references see [14]), we modify the value of a force. We remove the component of a force along the normal direction of an obstacle. This equation is as follows:

$$\vec{a} = \frac{1}{m}\left\{\vec{F} - \left(\vec{F}\cdot\vec{n}\right)\vec{n}\right\},$$

where $\vec{n}$ denotes the normal vector of an obstacle and $\vec{a}$ denotes an acceleration.

Consequently, a polygon is prevented from accelerating along the normal direction of an obstacle and the polygon doesn't penetrate the obstacle. However, if the obstacle has acceleration, the acceleration along the normal direction of the obstacle must be added to the polygon's acceleration.

5. Results

The tests were performed on a PC with the Intel Core Duo 1.66Ghz processor. We tested our software using three different arrangements of a square cloth model (shown in Figure 2). The model is defined by 200 polygons, 121 vertices, and the distribution of 4 particles was used. Additionally, we present benchmarks that show the use of hierarchical structures and the combination of hierarchical structures and particle systems.

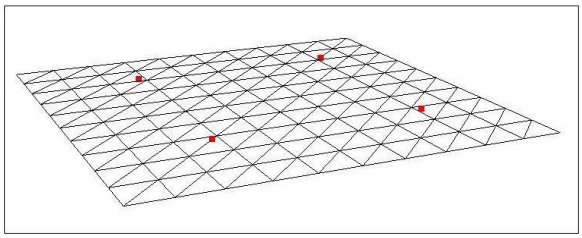

Figure 2: Cloth model

Figure 3 shows a cloth where two diagonal corners are constrained and benchmark results are shown in Figure 4. Though the cloth didn't self-collide, a polygon-polygon collision check was called in the case of no particles. On the other hand, a polygon-polygon collision check wasn't called in the case of using particles. The average time to perform collision detection was 16ms. In this example, the combination of hierarchical structures and particle systems (blue line) doesn't have an active interval, because particles don't detect self-collision possibilities.

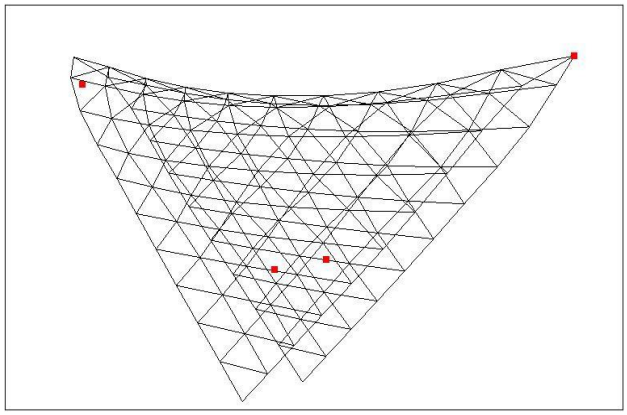

Figure 3: Cloth model where two diagonal corners are constrained

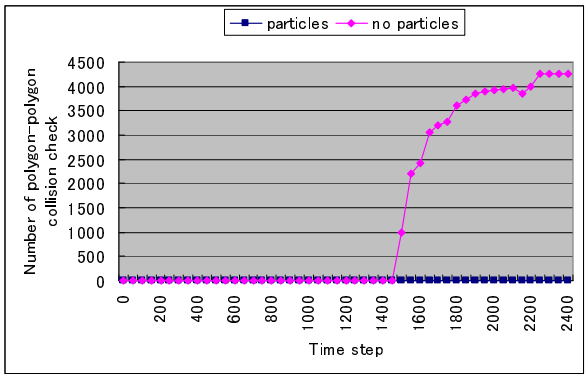

Figure 4: Benchmark for Figure 3

Figure 5 shows a cloth model where a one corner is constrained and results of this benchmark are shown in Figure 6. The number of polygon-polygon collision checks is reduced. This fact proves efficiency applying particle-based self-collision detection. The average time to perform collision detection on the active interval is 29ms.

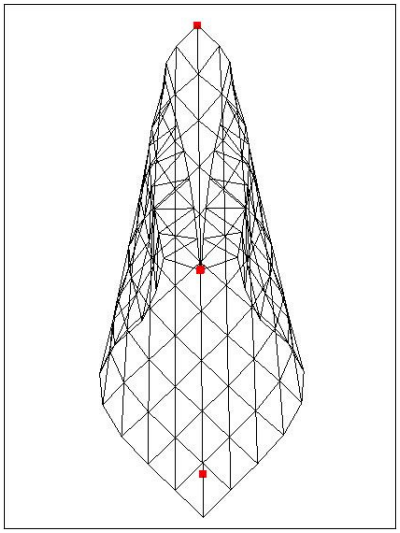

Figure 5: Cloth model where a one corner is constrained

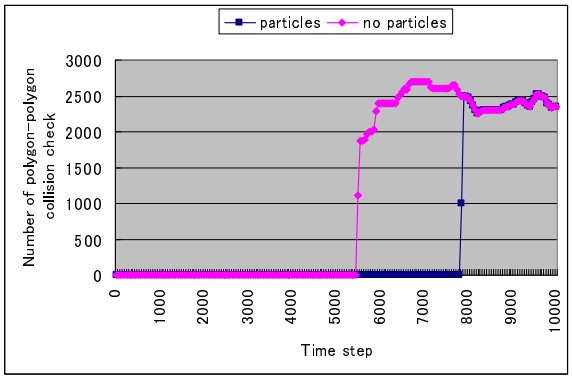

Figure 6: Benchmark for Figure 5

Figure 7 shows a cloth model where a central point is constrained; the benchmark results are shown in Figure 8. In this case, whether particle-based self-collision detection was used or not, the number of polygon-polygon collision checks is identical. This benchmark proved that particles could precisely detect self-collision detection. The average time to perform collision detection on active interval is 106ms.

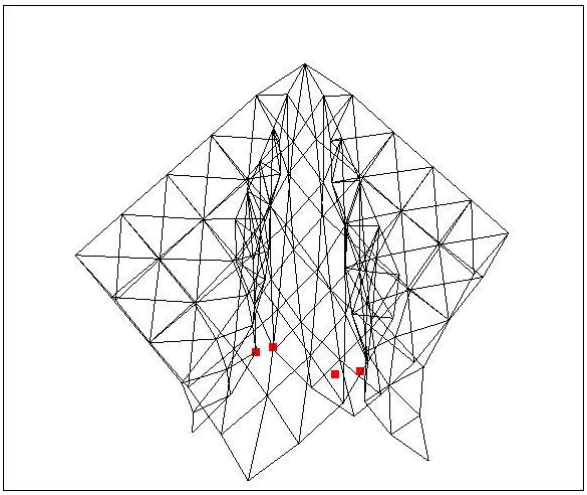

Figure 7: Cloth model where central point is constrained

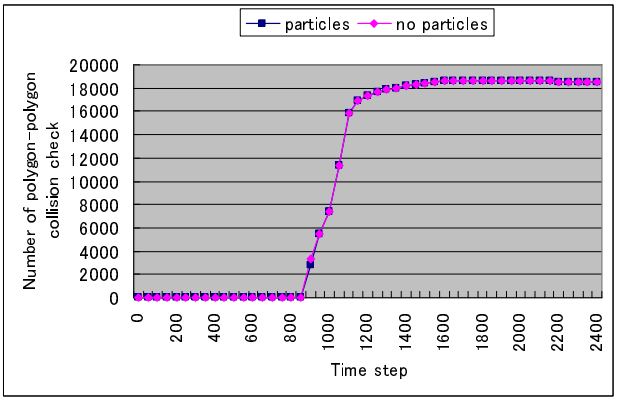

Figure 8: Benchmark for Figure 7

6. Conclusions and future work

In the work, we have proposed an algorithm combining hierarchical structures and particle systems for efficient self-collision detection. Particles smartly extract possible colliding polygons. Additionally, hierarchical structures efficiently deal with many polygons. In fact, after a deformable object self-collides, our software can't decrease processing time. Nevertheless, before a deformable object self-collides, combining those two algorithms allows efficiently reduce useless self-collision detection procedures providing precise self-collision detection.

However, there are several points which should be improved. Firstly, the approach is not robust, after a deformable object self-

collides, the procedure of self-collision detection should be enhanced. Because self-colliding polygons of a deformable object continue to collide with each other, calculation expenses greatly increase. However, since positional relationship between polygons changes slowly, we suppose to cache the relationship to reduce calculations.

Secondly, it is difficult to define appropriate values for attractive forces and repulsive forces. In the case of uniform grid-based collision detection, optimal grid resolution is required in order to keep accuracy and speed up collision detection procedures. Similarly, we have to assign appropriate values to attractive forces and repulsive forces. Otherwise, our algorithm doesn't work well.

Thirdly, our algorithm has been implemented only for a cloth model. When we have a curved surface, it is difficult to measure an original distance on the curved surface. In order to solve this problem, we consider invisible particles (IP). IPs are different from particles to detect collisions (CP). One IP is allocated to two CPs in order to calculate the original distance between the CPs. IP is forced to move from one CP to another CP. In addition, an IP can move only on the surface of an object. After an IP reaches a CP, we calculate the distance of the path along which the IP passed. By using this algorithm, we are planning to investigate applicability of the proposed approach for objects with changing topology in the future.

7. References

[1] R. Bigliani and J. W. Eischen, Collision Detection in Cloth Modeling, chapter in Cloth Modeling and Animation, A. K. Peters, pp. 197-217, 2000.

[2] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi, I-collide: An Interactive and Exact Collision Detection System for Large-scale Environments, In Proceedings of ACM International 3D Graphics Conference, pp. 189-196, 1995.

[3] V. Havran and F. Sixta, Comparison of Hierarchical Grids, Ray Tracing News, 1999.

[4] M. Senin, N. Kojekine, V. Savchenko, and I. Hagiwara, Particle-based Collision Detection, In short papers proceedings of Eurographics EG2003, pp. 1-8, 2003.

[5] P. M. Hubbard, Collision Detection for Interactive Graphics Applications, IEEE Transactions on Visualization and Computer Graphics, pp. 218-230, 1995.

[6] G. van den Bergen, Efficient Collision Detection of Complex Deformable Models Using AABB Trees, Journal of Graphics Tools, pp. 1-14, 1997.

[7] S. Gottschalk, M. C. Lin, and D. Manocha, OBB-Tree: A Hierarchical Structure for Rapid Interference Detection, In Proceedings of ACM SIGGRAPH, pp. 171-180, 1996.

[8] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs, IEEE Transactions on Visualization and Computer Graphics, pp. 21-36, 1998.

[9] P. Volino and N. Magnenat Thalmann, Efficient Self-Collision Detection on Smoothly Discretized Surface Animations using Geometrical Shape Regularity, Computer Graphics Forum (EuroGraphics Proceedings), pp. 155-166, 1994.

[10] P. Volino, M. Courchesne, and N. Magnenat Thalmann, Versatile and Efficient Techniques for Simulating Cloth and Other Deformable Objects, In Proceedings of ACM SIGGRAPH, pp. 137-144, 1995.

[11] N. K. Govindaraju, D. Knott, N. Jain, I. Kabul, R. Tamstorf, R. Gayle, M. C. Lin, and D. Manocha, Interactive Collision Detection between Deformable Models using Chromatic Decomposition, ACM Transactions on Graphics, pp. 991-999, 2005.

[12] S. M. Omohundro, Five Balltree Construction Algorithms, Thechnical Report TR-89-063, International Computer Science Institute, Berkeley, CA, 1989.

[13] M. Desbrun, P. Schroder, and A. Barr, Interactive Animation of Structured Deformable Objects, In Proceedings of Graphics Interface '99, pp. 1-8, 1999.

[14] D. Baraff and A. Witkin, Large Steps in Cloth Simulation, In Proceedings of ACM SIGGRAPH, pp. 43–54, 1998.