# ICCD: Interactive Continuous Collision Detection between Deformable Models using Connectivity-Based Culling

Min Tang, Sean Curtis, Sung-Eui Yoon, and Dinesh Manocha

***Abstract*—We present an interactive algorithm for continuous collision detection between deformable models. We introduce multiple techniques to improve the culling efficiency and the overall performance of continuous collision detection. First, we present a novel formulation for continuous normal cones and use these normal cones to efficiently cull large regions of the mesh as part of self-collision tests. Second, we introduce the concept of "procedural representative triangles" to remove all redundant elementary tests between non-adjacent triangles. Finally, we exploit the mesh connectivity and introduce the concept of "orphan sets" to eliminate redundant elementary tests between adjacent triangle primitives. In practice, we can reduce the number of elementary tests by two orders of magnitude. These culling techniques have been combined with bounding volume hierarchies and can result in one order of magnitude performance improvement as compared to prior collision detection algorithms for deformable models. We highlight the performance of our algorithm on several benchmarks, including cloth simulations, N-body simulations and breaking objects.**

***Index Terms*—Continuous collision detection, deformable models, continuous normal cones, orphan set, self-collision, bounding volume hierarchies**

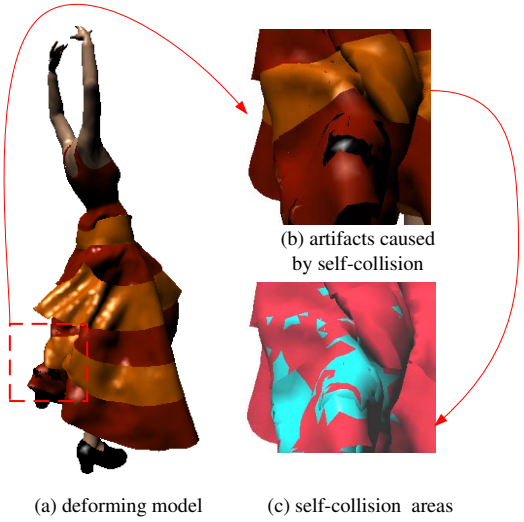


Fig. 1. **Self-collision in simulation:** (a) a deforming cloth model with 92k triangles and composed of multiple layers; (b) artifacts caused in the simulation by self-collision; (c) zoomed view of self-colliding primitives (the blue areas).

## I. Introduction

Interactive simulations with deforming or non-rigid objects are widely used in physically-based simulations, CAD/CAM, computer graphics, and robotics. In order to generate physically realistic or plausible motions, these systems enforce non-penetration constraints and need to detect all collisions between the primitives. The collision queries on deformable models can be classified into inter-object collisions between disjoint objects and self-collisions on a single object.

Most of the prior work in deformable collision detection has been limited to *discrete collision detection*. These algorithms check for overlaps at given discrete time steps in the simulation, and may miss collisions between the time steps. In order to resolve these problems, many researchers have proposed algorithms for *continuous collision detection* (CCD) [1], [2]. CCD techniques model the motion between the discrete time steps using continuous paths and check these paths for overlaps. The continuous algorithms are also used to perform time-of-contact computations for dynamic simulation, haptic rendering and local planning for sample-based motion planning algorithms [3], [4].

In this paper, we address the problem of fast and accurate CCD between deformable models. Accuracy is very important feature for various applications including cloth simulation, where a single missed collision can result in an invalid simulation or noticeable artifacts [5], [6]. This is highlighted in Figure 1.

In practice, performing CCD between complex deformable models at interactive rates still remains a major challenge [7]. The continuous formulation of the motion can result in a very large number of false positives during self-collisions. For example, all the adjacent triangle pairs that share a vertex or an edge may collide as the model is deforming. As a result, one of the challenges is to devise efficient algorithms with high culling efficiency.

Prior algorithms for interactive CCD are mainly limited to rigid and articulated models. In these cases, self-collision detections can either be skipped or carefully avoided. Many fast algorithms based on bounding volume hierarchies(BVHs) and GPU-based accelerations have been proposed for deformable models, but they may not be fast enough for interactive applications.

**Main Results:** We present an interactive algorithm for CCD

Min Tang is with the College of Computer Science and Technology, Zhejiang University, China. Email: tang_m@zju.edu.cn. Sean Curtis and Dinesh Manocha are with the Department of Computer Science, University of North Carolina at Chapel Hill, USA. Email: {seanc, dm}@cs.unc.edu. Sung-Eui Yoon is with the Department of Computer Science, Korea Advanced Institute of Science and Technology, South Korea. Email: sungeui@cs.kaist.ac.kr.

computation between complex deformable models including breaking objects. Our approach is applicable to all triangulated models and exploits the connectivity information between adjacent triangles in a mesh-based representation. We utilize this connectivity information to perform high-level and low-level culling to significantly reduce the number of elementary tests between triangle primitives. Specifically, we introduce three novel techniques:

**1. Continuous Normal Cones (CNC):** Normal cone tests are used to cull mesh regions that cannot have self-collisions during discrete self-collision tests [8], [9]. We extend this well-known normal cone tests to CCD. Our formulation computes a tight bound on the range of normals for triangles under a wide variety of continuous motions. We represent continuous normal cones using Bernstein basis functions. We use the convexity properties to compute the normal cones efficiently. Furthermore, we also present a fast continuous contour test method along with the Continuous Normal Cone.

**2. Procedural Representative Triangles (PRT):** We decompose potentially colliding triangle pairs into two sets: non-adjacent and adjacent pairs, to perform the elementary tests between the primitives. For non-adjacent triangle pairs, there are a lot of redundant elementary tests caused by topological sharing. By extending the concept of Representative Triangles in [10], we introduce "Procedural Representative Triangles" (PRT) to remove the redundant elementary tests. Compared to previous database based techniques [11], [12], our method has faster running performance and lower memory requirements.

**3. Orphan Set:** We derive an optimal bound on the maximum number of elementary tests required to be performed between adjacent triangle pairs. We introduce the notion of an *orphan set* of a mesh based on the connectivity between the triangles and show that only the primitives in the orphan sets need to be checked for exact collision detection among all the adjacent pairs. In our benchmarks, the "Orphan Set" formulation reduces the number of elementary tests between adjacent triangle primitives by almost three orders of magnitude.

We use a two-level bounding volume hierarchy (BVH) and use it to handle multiple-object simulations (including breaking objects) as well as self-collisions. We have applied the algorithm to many complex deformable models composed of tens or hundreds of thousands of triangles. Our algorithm can compute either the first time-of-contact or the full set of colliding triangles during the continuous time domain in tens or hundreds of milliseconds. As compared to prior approaches, our algorithm offers the following benefits:

- **Generality:** Our approach is applicable to various kinds of models and deformable simulations. These include self-collisions, inter-object collisions between multiple objects (i.e. N-body collisions), and breaking objects.
- **High culling efficiency:** We are able to achieve high culling efficiency on adjacent pairs by three orders of magnitude and reduce the number of overall false positives caused by adjacent pairs and non-adjacent pairs by two orders of magnitude in complex simulations.
- **Interactive performance:** Our hierarchy update and traversal algorithms have small overhead. As compared to prior CCD algorithms, we observe considerable performance improvement in our benchmarks.

This paper extends our previous work [11] and present more detailed exposition of our culling methods and results. We also introduce a novel culling method, called "Procedural Representative Triangles" (PRT). It provides an efficient way to remove redundant elementary tests among non-adjacent triangle pairs by testing the connectivity of meshes locally. PRT overcomes one of the main limitations of our previous work [11]. Also, it lowers the memory requirements and makes our algorithm easier to parallelize on multi-core platforms.

**Organization:** The rest of the paper is organized as follows: Section II gives a brief survey of prior work in collision detection. We introduce our notation and give an overview of our approach in Section III. The culling techniques are described in Section IV and Section V. We present the two-level hierarchy and overall algorithm in Section VI, and the results in Section VII. We compare its performance with prior approaches in Section VIII.

## II. Related Work

Collision detection has been widely studied in computer graphics, robotics, and computational geometry literature [13], [14]. In this section, we give a brief overview of prior work on collision detection between deformable models.

### A. Bounding Volume Hierarchies

Bounding volume hierarchies (BVHs) have been widely used to accelerate the performance of collision detection algorithms between rigid and deformable models. Examples of BVHs include sphere trees [15], [16], axis-aligned bounding box (AABB) trees [17], hierarchies based on tight fitting bounding volumes (BVs) such as oriented bounding boxes [18], discretely oriented polytopes (k-DOPs) [19], or hybrid combination of BVs [20].

Most algorithms for deformable models typically use simple BVs such as spheres or AABBs and recompute the BVHs during each frame [14]. Approaches to compute dynamic BVHs include refitting algorithms to update these hierarchies [21], [22], [23], [24], and performing dynamic or selective restructuring [25], [26].

### B. Deformable Models

There is considerable literature on efficient collision checking between deformable models. These include efficient algorithms based on normal cone culling and GPU-based approaches. Bergen [17] presented an early approach using refitting for deformable models.

**Normal Cone Culling:** Volino and Thalmann [8] proposed a culling technique for efficient self-collision detection at discrete time steps using bounds on the normals. This method takes advantage of the topology and connectivity of the mesh and checks for self-collision by using normal cones and two dimensional contour tests. They can be combined with hierarchical approaches to handle highly tessellated models [27], [9], [28]. We extend this normal cone culling method to continuous collision detection.

**GPU-based Algorithms:** The rasterization capabilities of commodity GPUs have been used for fast collision detection between deformable models [29], [30], [31]. These include many specialized algorithms for self-collision detection [32], [33], [34], [35]. Some of these approaches are limited to handling certain types of input models (e.g. closed objects or fixed mesh connectivity). Furthermore, their performance can vary based on the support of occlusion queries or read-backs from GPUs. We perform a detailed comparison with these approaches in Section VIII.

### C. Continuous Collision Detection

CCD algorithms check for collisions in the continuous time interval between two discrete time steps. These include interactive algorithms for rigid models [1] and articulated models [2], [36] that are based on tight-fitting pre-computed hierarchies. CCD techniques for deformable models [7], [33], [37] are mostly limited to models with fixed connectivity. We compare the performance of our algorithm with these approaches in Section VIII.

## III. Overview

In this section, we introduce the notation used in the rest of the paper and give an overview of our approach.

### A. Notation and Definitions

We use the symbols $V$, $E$, $F$, and $T$ to represent vertices, edges, faces, and triangles, respectively. We use lower-case symbols $v$, $e$, $f$, and $g$ to denote a specific vertex, edge, face, and triangle, respectively. The vector quantities are written in bold face, e.g., $\mathbf{n}$ for the normal of a triangle. Also, we use $\{v_i, f_j\}$ to denote a pairwise relationship between two mesh elements, in this case, $v_i$ and $f_j$.

Our CCD algorithm is applicable to triangulated meshes. We refer to each connected mesh as an *object* and the simulation may consists of one or more objects. We do not make any assumptions about the motion of any object or its deformation. Furthermore, the number of objects can change due to topological changes, such as breaking objects. We use the symbol $O^i$ to represent an object and let $M^i$ represent its mesh. Each $M^i$ is represented as triangles with the connectivity and adjacency information, i.e., its vertices, edges, and adjacent triangles. We denote the time interval, in which we perform CCD, to be $[0, 1]$. Let $M_t^i$ denote the configuration of a mesh at time $t \in [0, 1]$. Let $M_0^i$ and $M_1^i$ represent the configuration of mesh $i$ at the two discrete time steps, 0 and 1. As a result, the motion of each triangle during this time interval sweeps out a *triangular prism*[1].

### B. Continuous Collision Detection

In order to compute the first time-of-contact or the set of all collisions, we perform continuous collision detection, including self-intersections, on the meshes. We assume that we know the position of each vertex in the mesh at every time step. We also assume the position of each vertex is defined by a continuous interpolation function. Let a polynomial function $P(t)$, of time $t$, represent the interpolating function for vertex position.

In the simplest case, $P(t)$ is a linear function. Detecting collisions between two such triangles in motion reduces to performing pairwise vertex-face (VF) and edge-edge (EE) elementary tests [5], [9]. These VF and EE elementary tests require solving cubic algebraic equations, which are derived from co-planarity conditions. For two arbitrary triangles, we would need to perform 15 tests: 6 VF and 9 EE tests. For a linear interpolating function, these tests involve finding the roots of a cubic equation.

When $P(t)$ is a polynomial function with degree $d$ ($d > 1$), the resulting test becomes more complex. We still need to perform 15 VF and EE elementary tests, but the resulting polynomial equations would have degree $3d$. For example, if we used cubic spline functions as $P(t)$, we would need to find the roots of an equation with degree 9. Solving these higher order equations can be expensive and result in numerical inaccuracies. As a result, most deformable applications use a simple linear interpolating motion between the discrete time steps.

### C. BVHs for CCD

Most prior algorithms for CCD between complex models use BVHs. At each frame, these algorithms update the BVHs based on the new position of the swept triangles and traverse the BVHs to check for overlaps. Eventually, they perform elementary tests on the triangular prisms to check for exact collisions during the $[0, 1]$ time interval. However, the performance of these algorithms is governed by the two following factors:

**1. Culling efficiency of BVHs:** Intuitively speaking, as a BVH fits the mesh tightly, the BVH can provide higher culling efficiency and better CCD performance. In scenes with extreme deformations, the BVH computed for one frame may not provide good efficiency for the next frame and needs to be updated or recomputed. Many prior algorithms for collision detection between general deformable models use refitting algorithms [17], [21], [24], [33], and they may not work well on scenes with changing topologies.

**2. High number of false positives:** Self-collision detection is performed by recursively checking the BVs for overlap with other BVs. Given that the BVs of adjacent (or nearby) triangle primitives overlap, the hierarchical traversal does not cull many primitives at the lower levels of the tree. As a result, these algorithms perform exact collision tests on a high number of triangle pairs resulting in a very high number of false positives.

### D. Our Approach

We improve the performance of CCD algorithm by using novel culling techniques and a hierarchical representation that maximizes the effect of those culling techniques. Specifically,

[1]It is not strictly a prism. Here we just use the term for the volume formed by a swept triangle.

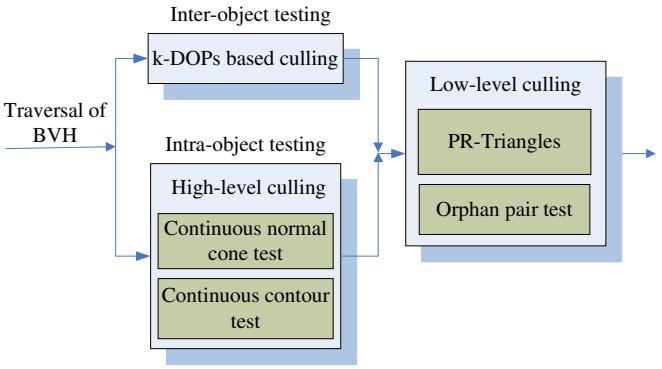


Fig. 2. **Culling pipeline:** By using k-DOPs based culling, high-level culling and low-level culling, the number of elementary tests is significantly reduced.

we introduce a novel two-level dynamic BVH based on k-DOPs that has a low update cost and can provide high culling efficiency. The two-level hierarchy is computed based on the connectivity of the primitives, and we use a combination of refitting and selective restructuring algorithms to update them.

During the traversal of our two-level dynamic BVH, we perform culling operations at two different levels. We first perform *high-level culling* based on a novel continuous normal cone (CNC) formulation. The CNC test consists of the construction of a cone which bounds the direction of all the normals of a surface in a time interval and a continuous contour test which detects collisions on the boundary of the surface in the same interval. These two tests, taken together, are sufficient to eliminate large regions of the mesh from consideration for self-intersection. The CNCs are associated with nodes in the hierarchy and are computed in a bottom-up manner. We use a compact representation of CNCs based on Bernstein basis functions, which have low storage and runtime overhead. These CNC tests work particularly well in culling the regions of the mesh with low curvature.

Our approach also performs low-level culling to further reduce the number of elementary tests. We apply our techniques to reduce the number of exact elementary tests between the triangle features. The first technique, PRT, eliminates all possible duplicate tests instantiated by pairs of non-adjacent triangles. The second technique, Orphan set, provides a small, exact bound on the elementary tests that need to be performed between adjacent triangle pairs.

**Overall CCD Pipeline:** For each simulation time step, the algorithm for CCD between deformable models is executed in two stages. At the first stage, only collisions between non-adjacent triangle pairs are considered. During the traversal of the two-level BVH, k-DOPs based culling is used to reduce inter-object testing. We also use high-level culling to reduce intra-object testing. The high-level culling consists of two methods: continuous normal cone testing and continuous contour testing. During the second stage, low-level culling is used to deal with non-adjacent and adjacent triangle pairs. For non-adjacent triangle pairs, PRT are used to remove redundant elementary tests. For adjacent triangle pairs, only a small set of feature pairs, called an orphan set, need to be checked at this stage. The overall running pipeline of above culling techniques is shown in Figure 2.

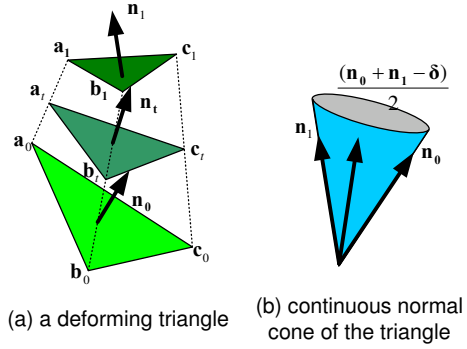


Fig. 3. **Continuous normal range of a deforming triangle**: For a deforming triangle, we construct a CNC that contains $\mathbf{n_0}$, $\mathbf{n_1}$, and $(\mathbf{n_0} + \mathbf{n_1} - \delta)/2$.

## IV. High-Level Culling

In this section, we present a novel high-level culling algorithm that significantly reduces the number of false positives, leading to more efficient execution of self-intersection queries.

One of the most expensive computations in deformable models is self-collision detection. Prior self-collision methods based on normal cones are limited to discrete collision detection. We extend them to CCD and present a compact representation to compute a normal cone and quickly check for collisions.

### A. Discrete Normal Cone

Given a continuous surface, S, bounded by a contour, C, Volino and Thalmann [8] presented a sufficient criterion for no self-intersection based on the following two conditions:

1) **Bounds on the normals:** There is a vector, $\mathbf{V}$, such that $(\mathbf{N} \cdot \mathbf{V}) > 0$ for every point of the surface, S, where $\mathbf{N}$ is the normal vector for a point of the surface.
2) **No self-intersections on the boundary:** The projection of the contour C along the vector $\mathbf{V}$ does not have any self-intersections on the projected plane.

The second condition is also called the *contour test*. Provot [9] presented an efficient method to implement the first condition based on normal cones. The normal cone for a region of triangles is computed by merging the normal vectors of those triangles.

In order to extend these tests to CCD, we need to develop continuous-versions of these tests over the range of normals and contours in the interval $t \in [0, 1]$.

### B. CNC: Continuous Normal Cone

In order to use the normal cone for CCD, we compute a normal cone that bounds the normals of the deforming triangles in the entire interval. Let $\mathbf{a_0}, \mathbf{b_0}, \mathbf{c_0}$ and $\mathbf{a_1}, \mathbf{b_1}, \mathbf{c_1}$ to be positions of the vertices of the triangles at the time frame $t = 0$ and $t = 1$, respectively, as shown in Figure 3(a). Also, let us define $\vec{\mathbf{v_a}} = \mathbf{a_1} - \mathbf{a_0}, \vec{\mathbf{v_b}} = \mathbf{b_1} - \mathbf{b_0}$, and $\vec{\mathbf{v_c}} = \mathbf{c_1} - \mathbf{c_0}$. Assuming that the vertices of the triangles undergo linearly interpolating motion, we use the following theorem to compute normal cones:

**CNC Theorem:** *Given the start and end positions of the vertices of a triangle during the interval* $[0, 1]$*, which are*

*linearly interpolated in the interval with respect to the time variable, t, the normal, $\mathbf{n_t}$, of the triangle, at time t, is given by the equation:*

$$\mathbf{n_t} = \mathbf{n_0}\cdot B_0^2(t) + (\mathbf{n_0} + \mathbf{n_1} - \delta)/2 \cdot B_1^2(t) + \mathbf{n_1}\cdot B_2^2(t),$$

*where* $\mathbf{n_0} = (\mathbf{b_0} - \mathbf{a_0}) \times (\mathbf{c_0} - \mathbf{a_0})$, $\mathbf{n_1} = (\mathbf{b_1} - \mathbf{a_1}) \times (\mathbf{c_1} - \mathbf{a_1})$, $\delta = (\vec{\mathbf{v}}_\mathbf{b} - \vec{\mathbf{v}}_\mathbf{a}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{a})$, *and* $B_i^2(t)$ *is the* $i^{th}$ *basis function of the Bernstein polynomials of degree 2.*

*Proof:* Please refer to [11] for a detailed proof. ■

We take advantage of the convex hull property associated with control points of Bernstein basis to compute a bound on CNCs. For a given triangle, the range of $\mathbf{n_t}$ is bounded by the *control vertices*; in our case, those control vertices are $\mathbf{n_0}$, $\mathbf{n_1}$, and $(\mathbf{n_0} + \mathbf{n_1} - \delta)/2$. We use these three vectors to construct a CNC for each triangle in the interval, as shown in Figure 3(b). We also use the method proposed in [9] to construct an axis and an apex angle of a normal cone from three vectors. Then, the CNCs are merged as described in [9] by traversing the hierarchy in a bottom-up manner.

### C. CCT: Continuous Contour Test

Computing the continuous normal cone satisfies the first condition for showing no self-collisions. We still need to satisfy second condition: a collision-free boundary for a moving surface. This typically involves computing a projection of the contour of $S$ and checking for self-intersections. Even for discrete collision detection, the contour test can be an expensive operation. Some prior algorithms either omit it under standard geometrical contexts [8] or use some approximations [9], [27]. In this section, we present an exact and efficient contour tests method for CCD. At a high level, we transform the contour tests into intersection tests between two edges that lie on the same plane. We refer to them as *planar (E,E) tests*, in order to differentiate it from the EE elementary tests used in CCD to check whether two swept edges overlap.

**Planar (E,E) Tests:** Given a node in the BVH with a CNC $C(\alpha, \mathbf{ax})$, where $\alpha$ is the apex angle, and $\mathbf{ax}$ is the axis of the cone. We project the boundary edges of the connected mesh associated with the node to a plane defined by $\mathbf{ax}$ and check for self-intersection among the projected edges.

We illustrate our approach with a simple example. Consider the two edges $\mathbf{ab}$ and $\mathbf{cd}$ in Figure 4(a). The vertices representing their positions are $\mathbf{a_0}, \mathbf{b_0}, \mathbf{c_0}, \mathbf{d_0}$ at $t = 0$, and $\mathbf{a_1}, \mathbf{b_1}, \mathbf{c_1}, \mathbf{d_1}$, at $t = 1$. In order to check whether any projection of these two edges intersect during $t \in [0, 1]$, we preform the following tests:

1) **Discrete line segment intersection test:** A discrete line segment intersection between $\mathbf{a_0b_0}$ and $\mathbf{c_0d_0}$ at $t = 0$ is shown in Figure 4(b). If these discrete segments do not intersect, we need to further test to ensure there is no intersection during $t \in (0, 1]$;
2) **Vertex/edge(VE) elementary test**: Suppose the two deforming edge segments intersect during the interval. Let $t \in (0, 1]$ be the time of first contact between them. For two moving edges, there is only one case of elementary contact type: one vertex of an edge just touches another edge. Analogous to the elementary tests between the triangles, i.e. VF or EE tests, we need to perform a VE test in the plane. Take the case in Figure 4, it boils down to 4 VE tests that are based on the following combinations: vertex $\mathbf{a}$ with edge $\mathbf{cd}$ (Figure 4(c)), vertex $\mathbf{b}$ with edge $\mathbf{cd}$ (Figure 4(d)), vertex $\mathbf{c}$ with edge $\mathbf{ab}$ (Figure 4(e)), and vertex $\mathbf{d}$ with edge $\mathbf{ab}$ (Figure 4(f)). If any of these four tests returns a true value, that implies an intersection between deforming edges $\mathbf{ab}$ and $\mathbf{cd}$.

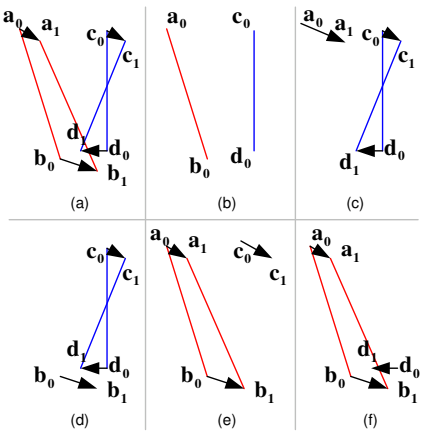


Fig. 4. **Planar (E,E) tests :** Checking whether two co-planar edges $\mathbf{ab}$ and $\mathbf{cd}$ (shown in (a)) intersect during the interval reduces to discrete line segment intersection test (b) and VE tests (c)-(f).

We use the following theorem to perform a VE test in a plane.

**VE Test Theorem:** *Suppose that a vertex* $\mathbf{a}$ *and an edge* $\mathbf{cd}$ *undergo linear deformation in the time interval* $[0, 1]$. *Let* $\mathbf{a_0}, \mathbf{b_0}, \mathbf{c_0}, \mathbf{d_0}$ *be the positions of all the vertices at* $t = 0$, *and* $\mathbf{a_1}, \mathbf{b_1}, \mathbf{c_1}, \mathbf{d_1}$ *be the positions at* $t = 1$, *respectively, as shown in Figure 4(c). Also, let us define* $\vec{\mathbf{v}}_\mathbf{a} = \mathbf{a_1} - \mathbf{a_0}, \vec{\mathbf{v}}_\mathbf{c} = \mathbf{c_1} - \mathbf{c_0}$, *and* $\vec{\mathbf{v}}_\mathbf{d} = \mathbf{d_1} - \mathbf{d_0}$. *Then, the intersection between the edge and the vertex can be computed by the roots of the following equation:*

$$\begin{aligned} (\mathbf{a_0} - \mathbf{d_0}) \times (\mathbf{c_0} - \mathbf{d_0}) \quad &+ \\ [(\vec{\mathbf{v}}_\mathbf{a} - \vec{\mathbf{v}}_\mathbf{d}) \times (\mathbf{c_0} - \mathbf{d_0}) \quad &+ \\ (\mathbf{a_0} - \mathbf{d_0}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{d})] \cdot \mathbf{t} \quad &+ \\ (\vec{\mathbf{v}}_\mathbf{a} - \vec{\mathbf{v}}_\mathbf{d}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{d}) \cdot \mathbf{t^2} \quad &= \quad 0. \end{aligned} \tag{1}$$

*Proof:* Please refer to [11] for a detailed proof. ■

After computing the intersection, a trivial test is used to check whether the intersection point lies between the two ends of the edge. Based on the above theorem, the problem of planar (E, E) tests can be solved by solving four quadratic equations. A naive implementation would perform planar (E, E) tests among all pairs in $O(N^2)$ complexity, where $N$ is the number of the edges in the contour. We use different acceleration techniques to speed up the computation, including:

1) **Bounding box culling:** We compute a k-DOP for each deforming edge swept over the time interval $[0, 1]$. If the k-DOPs of two boundary edges do not overlap, the edges cannot intersect.

2) **Fewer projections:** The projected boundary edges used for the parent nodes can be directly reused for the children nodes. As a result, only a few additional boundary edges need to be projected for successive child nodes. We maintain a cache that contains the boundary edges used for the parent nodes. We store boundary edges in the cache only if the normal cone test is satisfied, but the contour tests fails. This is performed as part of hierarchy traversal.
3) **Fewer intersections:** If two contour edges for a particular node intersect, then any descendant nodes that have those edges in its contour will also fail the contour test. So, we do not perform this test on children nodes.

These optimizations are easy to implement and result in considerable speedups in our benchmarks. For example, we are able to improve the performance of continuous contour test by 93% for the cloth-simulation benchmark (Figure 12) based on these accelerations.

### D. Discussion

**Extension to other interpolating functions:** In our derivation, the CNC is defined with the assumption that the position of the deforming vertices are linearly interpolated (i.e., $P(t)$ is linear). But for higher-order functions, e.g. quadratic, cubic, etc., the CNC equation for other motions, e.g., polynomial interpolation, can be derived in a similar manner. A higher order formulation will result in a larger set of control points for the higher-order Bernstein functions. When the $P(t)$ is a polynomial function with degree $d$, a Bernstein basis with degree $2d$ is needed to perform the CNC test.

**Robustness:** As pointed out by Andersson et al. [40], there are several pitfalls to use the criterion of [8] when some hypotheses such as exclusion of singularities, simple connectedness of the parametric domain, interpretation of the projection, non-selfintersection of the projected contour, etc. are not satisfied. In our definition of CNC and CCT, the *simple connectedness* is ensured by storing connected triangles at BVH nodes, and *non-selfintersection of projected contour* is checked at the CCT phase. These methods enhance the robustness of our algorithm.

## V. Low-Level Culling

In the previous section, we presented a test to cull potentially large regions of the mesh from consideration for self-intersection. Ultimately, we need to test for collisions in the remaining regions. To perform these tests, we logically partition the candidate triangle pairs into two sets: non-adjacent and adjacent pairs. The initial phase culls the non-adjacent triangle pairs to find potentially colliding triangle pairs. We perform exact collision tests on those pairs. The second phase uses the novel concept of an *orphan set* to perform the optimal number of tests between adjacent triangle pairs.

### A. Non-adjacent Phase

The high-level culling produces non-adjacent triangle pairs whose bounding volumes overlap. For each of these pairs we exhaustively perform all 15 elementary tests. By virtue of their non-adjacency, all intersections detected are meaningful. The same criteria may not hold for adjacent pairs.

If we were to evaluate every test arising from the non-adjacent triangle pairs, we would perform redundant elementary tests. This is because a single vertex (or edge) is typically shared by multiple triangles. Rather, we use an efficient duplication removal technique, PRT, to guarantee all these elementary tests will be evaluated once and only once.

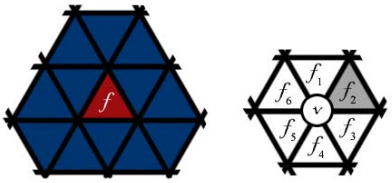


Fig. 5. **Two regions of a single mesh:** The highlighted vertex on the right intersects the red vertex on the left. The shaded triangle on the right is the vertex's representative triangle.

### B. Procedural Representative Triangles (PRT)

Curtis et al. [10] uses the concept of "Representative Triangles" (R-Triangles) to cheaply eliminate duplicate elementary tests. The approach provides a unique assignment of every edge and vertex to an incident triangle and uses this mapping to dispatch elementary tests only once. Figure 5 illustrates how R-Triangles work. The two triangle mesh sections represent portions of the same mesh. The vertex, $v$, in the sub-mesh on the left intersects the red triangle, $f$, on the right. Normal triangle culling will produce six triangle pairs: ($f$, $f_i$) for $i$ = 1...6. Only one triangle, $f_2$, represents the vertex $v$. So, this test will be performed only once, when the pair ($f$, $f_2$) is considered. In order for this approach to work, all triangle pairs must be considered. If, for whatever reason, the pair ($f$, $f_2$) were omitted from consideration, the relevant VF test would not be performed. Figure 5 implies that the two portions of the mesh are separated by some arbitrary topology. If the triangle $f_2$ were actually one of the blue triangles in the sub-mesh on the left, our non-adjacent phase would dismiss the triangle pair ($f$, $f_2$). The VF test would not be performed and the collision would be missed–an unacceptable outcome.

Previously, duplicate tests were prevented through the use of a run-time database. For each feature pair to be tested, a look-up would be performed in the database to see if it had already been performed. If not, the test is performed and the pair is included in the database. This has all of the costs normally associated with maintaining such a data structure. Ideally, we'd like to benefit from all of the advantages of R-Triangles.

To do so, we extend the idea of "Representative Triangles" to PRT. Instead of static representation, we provide a dynamic, procedural definition which accommodates our non-adjacent pair constraint. In R-Triangles, the choice of triangle to act as R-Triangle for a vertex is arbitrary. Our non-adjacent constraint can limit this choice. In Figure 5 if face $f_2$ is one of the blue triangles on the left, then $f_2$ cannot serve as R-Triangle for the feature pair ($v$, $f$). But for collision with a different face, distant from those shown in the figure, it would satisfy

**Algorithm 1** VFtest: Testing collision between given vertex $v$ from $t_1$ and face $f$ from $t_2$ by using *PRT*

```
for all Triangle t_i ∈ the incident triangle set of v do
  if t_i not adjacent t_2 then
    if t_i == t_1 then
      DoVFTest(v, f)
    else
      return
    end if
  end if
end for
```

our non-adjacent constraint. This dynamic property means that our procedural definition of PR-Triangles are actually dependent on the actual feature pair being considered.

For PR-Triangles to work, we need to guarantee two properties: completeness and correctness. PR-Triangles must be complete; they cannot miss a collision. They must also be correct; no duplicate queries should be allowed. In other words, every relevant feature test should be performed once and only once. We will show these properties as we describe the functions which determine representation.

**Definition:** For a given feature, there is a set of incident triangles. For a pair of features, the cartesian product between their two incident triangle sets represents all triangle pairs which could request the particular feature test. Because PR-Triangles is dependent, not on a single feature, but the feature pair, we define one PR-Triangle for each feature, simultaneously. The PR-Triangle pair is simply going to be a non-adjacent triangle pair in the cartesian product.

Completeness is guaranteed so long as a non-adjacent pair of triangles appear in the cartesian product. In the non-adjacent phase, we only consider non-adjacent triangle pairs. So, if no other triangle pair in the cartesian product is non-adjacent, there must always exist at least one. (If all triangle pairs in the cartesian product are adjacent, then the feature pair is handled in the following adjacent phase.) So, PR-Triangles is complete.

To be correct we simply need to ensure that no matter how the feature pair is expressed, or the state of the collision detection, the same PR-triangles are always selected. We accomplish this by having some static, topologically-based data structures. Each feature has a list of incident triangles. The order of the list is arbitrary, but fixed for the life of the mesh (barring topological changes.) For vertices, the size of the list is arbitrary and dependent on the fan. For edges, it is either one or two triangles (for well-defined meshes.) By definition a triangle is always incident to itself.

**VF PR-Triangles:** Algorithm 1 shows the process by which PR-Triangles are assigned to a VF feature pair. The face represents itself and the vertex's PR-Triangle is simply the first triangle in its incident list which is non-adjacent. Because the vertex's incident list is in fixed order, it will always examine them in the same order and, with respect to the given face, always find the same PR-Triangles.

**EE PR-Triangles:** Algorithm 2 shows the process by which PR-Triangles are assigned to a EE feature pair. EE feature pairs are simultaneously simpler and more complex. They are simpler because there are at most four triangle pairs in the

**Algorithm 2** EEtest: Testing collision between two edges, an edge $e_1$ from $t_1$ and an edge $e_2$ from $t_2$, by using *PRT*

```
e = min( e_1, e_2 )
E = max( e_1, e_2 )
for all i ∈ [0,1] do
  for all j ∈ [0,1] do
    t_e = ith incident face of e
    t_E = jth incident face of E
    if t_e not exist || t_E not exist then
      continue
    end if
    if t_e not adjacent t_E then
      if (t_e == t_1 & t_E == t_2) || (t_e == t_2 & t_E == t_1)
      then
        DoEETest(e_1, e_2)
      else
        return
      end if
    end if
  end for
end for
```

cartesian product. It's more complex because the PR-Triangles must be invariant to whether the edge pair ($e_i$, $e_j$) or ($e_j$, $e_i$) are examined (both cases can easily arise during typical processes.) For the cartesian product of the EE pair, we need to be able to enforce a fixed order of operation. We define the order as follows: (($e_m$, $E_m$), ($e_m$, $E_M$), ($e_M$, $E_m$), ($e_M$, $E_M$)), where $e$ and $E$ are the edges with the smaller and larger mesh indices, respectively. And the subscripts $m$ and $M$ refer to the incident triangle with the smaller and larger index respectively. The list can be truncated if either edge is incident only to a single triangle. Determining which edge is $e$ and which is $E$ happens at runtime. But the incident lists for each edge can be constructed such that the first incident triangle has the lower index.

**Advantages:** As compared to prior methods [11], [12], PRT have the following advantages:

1) **Lower memory requirements:** Traditional methods for eliminating duplicates involve a run-time database which caches the tests evaluated. With PR-Triangles, there is no need to maintain such a table. The size of the database is theoretically has an $O(n^2)$-bounded size, so this potentially massive memory footprint are eliminated. For example, about 50M memory for the cloth-simulation benchmark (Figure 12) and about 18M memory for the N-body collision benchmark (Figure 11) are saved.
2) **Faster running time:**Because operations to maintain and query the database consist of extensive memory allocation/deallocation operations, the database can provide a significant cost. These system calls are replaced by local query operations so the running speed of the algorithm is improved. It is also worth noting that in the vast majority of cases, the first triangle pair in the cartesian product proves to be non-adjacent and no further tests need be performed. This makes the procedural definition very fast.

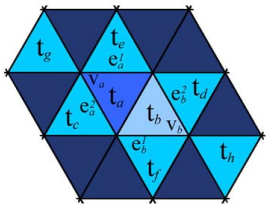


Fig. 6. **Elementary tests for adjacent triangles:** Two adjacent triangles ($t_a$ and $t_b$) share a common edge. Only four elementary tests are necessary: $\{e_b^1, e_a^1\}$, $\{e_b^2, e_a^2\}$, $\{v_a, t_b\}$, and $\{v_b, t_a\}$. Of those four tests, there is a non-adjacent triangle pair which would perform the same tests. They are: $\{t_e, t_f\}$, $\{t_c, t_d\}$, $\{t_g, t_b\}$, and $\{t_h, t_a\}$, respectively.

3) **Better suited to parallelization:**Using PR-Triangles, eliminating duplicate tests becomes a read-only operation. This makes it easier to parallelize as there is no need for explicit synchronization or using locking operations required in maintaining a database.

### C. Adjacent Phase

Many researchers [33], [7], [41] have observed that for a given pair of adjacent triangles, all 15 elementary tests need not be performed for exact collision detection. If two triangles are adjacent (i.e they share an edge or vertex), a lesser subset of the 15 tests is sufficient. Beyond that, others have recognized that the results of the non-adjacent phase could be used to further reduce the number of tests between adjacent pairs [33]. Figure 6 shows a typical example of two edge adjacent triangles: ta and tb. Only four tests are actually meaningful: $\{v_a, t_b\}$, $\{v_b, t_a\}$, $\{e_b^1, e_a^1\}$, and $\{e_b^2, e_a^2\}$.

Now consider the non-adjacent triangle pair ($t_b$, $t_g$). If,during the non-adjacent phase, $t_b$ and $t_g$ have been tested and found not to intersect we can easily argue that the test $\{v_a, t_b\}$ cannot produce a collision. So, the test is unnecessary.

Directly exploiting this relationship requires some form of database to be maintained. The database would store the results of the non-adjacent phase. The adjacent phase would use the database to determine if a particular test on an adjacent pair is necessary. For any particular element pair on an adjacent pair of triangles, the non-adjacent triangle pair, whose collision state could eliminate some of these elementary tests, depends on both the type of element pair and the nature of the adjacency between the adjacent triangles. Each adjacent triangle pair could require multiple unique database queries (as mentioned in [33].)

This idea can be taken much farther and made far more efficient. The costs (in both memory and query time) of the database is unnecessary because the adjacent tests that need to be evaluated depend only on the topology of the mesh. The concept of orphan set formalizes this relationship and provides an optimal set of tests to perform without complex run-time data structures, algorithms, and cache-antagonistic random memory access.

### D. Orphan Pairs and Tests

Overall, the orphan tests are the elementary tests between adjacent triangles that *don't* get performed during the non-adjacent phase. More precisely, an orphan test is a test between

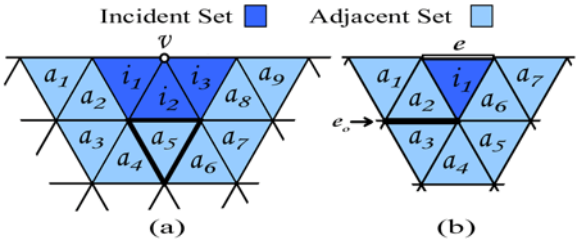


Fig. 7. $OIS$ **and** $OAS$ **for a vertex and an edge:** ($a$) shows the $OIS$ and $OAS$ for the vertex $v$. Vertex $v$ forms an orphan pair with face $a_5$. ($b$) shows the $OIS$ and $OAS$ for the edge $e$. Edge $e$, likewise, forms an orphan pair with (but not limited to) edge $e_o$.

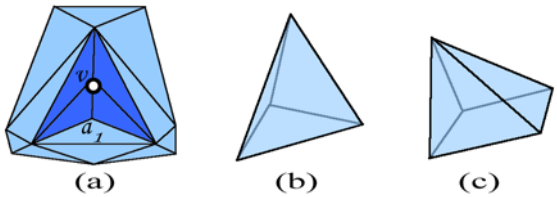


Fig. 8. **Internal orphans:** The element pair in ($a$), ($v, a_1$), is an internal orphan pair. The pyramids in ($b$) and ($c$) consist of nothing but orphan pairs.

an elementary pair (edge-edge or vertex-face) for which no pair of triangles exists such that each triangle is incident to one of the elements and both triangles are non-adjacent. By *incident* triangles, we mean the triangles that include the element in its construction, e.g., the fan of triangles around a vertex. An orphan pair is the pair of elements in the orphan test. The collection of all orphan tests is the orphan set.

*1) Orphan Classification:* Conceptually, there are two types of orphans: boundary and interior. Boundary orphans appear in open manifold meshes. In fact every edge and vertex on the boundary of a mesh are part of one or more orphan pairs. Figure 7 shows two such orphan pairs. Figure 7(a) shows a vertex, $v$, on the boundary of the mesh. The triangle $a_5$ is adjacent to every triangle incident to $v$. There is no non-adjacent triangle pair (including $a_5$) that would execute the test ($v$, $a_5$). So, ($v$, $a_5$) is an orphan pair. Similarly, in Figure 7(b), we see one of the orphan pairs for $e$: ($e$, $e_0$).

Orphans are also possible on the interior of the mesh. There are some special cases, such as a tetrahedron (Figure 8(b)) or four-sided pyramid (Figure 8(c)), in which every triangle is adjacent to every other triangle. Additionally, any ring with a circumference formed by three triangles will have interior orphans. There is a more general condition that can lead to interior orphans–if a triangle in the mesh has vertices on its interior, orphans can also arise (Figure 8(a)).

### E. Orphan Set Computation

The number and location of orphans are strictly a function of the topology and connectivity of the mesh. Deformations that don't change the connectivity in the mesh will leave the set of orphans unchanged. It is sufficient to identify all orphans in a pre-processing step. The algorithms to identify the orphans use two concepts: the Orphan Incident Set ($OIS$) and the Orphan Adjacent Set ($OAS$). The $OIS$ of an element contains all triangles that are incident to that element (e.g. the fan around a vertex.) The $OAS$ of an element is the set of all triangles

adjacent to the triangles in $OIS$ but not in $OIS$. These sets are also known as the one-ring and two-ring, respectively. These sets are illustrated in Figures 7(a) and (b). The detailed algorithms for identifying these orphans can be found in [39].

*1) Topological changes:* In the event of topological changes, such as fracturing or tearing, the orphan set can easily be updated. Changes to the orphan set are limited to the region of the fracture or tear. We form a set consisting of the triangles adjacent to the fracture (i.e. any triangle on the fracture, and the triangles adjacent to those.) We eliminate any orphan test in the orphan set that had an element eliminated by the fracture. Finally, we perform orphan finding algorithms on all of the edges and vertices in the fracture adjacent set.

### F. Processing Orphan Set

The adjacent phase simplifies to evaluating all of the tests in the orphan set. It is possible to perform tests on only a subset of the orphan set. We require some external condition which indicates that a particular orphan pair can't intersect. BV-overlap tests are insufficient for this purpose. Because orphans lie on adjacent triangles, BV-overlap tests won't cull the triangle pair and, therefore, can't cull the orphan pair. However, any test which is immune to this particular adjacency artifact would be sufficient to cull orphan tests.

The CNC test is one such culling mechanism that we exploit. If the CNC test shows that no self-intersection is possible in a region, any orphan pair entirely contained in that region can be summarily dismissed without testing.

Even without such a culling mechanism, the orphan set tends to be quite small relative to the number of possible tests between adjacent triangles.

### G. Completeness and Optimality

It is important to recognize that a system using orphan sets is both complete and optimal with respect to tests between adjacent triangles.

**Completeness:** A collision detection algorithm is complete if no collisions are missed. In CCD all intersection tests among the primitives are reduced to intersections between VF or EE pairs. If the specific VF or EE pair is incident to non-adjacent triangles, it will be checked for overlap in the non-adjacent phase. Otherwise that pair corresponds to an orphan pair and will be evaluated during the adjacent phase. Any VF or EE test that results in a collision will be evaluated. As a result, the orphan set formulation will not miss any collision.

**Optimality:** The orphan set is optimal in the sense that it contains only those tests between adjacent triangles that *cannot* be evaluated in the non-adjacent phase. No other tests between adjacent triangles are necessary. But every pair in the orphan set must be accounted for, whether through direct evaluation or elimination via some technique similar to CNC.

Although the size of the orphan set represents an upper bound on the number of elementary tests between adjacent triangles that need to be explicitly evaluated, the bound tends to be quite small. In our benchmarks, the number of orphan tests are three orders of magnitude smaller than the number of tests which would otherwise be performed between the adjacent triangles (see Table 1).

TABLE I
THIS TABLE SHOWS THE NUMBER OF ELEMENTARY TESTS PERFORMED BETWEEN ADJACENT TRIANGLES. THE FOURTH COLUMN REPRESENTS THE WORK PERFORMED BY [33], IN WHICH 9 ELEMENTARY TESTS ARE PERFORMED FOR VERTEX-ADJACENT PAIRS AND 4 FOR EDGE-ADJACENT PAIRS. THE ORPHAN SET IS SIGNIFICANTLY SMALLER–ROUGHLY 0.1% OF THE TESTS PERFORMED IN [33].

| Model | Tri# | Adjacent pairs | Elementary tests of [33] | Orphan Set |
|---|---|---|---|---|
| Cloth (Figure 12) | $92K$ | $564K$ | $4.4M$ | $4.8K$ |
| Princess (Figure 9) | $40K$ | $269K$ | $2.1M$ | $2K$ |
| N-body (Figure 11) | $34K$ | $198K$ | $1.5M$ | 200 |
| Letters (Figure 13) | $5K$ | $29K$ | $224K$ | 114 |
| Flamenco (Figure 14) | $49K$ | $291K$ | $2.3M$ | $10.8K$ |

## VI. DT-BVH: DYNAMIC TWO-LEVEL BVH

In order to fully utilize our culling algorithms, we use a two-level hierarchy, DT-BVH, and use it for interactive CCD.

The two-level hierarchy is built based on mesh connectivity and bounds on the normals of the triangles. The two-levels consist of the following two parts:

- The first-level of the hierarchy, $HW$, is a k-DOP hierarchy and each of the leaf nodes represent one of the objects $O^i$ and its mesh $M^i$.
- The second-level of DT-BVH represents a k-DOP hierarchy of each mesh, $M^i$. We denote each of these hierarchies as $H^i = BVH(M^i)$. We also maintain a CNC, contour information, and associated orphan pairs for each node in this hierarchy.

In dynamic scenes with changing topologies, the number of objects in the scene may change and we update these hierarchies accordingly.

Please note that CNC is only associated with the nodes of the second-level hierarchy, which contains a connected subset of the mesh $M^i$.

### A. DT-BVH Construction

Based on the definition of DT-BVH, we use a two-level construction algorithm. We consider the connectivity of each mesh, $M^i$, to compute its BVH, $H^i$. We compute an object that is connected component and construct its BVH in a bottom-up manner. Each node of $H^i$ represents a subset of the mesh $M^i$. We compute the CNCs and the boundary edges associated with each node of $H^i$ in a bottom-up manner. Next, we compute the first-level hierarchy, HW, based on the root nodes of each $H^i$ in a top down manner.

### B. Updating DT-BVH

As the models undergo deformation, we update the nodes of DT-BVH. Our goal is to perform the update operation quickly and ensure that the resulting hierarchy provides tight culling efficiency. Again, a two-level approach is used.

- **Refitting $H^i$'s:** We use a simple, linear time refitting algorithm to update the k-DOPs and CNCs of each $H^i$ in a bottom-up manner. The refitting algorithm updates the extents of each k-DOP associated with the nodes of $H^i$. We also compute CNCs of each leaf node as described in

**Algorithm 3** Updating Continuous Normal Cones

```
// Calculate the normal cone for deforming triangle.
if IsLeaf() then
    cone = cone(n_0, n_1, n_0 + n_1 − δ)
    return
end if
// Update the cone of parent node by merging children.
cone = left→cone + right→cone
```

Section IV-B. The CNCs of the intermediate nodes are computed in a bottom-up manner (Algorithm 3), based on the CNCs of their child nodes.

- **Restructuring HW:** Given each updated $H^i$, we use a restructuring algorithm to update HW. Our goal is to compute a tight-fitting BVH. Given scenes with moving or breaking objects, a simple refitting approach may result in a poor hierarchy in terms of culling efficiency. Instead we use a restructuring approach, which regroups some of the primitives in the tree. If the number of objects in the scene is small, we use a simple, top-down rebuilding algorithm of complexity $O(n \log n)$, where $n$ is the number of objects. If the number of objects is high, we perform selective restructuring, as described below.

**Selective Restructuring for Collision Detection:** In order to reduce the restructuring time, we use a selective restructuring algorithm, which restructures localized regions of the hierarchy. Particularly, we identify regions with poor culling efficiency. We use a volumetric metric [26] that measures the culling efficiency of any sub-BVH within the hierarchy. We perform restructuring operations on regions where the restructuring benefit in terms of improved culling efficiency is greater than the cost of restructuring. Each restructuring operation only affects a portion of the tree [26]. This formulation quickly computes a tree with good culling efficiency and can also handle breaking objects.

### C. Continuous Collision Detection using DT-BVH

Our collision algorithm starts with updating DT-BVH, as described above. The collision checking process is started by performing self-collisions on the root node of $HW$. As the recursive algorithm reaches the leaf nodes $H^i$, then self-collision algorithm is invoked on the corresponding $H^i$. For a node of $H^i$ with CNC, we check whether the apex angle of the normal cone is less than $\pi$ and also perform the continuous contour test. If these two tests are satisfied, then, we do not need to traverse deeper to check for self-collisions. And all the orphan pairs associated with the node are marked and will be skipped in orphan tests. The pseudo code description of the algorithm is given in Algorithm 4. Then, elementary tests are used to check for collisions between the leaf nodes of $H^i$ (Please refer to [11] for a detailed algorithm). Finally, we perform orphan tests for all the orphan pairs which not been marked in high-level culling phase.

## VII. Implementation and Performance

In this section, we describe our implementation and highlight the performance of our algorithm on many different benchmarks. We have implemented our algorithm on a standard Intel Pentium 4 with 2.66 GHz and 2GB RAM by using Microsoft Visual Studio 2005. All the timings are generated using a single thread.

**Algorithm 4** SelfCollide(Node $N^i$): Perform self-collision on a node of $H^i$

```
if IsLeaf(N^i) then
    return // Skip leaf nodes.
end if
// Continuous normal cone test.
if TestCNC() == true then
    // Continuous contour test.
    if CCT() == NoIntersection then
        // Associated orphan pais skipped in low-level culling.
        SkipOrphanPairs()
        return // This region is skipped by high-level culling.
    end if
end if
// Check the descendants.
SelfCollide(N^i →left) AND SelfCollide(N^i →right)
Collide(N^i →left, N^i →right)
```

### A. Benchmarks

In order to test the performance of our algorithm, we used six different benchmarks, arising from different simulations with different characteristics.

- **Folding cloth simulation:** We drop a cloth on top of a ball and, then, rotate the ball. It results in a high number of self-collisions in the benchmark consisting of 92K triangles (Figure 12).
- **Princess:** A dancer with flowing skirt ($40K$ triangles) sits on the ground. This model has many inter- and intra-object collisions (Figure 9).
- **N-body collision:** A scene with hundreds of balls (34K triangles) that are colliding with each other (Figure 11). This sequence is generated using a rigid-body simulator.
- **Breaking and deforming letters:** Multiple deforming characters ($5K$ triangles) fall into a bowl and break into pieces (Figure 13).
- **Bunny-Dragon breaking simulation:** We drop a bunny model on top of a dragon model (total $253K$ triangles) and the dragon model breaks into a high number of smaller pieces (Figure 10).
- **Flamenco:** A fiery flamenco dancer wearing colorful skirt with ruffles($49K$ triangles). This benchmark has many inter- and intra-object collisions (Figure 14).

All the benchmarks have multiple simulation steps. We perform continuous collision detection between each discrete steps and compute the first time-of-contact.

### B. Performance

In this section, we analyze the performance of our algorithm. The running time of our algorithm is governed by three steps: updating DT-BVH (performing selective restructuring for breaking models and refitting for simple models, e.g., no

TABLE II
**Performance and Speedup:** THIS TABLE SHOWS THE AVERAGE QUERY TIME OF OUR METHOD AND PERFORMANCE IMPROVEMENT OVER THE BASE IMPLEMENTATION AND GPU-BASED TECHNIQUE OF [35]. PERFORMANCE IMPROVEMENT OVER THE BASE IMPLEMENTATION IS MAINLY DUE TO OUR DT-BVH HIERARCHY REPRESENTATION AND IMPROVED CULLING METHODS.

| Model | Query (time $ms$) | Speedup over: the base impl. | Speedup over GPU-based method |
|---|---|---|---|
| Cloth | 290 | $9X$ | $2.4X$ |
| Princess | 45 | $8.8X$ | $12X$ |
| Flamenco | 185 | $9.4X$ | N/A |
| N-body | 89 | $14.6X$ | N/A |
| Letters | 9.4 | $12.6X$ | $10X$ |
| Dragon | 878 | $21.4X$ | N/A |

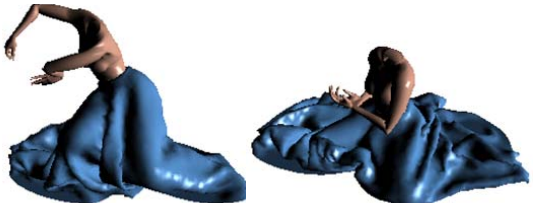

Fig. 9. **Princess benchmark:** A dancer with a flowing skirt. This model has 60K vertices and 40K triangles. Our novel CCD algorithm takes 45ms per frame to compute all the collisions, and is about one order of magnitude faster than prior approaches.

drastic deformations), traversing the DT-BVH, and performing elementary tests.

We assume that the triangles are deforming under linear continuous motion and implement the EE and VF elementary tests used to check triangular prisms for overlap by solving cubic equations. In practice, each such elementary tests takes about 0.2 microseconds on average. Moreover, we perform a planar (E,E) test by performing multiple VE elementary tests. Each VE elementary test reduces to solving a quadratic equation and takes about 0.1 microseconds on average.

In order to demonstrate the benefit of our hierarchical representation and culling techniques, we implemented a "base" version without any of these culling methods. The "base" version also uses an k-DOPs hierarchy computed using refitting algorithms (for models with fixed connectivity) and rebuilding algorithms (for models with changing topologies or breaking objects). We used the same implementation of the elementary tests, using the cubic equation solver from [9], in both of these implementations.

Table II shows the average CCD time of our algorithm and performance improvement over the base method and GPU-based technique [35]. We observe almost one order of magnitude improvement due to the improved culling efficiency.

The main benefits of our algorithm come from the high culling efficiency of the DT-BVH, along with the benefits of the high-level and low-level culling methods. We observe significant reduction in the number of elementary tests (in terms of false positives). Moreover, the time to update the DT-BVH hierarchy is relatively small (at most $5-10\%$ of the total query time). This results in almost one order of magnitude improvement in our benchmarks.

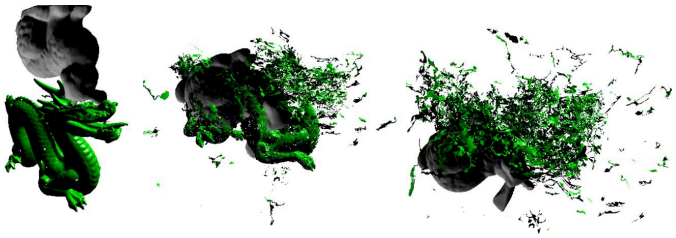

Fig. 10. **Dragon benchmark:** In this simulation, a bunny model is dropped on top of the dragon model and the dragon model breaks into many pieces. This model has 193K vertices and 253K triangles. In this scene with changing topologies, our algorithm obtains high culling efficiency and reduces the number of false positives by 20 times, as compared to prior CCD algorithms. The average CCD query time is about 878ms, about an order of magnitude faster than prior algorithms.

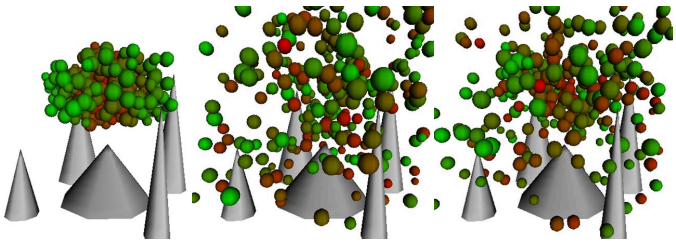

Fig. 11. **N-body benchmark:** In this simulation, multiple balls are colliding with each other. This scene has 18K vertices and 34K triangles. Our culling algorithms reduce the number of elementary test by 18 times and can find all collisions in about 89ms per frame.

## VIII. ANALYSIS AND COMPARISON

In this section we provide in-depth analysis on our results and compare its performance with those of prior methods.

### A. Analysis

**High-level culling & low-level culling:** Figure 15 shows the relative ratio of the number of elementary tests that we perform after each culling step. As shown in the figure, the high-level culling achieves high culling efficiency for deforming models with large relatively flat areas. It can remove the elementary tests among both adjacent triangle pairs and non-adjacent triangle pairs. On the other hand, the low-level culling shows similar culling ratio across different benchmarks. This result is mainly because it relies on the local topological structure of deforming models.

**Bounding volume:** We selected k-DOPs (specifically 18-DOPs) as bounding volumes over AABBs for their superior culling efficiency. Based on our experiments, k-DOPs offer better overall performance for CCD than AABBs. The cost to update the hierarchy is a small fraction of the overall collision query. But the improved culling efficiency yields an overall gain. Please note that this is not the case in discrete collisions or ray tracing. The benefits in terms of fewer false positives with k-DOPs offer a slight net speedup (5%-18%).

**Memory overhead:** The storage overhead of DT-BVH is about 500 bytes per triangle. The memory requirements of our two-level BVH are not optimized in our current implementation and higher as compared to maintaining a single BVH per object. Moreover, we store more information with the intermediate nodes of the second-level BVHs including CNCs, contour, etc.

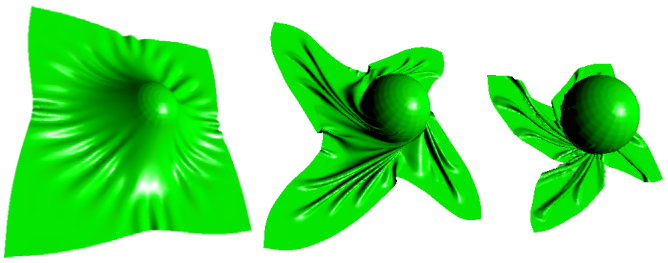

Fig. 12. **Cloth benchmark:** We drop a cloth on top of a rotating ball. This model has 46K vertices and 92K triangles and the simulation results in a high number of self-collision. Our algorithm takes about 290ms on average to perform continuous collision detection. Our culling techniques reduce the number of false positives by 38 times.

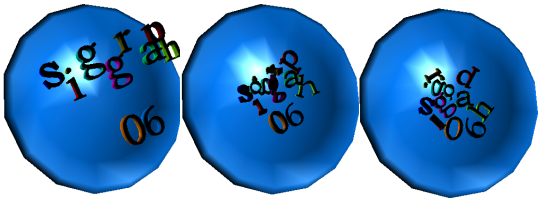

Fig. 13. **Letters benchmark:** Multiple characters interact with a bowl. This model has 3K vertices and 5K triangles.It takes 9.4ms on average for CCD, which is almost 10 times faster than the running time presented in [35].

**Culling efficiency:** Table III shows the improvement in the number of elementary tests performed per frame. The two orders of magnitude improvement is due to our hierarchical representation and culling algorithms.

### *B. Comparison*

**GPU-based accelerations:** The GPU-based algorithms use the rasterization hardware to perform occlusion queries [33] or compute 3D distance fields [35], and readback these fields. Their performance can vary based on the specific GPU and driver implementation. They have been combined with AABB culling to improve the performance of CCD. We compare the performance with the implementation of [35] and observe considerable speedups on some of the benchmarks (up to 10X). As compared to occlusion queries or readbacks, our hierarchy traversal with CNC and contour tests appears to have a lower overhead. For example, Sud et al. [35] reported that their method spends about 50 ms for readbacks. However, our method spends about 45 ms on the whole CCD computation with the Princess benchmark. Furthermore, the low-level culling algorithms significantly reduce the number of elementary tests.

**Kinetic BVHs and updates:** [24], [42] used kinetic BVH and separation lists to reduce the number of updates and tests on the BVH. This is an event-based approach and complementary to our work. We use a single two-level hierarchy for all the objects in the scene as well as new culling algorithms, which appear faster in practice. On the other hand, it becomes harder to maintain the kinetic separation lists efficiently, especially in complex scenes with hundreds of thousands of triangles. As a result, our approach could be faster on such complex scenes, especially with breaking objects.

**Lower-level culling:** Many other authors have also proposed methods to reduce the number of elementary tests between adjacent primitives [33], [7], [41], [10], [12]. Our formulation is more general and achieves higher culling and fewer elementary tests as compared to the prior approaches. We also compared the culling efficiency of our algorithm with that presented in [7]. We observe that our method performs 8.9 times and 7 times fewer elementary tests in the cloth and N-body collision benchmarks, respectively.

Fig. 14. **Flamenco:** A fiery flamenco dancer wearing colorful skirt with ruffles. This model has 25.7K vertices and 49K triangles. Our novel CCD algorithm takes 185ms per frame to compute all the collisions.

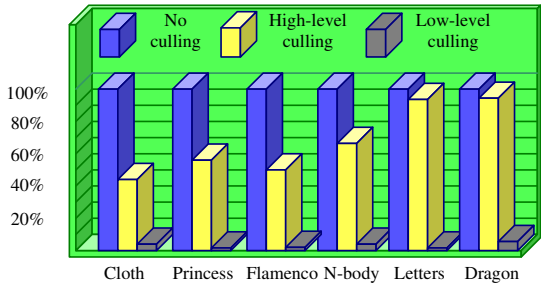


Fig. 15. **High-level culling & low-level culling:** This figure shows the culling efficiency of high-level culling & low-level culling respectively by measuring the ratio of elementary tests performed.

Algorithms [41], [10] can be classified as feature based culling methods. By making some sort of assignment at preprocessing stage, all the replication of elementary test can be naturally solved. In practice, these methods are limited to scenes with fixed connectivity. For breaking scenes with changing topology, it can be inefficient to adjust the assignment dynamically. Our culling method is more general and robust to deal with various kinds for deforming scene.

**Representative Triangles:** [10] proposes "Representative Triangles" to reduce the redundant elementary tests. It can drastically reduce the number of elementary tests. As a downside of this method, due to the randomness of feature distribution, all branches of the BVH need to be traversed to ensure the completeness of elementary tests. This make it hard to integrated with other triangle-based culling method. Our PRT extends above method by distributing elementary tests only among non-adjacent triangle pairs. So it can be used as a further culling method to the high-level culling.

**Adjacency-based culling:** [12] extends the idea of [33] to reduce the number of elementary tests between adjacent triangle pairs. By utilizing a hierarchical method, the collision detection results for non-adjacent triangle pairs are used to cut down the elementary tests among triangle pairs sharing one

TABLE III
**Improved culling efficiency:** THIS TABLE SHOWS THE NUMBER OF ELEMENTARY TESTS PERFORMED PER FRAME BY THE BASE METHOD AND OUR IMPROVED ALGORITHM. THE COMBINATION OF DT-BVH AND IMPROVED CULLING ALGORITHMS REDUCES THE NUMBER OF FALSE POSITIVES BY ALMOST TWO ORDERS OF MAGNITUDE.

| Model | Base implementation | Our algorithm |
|---|---|---|
| Letters | $340K$ | $8K$ |
| Princess | $932K$ | $14K$ |
| Flamenco | $1,125K$ | $16K$ |
| N-body | $3,359K$ | $188K$ |
| Cloth | $7,522K$ | $216K$ |
| Dragon | $16,199K$ | $981K$ |

and only one vertex. Then, these results are further exploited to cut down the elementary tests among triangle pairs sharing an edge. The algorithm is similar to our low-level culling. However, the number of elementary tests in our orphan set is much less than that of [12] and, thus, our algorithm can show better performance.

**Improved normal cone tests:** Most prior work using normal cones has been limited to discrete collision detection. Recently, [37] presented a technique to bound the normals of a mesh for continuous motion, using a "canonical cone". However, their formulation can be rather conservative and inefficient as compared to our fast culling test based on Bernstein basis representation.

### C. Limitations

Our approach has some limitations. First of all, the benefit of our approach is limited by the extent of connectivity in the model. As the objects break into pieces and loses mesh connectivity, the benefit of high-level and low-level culling techniques decreases. Secondly, our normal bounds of CNCs can be quite conservative, especially on models with high curvatures. We observe this in cloth simulation benchmarks, after the cloth folds multiple times.

## IX. CONCLUSION AND FUTURE WORK

We have presented a novel algorithm for CCD between complex deformable models. Our approach is based on a two-level hierarchy and applicable to models arising in different applications, including cloth simulation, breaking objects and N-body simulations. We introduce high-level and low-level culling techniques that significantly reduce the number of false positives. We have tested the performance on different benchmarks and observed considerable improvement in performance over prior CCD algorithms.

There are many avenues for future work. Firstly, we would like to address some of the limitations pointed out in Section VIII-C. Secondly, we want to further improve the performance, especially on scenes with breaking objects that reduce the mesh connectivity. One option would be to develop novel algorithms that can easily utilize the multiple cores on current processors and ensure good cache efficiency. Finally, we would like to integrate our collision detection algorithm into different simulators and use application-specific optimizations to improve the performance.

## ACKNOWLEDGMENTS

We would like to thank Rasmus Tamstorf for the Flamenco benchmark and useful discussions. We thank Stephane Redon for useful discussions and his initial code for elementary tests. We also thank Naga Govindaraju, Avneesh Sud, Russ Gayle and Ming Lin for useful discussions and the benchmarks. Min Tang would thank his wife, Jing Huang, and his child, Tommy, for their support.

This research is supported in part by National Natural Science Foundation of China (No. 60803054), ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134, 0429583 and 0404088, DARPA/RDECOM Contract N61339-04-C-0043, Disney and Intel. Sung-eui Yoon was supported in part by KAIST & LG seed grants, MKE/IITA [2008-F-033-01], MKE/IITA u-Learning, MKE digital mask control, KRF-2008-313-D00922, MSRA E-heritage, and DAPA & ADD contract(UD080042AD). Min Tang is supported in part by National Key Technology R&D Program, China (No. 2006BAF01A45-05), Doctoral subject special scientific research fund of Education Ministry of China (No. 20070335074), and Natural Science Foundation of Zhejiang, China (No. Y107403).

## REFERENCES

[1] S. Redon, A. Kheddar, and S. Coquillart, "Fast continuous collision detection between rigid bodies," *Proc. of Eurographics (Computer Graphics Forum)*, vol. 21, no. 3, pp. 279–288, 2002.
[2] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha, "Fast continuous collision detection for articulated models," in *Proceedings of ACM Symposium on Solid Modeling and Applications*, 2004, pp. 145–156.
[3] M. Foskey, M. Garber, M. Lin, and D. Manocha, "A voronoi-based hybrid planner," *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2001.
[4] L. Zhang and D. Manocha, "Motion interpolation with distance constraints," Department of Computer Science, UNC Chapel Hill, Tech. Rep. TR 08-001, 2008.
[5] R. Bridson, R. Fedkiw, and J. Anderson, "Robust treament for collisions, contact and friction for cloth animation," *Proc. of ACM SIGGRAPH*, pp. 594–603, 2002.
[6] D. Baraff, A. Witkin, and M. Kass, "Untangling cloth," *Proc. of ACM SIGGRAPH*, pp. 862–870, 2003.
[7] M. Hutter and A. Fuhrmann, "Optimized continuous collision detection for deformable triangle meshes," in *Proc. WSCG '07*, 2007, pp. 25–32.
[8] P. Volino and N. M. Thalmann, "Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity," *Computer Graphics Forum (EuroGraphics Proc.)*, vol. 13, no. 3, pp. 155–166, 1994.
[9] X. Provot, "Collision and self-collision handling in cloth model dedicated to design garment," *Graphics Interface*, pp. 177–189, 1997.
[10] S. Curtis, R. Tamstorf, and D. Manocha, "Fast collision detection for deformable models using representative-triangles," in *SI3D '08: Proceedings of the 2008 Symposium on Interactive 3D graphics and games*, 2008, pp. 61–69.
[11] M. Tang, S. Curtis, S. Yoon, and D. Manocha, "Interactive continuous collision detection between deformable models using connectivity-based culling," *Proc. of SPM08 (ACM Solid and Physical Modeling Symposium)*, pp. 25–36, 2008.
[12] M. Tang, S. Yoon, and D. Manocha, "Adjacency-based culling for continuous collision detection," *The Visual Computer, Proc. of CGI08 (Computer Graphics International 2008)*, vol. 24, no. 7-9, pp. 545–553, 2008.
[13] C. Ericson, *Real-Time Collision Detection*. Morgan Kaufmann, 2004.
[14] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino, "Collision detection for deformable objects," *Computer Graphics Forum*, vol. 19, no. 1, pp. 61–81, 2005.
[15] P. M. Hubbard, "Interactive collision detection," in *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, October 1993.

[16] G. Bradshaw and C. O'Sullivan, "Adaptive medial-axis approximation for sphere-tree construction," *ACM Trans. on Graphics*, vol. 23, no. 1, pp. 1–26, 2004.
[17] 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.
[18] S. Gottschalk, M. Lin, and D. Manocha, "OBB-Tree: A hierarchical structure for rapid interference detection," *Proc. of ACM Siggraph'96*, pp. 171–180, 1996.
[19] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *IEEE Trans. on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–37, 1998.
[20] A. Sanna and M. Milani, "CDFast: an algorithm combining different bounding volume strategies for real time collision detection," *SCI Proceedings*, vol. 2, pp. 144–149, 2004.
[21] T. Larsson and T. Akenine-Möller, "A dynamic bounding volume hierarchy for generalized collision detection," *Computers and Graphics*, vol. 30, no. 3, pp. 451–460, 2006.
[22] C. Lauterbach, S. Yoon, D. Tuft, and D. Manocha, "RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs," *IEEE Symposium on Interactive Ray Tracing*, pp. 39–46, 2006.
[23] D. L. James and D. K. Pai, "BD-Tree: Output-sensitive collision detection for reduced deformable models," *Proc. of ACM SIGGRAPH*, pp. 393–398, 2004.
[24] G. Zachmann and R. Weller, "Kinetic bounding volume hierarchies for deforming objects," in *ACM Int'l Conf. on Virtual Reality Continuum and its Applications*, 2006.
[25] M. Otaduy, O. Chassot, D. Steinemann, and M. Gross, "Balanced hierarchies for collision detection between fracturing objects," in *IEEE Virtual Reality*, 2007, pp. 83–90.
[26] S. Yoon, S. Curtis, and D. Manocha, "Ray tracing dynamic scenes using selective restructuring," *Proc. of Eurographics Symposium on Rendering*, 2007.
[27] J. Mezger, S. Kimmerle, and O. Etzmu$\beta$, "Hierarchical techniques in cloth detection for cloth animation," *Journal of WSCG*, vol. 11, no. 1, pp. 322–329, 2003.
[28] P. Volino and N. M. Thalmann, "Accurate collision response on polygon meshes," in *Proc. of Computer Animation*, 2000, pp. 154–163.
[29] B. Heidelberger, M. Teschner, and M. Gross, "Real-time volumetric intersections of deforming objects," *Proc. of Vision, Modeling and Visualization*, pp. 461–468, 2003.
[30] D. Knott and D. K. Pai, "CInDeR: Collision and interference detection in real-time using graphics hardware," *Proc. of Graphics Interface*, pp. 73–80, 2003.
[31] N. Govindaraju, M. Lin, and D. Manocha, "Fast and reliable collision detection using graphics hardware," *Proc. of ACM VRST*, 2004.
[32] B. Heidelberger, M. Teschner, and M. Gross, "Detection of collisions and self-collisions using image-space techniques," *Journal of WSCG*, vol. 12, no. 3, pp. 145–152, 2004.
[33] N. Govindaraju, D. Knott, N. Jain, I. Kabul, R. Tamstorf, R. Gayle, M. Lin, and D. Manocha, "Collision detection between deformable models using chromatic decomposition," *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)*, vol. 24, no. 3, pp. 991–999, 2005.
[34] A. Sud, M. A. Otaduy, and D. Manocha, "DiFi: Fast 3D distance field computation using graphics hardware," *Computer Graphics Forum (Proc. Eurographics)*, vol. 23, no. 3, pp. 557–566, 2004.
[35] A. Sud, N. Govindaraju, R. Gayle, I. Kabul, and D. Manocha, "Fast proximity computation among deformable models using discrete voronoi diagrams," *Proc. of ACM SIGGRAPH*, pp. 1144–1153, 2006.
[36] X. Zhang, S. Redon, M. Lee, and Y. J. Kim, "Continuous collision detection for articulated models using taylor models and temporal culling," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, vol. 26, no. 3, p. 15, 2007.
[37] W. S.-K. Wong and G. Baciu, "Dynamic interaction between deformable surfaces and nonsmooth objects," *IEEE Tran. on Visualization and Computer Graphics*, vol. 11, no. 3, pp. 329–340, 2005.
[38] M. Lin and D. Manocha, "Collision and proximity queries," in *Handbook of Discrete and Computational Geometry*, 2003.
[39] Y. Kim, G. Varadhan, M. Lin, and D. Manocha, "Efficient swept volume approximation of complex polyhedral models," *Proc. of ACM Symposium on Solid Modeling and Applications*, pp. 11–22, 2003.
[40] L.-E. Andersson, N. F. Stewart, and M. Zidani, "Conditions for use of a non-selfintersection conjecture," *Comput. Aided Geom. Des.*, vol. 23, no. 7, pp. 599–611, 2006.
[41] W. S.-K. Wong and G. Baciu, "A randomized marking scheme for continuous collision detection in simulation of deformable surfaces," *Proc. of ACM VRCIA*, pp. 181–188, 2006.
[42] R. Weller and G. Zachmann, "Kinetic separation lists for continuous collision detection of deformable objects," in *Virtual Reality Interactions and Physical Simulation*, 2006, pp. 189–196.

**Min Tang** is an associate professor in the college of computer science at Zhejiang University, China since 2000. He received his B.S., M.S., and Ph.D. from Zhejiang University in 1994, 1996 and 1999 respectively. From June 2003 to May 2004, he was a visiting scholar at Wichita State University. From April 2007 to April 2008, he was a visiting scholar at the University of North Carolina at Chapel Hill. His research interests include geometry modeling, collision detection and GPU-based algorithm acceleration.

**Sean Curtis** received a BA in German from Brigham Young University, a BS in Computer Science from the University of Utah, and a MS in Computer Science from the University of North Carolina, Chapel Hill, where he pursued research in graphics and simulation. He is currently an engineer at Walt Disney Animation Studios.

**Sung-Eui Yoon** is currently an assistant professor at Korea Advanced Institute of Science and Technology (KAIST). He received the B.S. and M.S. degrees in computer science from Seoul National University in 1999 and 2001 respectively. He received his Ph.D. degree in computer science from the University of North Carolina at Chapel Hill in 2005. He was a postdoctoral scholar at Lawrence Livermore National Laboratory. His research interests include scalable graphics and geometric algorithms, interactive rendering, geometric problems, and cache-coherent algorithms and layouts.

**Dinesh Manocha** is currently a Phi Delta Theta/Mason Distinguished Professor of Computer Science at the University of North Carolina at Chapel Hill. He received his Ph.D. in Computer Science at the University of California at Berkeley 1992. He received Junior Faculty Award in 1992, Alfred P. Sloan Fellowship and NSF Career Award in 1995, Office of Naval Research Young Investigator Award in 1996, Honda Research Initiation Award in 1997, and Hettleman Prize for Scholarly Achievements at UNC Chapel Hill in 1998. He has also received more than ten best paper & panel awards at top conferences in graphics, modeling, simulation and visualization. Many of the technologies developed by his group on collision detection, GPU-based algorithms and large model rendering have been widely used. He has published more than 250 papers in leading conferences and journals on computer graphics, geometric modeling, robotics, virtual environments and computational geometry. He has also served as a program committee member for more than 50 leading conferences in these areas and also served in the editorial board of many journals.