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

Min Tang †* Sean Curtis * Sung-Eui Yoon ‡ Dinesh Manocha *
http://gamma.cs.unc.edu/CBC/

## Abstract

We present an interactive algorithm for continuous collision detection between deformable models. We introduce two techniques to improve the culling efficiency and reduce the number of potentially colliding triangle candidate pairs. First, we present a novel formulation for continuous normal cones and use these normal cones to efficiently cull large regions of the mesh from self-collision tests. Second, we exploit the mesh connectivity and introduce the concept of "orphan sets" to eliminate almost all redundant elementary tests between adjacent triangles. In particular, we can reduce the number of elementary tests by many 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 algorithms for deformable models. We highlight the performance of our algorithm on several benchmarks, including cloth simulations, N-body simulations and breaking objects.

**CR Categories:** I.3.5 [Computing Methodologies]: Computational Geometry and Object Modeling—Geometric algorithms; I.3.7 [Computing Methodologies]: Three-Dimensional Graphics and Realism—Visible surface algorithms, animation, virtual reality;

**Keywords:** Continuous collision detection, deformable models, continuous normal cone, orphan set, self-collision

## 1 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 a given time step 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) [Redon et al. 2002; Redon et al. 2004]. CCD techniques model the motion between the discrete time instances using continuous paths and check these paths for collisions. The continuous algorithms are also used to perform time-of-contact computations for dynamic simulation and haptic rendering as well as local planning for sample-based motion planning algorithms [Foskey et al. 2001; Zhang and Manocha 2008].

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 [Bridson et al. 2002; Baraff et al. 2003]. However, performing CCD between complex deformable models at interactive rates still remains a major challenge [Hutter and Fuhrmann 2007]. Prior algorithms for interactive CCD are mainly limited to rigid and articulated models. Many fast algorithms based on bounding volume hierarchies and GPU-based accelerations have been proposed for deformable models, but they are not fast enough for interactive applications.

**Main Results:** We present an interactive algorithm for CCD 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 two novel concepts:

**1. Continuous Normal Cone test for self-collisions:** We extend the well-known normal cone test for discrete collision detection [Volino and Thalmann 1994; Provot 1997] 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 construct normal cone efficiently. Furthermore, we also present a fast continuous contour test method along with the Continuous Normal Cone.

**2. Orphan Set:** We decompose potentially colliding triangle pairs into two sets: non-adjacent and adjacent, to perform the elementary tests between the primitives. We provide an optimal bound on the maximum number of elementary tests required 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 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 collisions during the continuous time domain in tens or hundreds of milliseconds. As compared to prior approaches, our algorithm offers the following benefits:

*University of North Carolina at Chapel Hill, USA
† Zhejiang University, China
‡ Korea Advanced Institute of Science and Technology (KAIST), South Korea

- **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 and reduce the number of false positives by almost 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.

**Organization:** The rest of the paper is organized as follows: Sec. 2 gives a brief survey of prior work in collision detections. We introduce our notation and give an overview of our approach in Sec. 3. The culling techniques are described in Sec. 4 and Sec. 5. We present a two-level hierarchy, overall algorithm, and the results from our implementation in Sec. 6. We compare its performance with prior approaches in Sec. 7.

## 2 Related Work

Collision detection has been widely studied in computer graphics, robotics, and computational geometry literature [Ericson 2004; Teschner et al. 2005]. In this section, we give a brief overview of prior work on collision detection between deformable models.

### 2.1 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 [Hubbard 1993; Bradshaw and O'Sullivan 2004], axis-aligned bounding box (AABB) trees [van den Bergen 1997], hierarchies based on tight fitting bounding volumes (BVs) such as oriented bounding boxes [Gottschalk et al. 1996], discretely oriented polytopes (k-DOPs) [Klosowski et al. 1998], or hybrid combination of BVs [Sanna and Milani 2004].

Most algorithms for deformable models typically use simple BVs such as spheres or AABBs and recompute the BVH during each frame [Teschner et al. 2005]. Approaches to compute dynamic BVHs include refitting algorithms to update these hierarchies [Larsson and Akenine-Möller 2006; Lauterbach et al. 2006; James and Pai 2004; Zachmann and Weller 2006], and performing dynamic or selective restructuring [Otaduy et al. 2007; Yoon et al. 2007].

### 2.2 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. [van den Bergen 1997] presents an early approach using refitting for deformable models.

**Normal Cone Culling:** Volino and Thalmann [Volino and Thalmann 1994] 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 [Mezger et al. 2003; Provot 1997; Volino and Thalmann 2000]. 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 [Heidelberger et al. 2003; Knott and Pai 2003; Govindaraju et al. 2004]. These include many specialized algorithms for self-collision detection [Heidelberger et al. 2004; Govindaraju et al. 2005; Sud et al. 2004; Sud et al. 2006]. Some of these approaches are limited in terms of handling the type 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 7.

### 2.3 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 [Redon et al. 2002] and articulated models [Redon et al. 2004; Zhang et al. 2007] that are based on tight-fitting pre-computed hierarchies. CCD techniques for deformable models [Govindaraju et al. 2005; Wong and Baciu 2005; Hutter and Fuhrmann 2007] are mostly limited to models with fixed connectivity. We compare the performance of our algorithm with these approaches in Section 7.

## 3 Overview

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

### 3.1 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 $t$ 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 breakage. 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].

### 3.2 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-intersection, on the meshes. We assume that we know the position of each vertex in the meshes at every discrete time step. We also assume the position of each vertex, between time steps, is defined

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

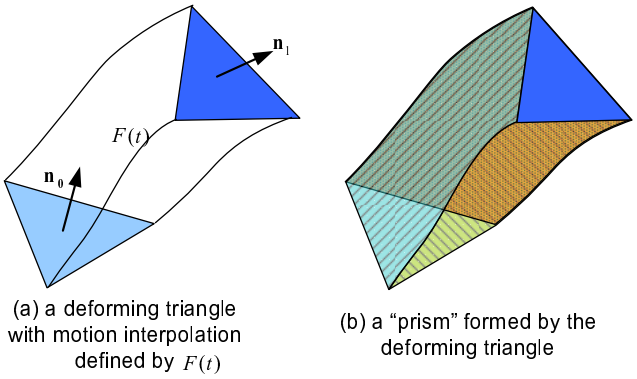


**Figure 1: Triangle under continuous motion through interpolation**: *For the triangle under the motion defined by $F(t)$, its swept trajectory forms a "prism".*

by some continuous interpolation function. Let a polynomial function $F(t)$, of time $t$, represent the interpolating function for vertex position.

In the simplest case, $F(t)$ is a linear function. Detecting collisions between two triangles in motion reduces to performing pairwise vertex-face (VF) and edge-edge (EE) elementary tests [Provot 1997; Bridson et al. 2002]. These VF and EE elementary tests require solving cubic algebraic equations, which are derived from coplanarity 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 $F(t)$ is a polynomial function with degree $d$ ($d > 1$), as shown in Figure 1, the resulting test becomes more complex. We still need to perform 15 VF and EE elementary tests, but the equations we need to have solved would have degree $3d$. For example, if we used cubic spline functions as $F(t)$, 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 an linearly interpolating motion between the discrete instances.

### 3.3 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 two factors:

**1. Culling Efficiency of BVH:** How well a BVH fits the mesh it is supposed to approximate directly affects its culling efficiency. 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 [van den Bergen 1997; Larsson and Akenine-Möller 2006; Zachmann and Weller 2006; Govindaraju et al. 2005], 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 level of the tree and reaches all the leaf nodes of the hierarchy. As a result, these algorithms perform exact collision tests on a high number of triangle pairs, very few of which actually intersect, resulting in a very high number of false positives.

### 3.4 Our Approach

We improve the performance of CCD algorithm by using novel culling techniques and a hierarchical representation which maximizes the effect of those culling techniques. Specifically, 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 cull intersection candidates at two-levels. We first perform *high-level culling* based on a novel continuous normal cone (CNC) formulation. The CNC test consists of the creation 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* by reducing the number of elementary tests. Many authors have observed that there is no need to perform all of the 15 elementary tests between adjacent triangles, i.e. a triangle pair that shares an edge or a vertex [Govindaraju et al. 2005]. We introduce the concept of an orphan set to exploit the topological and connectivity relationships between adjacent triangles. The tests which belong to the orphan set represent, in some sense, the optimal number of elementary tests between adjacent triangles.

## 4 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 methods for self-collisions 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.

Given a continuous surface, S, bounded by a contour, C, Volino and Thalmann [Volino and Thalmann 1994] 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 [Provot 1997] presented an efficient method to implement the first condition based on normal cones.

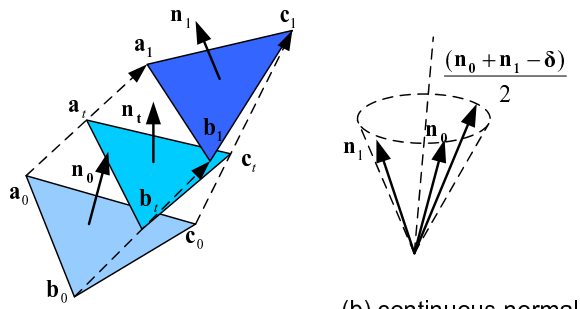


**Figure 2: Continuous normal range of a deforming triangle**: *For a deforming triangle, we construct a CNC that contains* $\mathbf{n_0}$, $\mathbf{n_1}$, *and* $(\mathbf{n_0}+\mathbf{n_1}-\delta)/2$.

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 $t \in [0,1]$.

### 4.1 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 0 and 1, respectively, as shown in Figure 2(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 the vertices of the triangles are under 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* $[0,1]$, *whose positions 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** We define the following terms: $\mathbf{a_t} = \mathbf{a_0} + \vec{\mathbf{v}}_\mathbf{a} \cdot t$, $\mathbf{b_t} = \mathbf{b_0} + \vec{\mathbf{v}}_\mathbf{b} \cdot t$, $\mathbf{c_t} = \mathbf{c_0} + \vec{\mathbf{v}}_\mathbf{c} \cdot t$. The normal vector of triangle $\triangle \mathbf{a_t b_t c_t}$ is given as:

$$\begin{aligned}
\mathbf{n_t} &= (\mathbf{b_t}-\mathbf{a_t}) \times (\mathbf{c_t}-\mathbf{a_t}) \\
&= [(\mathbf{b_0}-\mathbf{a_0}) + (\vec{\mathbf{v}}_\mathbf{b} - \vec{\mathbf{v}}_\mathbf{a})\cdot t] \times [(\mathbf{c_0}-\mathbf{a_0}) + (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{a})\cdot t] \\
&= (\mathbf{b_0}-\mathbf{a_0}) \times (\mathbf{c_0}-\mathbf{a_0}) + (\vec{\mathbf{v}}_\mathbf{b} - \vec{\mathbf{v}}_\mathbf{a}) \times (\mathbf{c_0}-\mathbf{a_0}) \cdot t + \\
&\quad (\mathbf{b_0}-\mathbf{a_0}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{a}) \cdot t + \\
&\quad (\vec{\mathbf{v}}_\mathbf{b} - \vec{\mathbf{v}}_\mathbf{a}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{a}) \cdot t^2. \qquad (1)
\end{aligned}$$

Let $\mathbf{n_0}$ and $\mathbf{n_1}$ be the normal vectors of triangle $\triangle \mathbf{a_0 b_0 c_0}$ and $\triangle \mathbf{a_1 b_1 c_1}$, respectively. Then:

$$\begin{aligned}
\mathbf{n_0} &= (\mathbf{b_0}-\mathbf{a_0}) \times (\mathbf{c_0}-\mathbf{a_0}), \qquad (2) \\
\mathbf{n_1} &= (\mathbf{b_1}-\mathbf{a_1}) \times (\mathbf{c_1}-\mathbf{a_1}) \\
&= (\mathbf{b_0} + \vec{\mathbf{v}}_\mathbf{b} - \mathbf{a_0} - \vec{\mathbf{v}}_\mathbf{a}) \times (\mathbf{c_0} + \vec{\mathbf{v}}_\mathbf{c} - \mathbf{a_0} - \vec{\mathbf{v}}_\mathbf{a}) \\
&= (\mathbf{b_0}-\mathbf{a_0}) \times (\mathbf{c_0}-\mathbf{a_0}) + (\vec{\mathbf{v}}_\mathbf{b} - \vec{\mathbf{v}}_\mathbf{a}) \times (\mathbf{c_0}-\mathbf{a_0}) + \\
&\quad (\mathbf{b_0}-\mathbf{a_0}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{a}) + (\vec{\mathbf{v}}_\mathbf{b} - \vec{\mathbf{v}}_\mathbf{a}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{a}) \qquad (3)
\end{aligned}$$

Based on above equations, we obtain:

$$\begin{aligned}
\mathbf{n_1} - \mathbf{n_0} &= (\vec{\mathbf{v}}_\mathbf{b} - \vec{\mathbf{v}}_\mathbf{a}) \times (\mathbf{c_0}-\mathbf{a_0}) + (\mathbf{b_0}-\mathbf{a_0}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{a}) + \\
&\quad (\vec{\mathbf{v}}_\mathbf{b} - \vec{\mathbf{v}}_\mathbf{a}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{a}). \qquad (4)
\end{aligned}$$

We define:

$$\delta = (\vec{\mathbf{v}}_\mathbf{b} - \vec{\mathbf{v}}_\mathbf{a}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{a}). \qquad (5)$$

Then from equations (4) and (5):

$$\mathbf{n_1} - \mathbf{n_0} - \delta = (\vec{\mathbf{v}}_\mathbf{b} - \vec{\mathbf{v}}_\mathbf{a}) \times (\mathbf{c_0}-\mathbf{a_0}) + (\mathbf{b_0}-\mathbf{a_0}) \times (\vec{\mathbf{v}}_\mathbf{c} - \vec{\mathbf{v}}_\mathbf{a}). \qquad (6)$$

By plugging the equations (2),(5), and (6) into equation (1), $\mathbf{n_t}$ can be represented as:

$$\begin{aligned}
\mathbf{n_t} &= \mathbf{n_0} + (\mathbf{n_1} - \mathbf{n_0} - \delta)\cdot t + \delta \cdot t^2 \\
&= \mathbf{n_0}\cdot(1-t)^2 + (\mathbf{n_0}+\mathbf{n_1}-\delta)/\mathbf{2}\cdot 2t\cdot(1-t) + \mathbf{n_1}\cdot t^2 \\
&= \mathbf{n_0}\cdot B_0^2(t) + (\mathbf{n_0}+\mathbf{n_1}-\delta)/2\cdot B_1^2(t) + \mathbf{n_1}\cdot B_2^2(t).
\end{aligned}$$

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 2(b). Particularly, we use the method proposed in [Provot 1997] to construct an axis and an apex angle of a normal cone from three vectors. Then, the CNCs will be merged as described in [Provot 1997] by traversing the hierarchy in a bottom-up manner while the BVH is refitting.

**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., $F(t)$ is linear). But for higher-order functions, e.g. quadratic, cubic, etc., the CNC equation for other motions, e.g., polynomial interpolation, the CNC equation can be similarly derived. The result would produce a larger set of control points for the higher-order Bernstein functions. When the $F(t)$ is a polynomial function with degree $d$, a Bernstein basis with degree $2d$ will be used to perform the continuous normal cone test.

### 4.2 CCT: Continuous Contour Test

Computing the continuous normal cone covers the first condition for showing no self-intersections. We still require the 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 in the case of discrete collision detection, the contour test can be an expensive operation. Some prior algorithms either omit it under standard geometrical contexts [Volino and Thalmann 1994] or use some approximations [Provot 1997; Mezger et al. 2003]. In this section, we present an exact, but 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 3(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

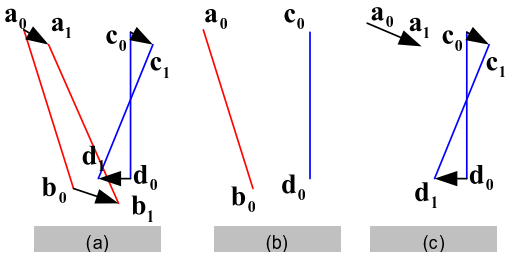


**Figure 3: Planar (E,E) tests :** *Checking whether two co-planar edges* $\mathbf{ab}$ *and* $\mathbf{cd}$ *(shown in (a)) intersect during the interval reduces to discrete line segment intersection test (b) and VE tests (c).*

$t = 1$. 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 $\mathbf{a_0 b_0}$ and $\mathbf{c_0 d_0}$ at $t = 0$ is shown in Figure 3(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 the VE test in the plane. Take the case in Figure 3(c), and it boils down to 4 VE tests that are based on the following combinations: vertex $\mathbf{a}$ with edge $\mathbf{cd}$, vertex $\mathbf{b}$ with edge $\mathbf{cd}$, vertex $\mathbf{c}$ with edge $\mathbf{ab}$, and vertex $\mathbf{d}$ with edge $\mathbf{ab}$. If any of these four tests returns a true value, that implies an intersection between deforming edges $\mathbf{ab}$ and $\mathbf{cd}$.

We use the following theorem to perform a (V,E) 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 3(c). Also, let us define* $\vec{\mathbf{v_a}} = \mathbf{a_1} - \mathbf{a_0}, \vec{\mathbf{v_c}} = \mathbf{c_1} - \mathbf{c_0}$, *and* $\vec{\mathbf{v_d}} = \mathbf{d_1} - \mathbf{d_0}$. *Then, the intersection between the edge and the vertex is governed by the root of the following equation:*

$$(\mathbf{a_0} - \mathbf{d_0}) \times (\mathbf{c_0} - \mathbf{d_0}) + [(\vec{\mathbf{v_a}} - \vec{\mathbf{v_d}}) \times (\mathbf{c_0} - \mathbf{d_0}) + (\mathbf{a_0} - \mathbf{d_0}) \times (\vec{\mathbf{v_c}} - \vec{\mathbf{v_d}})] \cdot \mathbf{t} + (\vec{\mathbf{v_a}} - \vec{\mathbf{v_d}}) \times (\vec{\mathbf{v_c}} - \vec{\mathbf{v_d}}) \cdot \mathbf{t^2} = \mathbf{0}. \quad (7)$$

**Proof** Suppose the deforming vertex overlaps with the deforming edge, then we get

$$(\mathbf{a} - \mathbf{d}) \times (\mathbf{c} - \mathbf{d}) = \mathbf{0}.$$

Based on the equations $\mathbf{a} = \mathbf{a_0} + \vec{\mathbf{v_a}} \cdot \mathbf{t}$, $\mathbf{c} = \mathbf{c_0} + \vec{\mathbf{v_c}} \cdot \mathbf{t}$, and $\mathbf{d} = \mathbf{d_0} + \vec{\mathbf{v_d}} \cdot \mathbf{t}$, we get the equation (7).

After computing the intersection, a trivial test is used to ensure it 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 many acceleration techniques to speed up the computation. Those include:

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 the process of hierarchy traversal.

3. **Fewer intersections:** If two contour edges for a particular node intersect, then any descendant node that has those edges in its contour will also fail the continuous contour test. So, we do not perform the test on children nodes.

These optimizations are easy to implement and can result in considerable speedups in many benchmarks. As an example, we are able to improve the performance of continuous contour test by 93% for the cloth-simulation benchmark (Fig. 11) based on these accelerations.

**Robustness:** As pointed out by Andersson et al. [Andersson et al. 2006], there are several pitfalls to use the criterion of [Volino and Thalmann 1994]. 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 CCT phase. All of these enhance the robustness of our algorithm.

## 5 Orphan Set

In the previous section, we present a test to cull potentially large regions of the mesh from consideration for self-intersection. Ultimately, we must 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.

### 5.1 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, we know that any intersection detected is meaningful. The same can't be said for adjacent pairs.

As we perform these tests between the non-adjacent triangle pairs we will inherently encounter redundant tests since an edge or vertex can be shared by multiple triangles. We use an efficient database structure to record which $(V, F)$ and $(E, E)$ pairs have already been tested. For a given non-adjacent triangle pair we examine each of the 15 elementary tests and determine if that test has already been performed and stored in the database. If it has not, we add it to the database and perform the test.

### 5.2 Adjacent Phase

Many researchers [Govindaraju et al. 2005; Hutter and Fuhrmann 2007; Wong and Baciu 2006] have observed that for a given pair of adjacent triangles, not all 15 elementary tests are necessary. If two triangles are adjacent, then some subset of the 15 elementary tests operate on adjacent elements. Adjacent elements trivially intersect and the results of tests on them would be meaningless.

Beyond that, some have recognized that the results of the non-adjacent phase could be used to further reduce the number of tests between adjacent pairs [Govindaraju et al. 2005]. Figure 4 shows

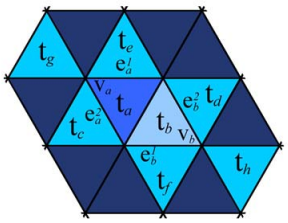


**Figure 4: Elementary tests for adjacent triangles:** *Two adjacent triangles ($t_a$ and $t_b$) share a common edge. Only four elementary tests are necessary: $\{e_b^1, e_a^1\}$, $\{e_b^2, e_a^2\}$, $\{v_a, t_b\}$, and $\{v_b, t_a\}$. Of those four tests, there is a non-adjacent triangle pair which 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.*

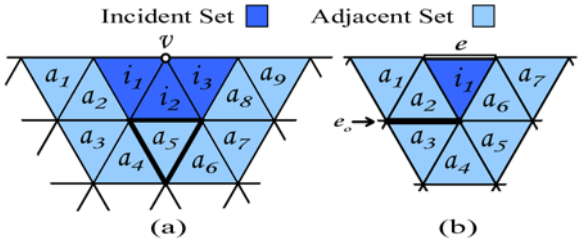


**Figure 5: $OIS$ and $OAS$ for a vertex and an edge:** *(a) shows the $OIS$ and $OAS$ for the vertex $v$. Vertex $v$ forms an orphan pair with face $a_5$. (b) shows the $OIS$ and $OAS$ for the edge $e$. Edge $e$, likewise, forms an orphan pair with (but not limited to) edge $e_o$.*

a typical example of two edge adjacent triangles: $t_a$ and $t_b$. 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 that some form of database 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 justify summary dismissal of the elementary test, is dependent 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 seen in [Govindaraju et al. 2005].)

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

## 5.3 Orphan Pairs and Tests

Simply put, 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 an elementary pair (edge-edge or vertex-face) for which no pair of triangles exists such

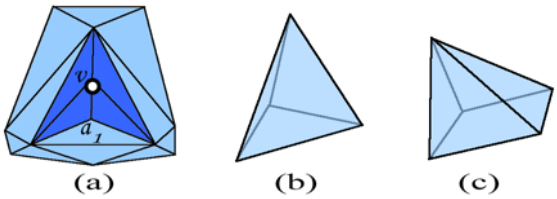


**Figure 6: Internal orphans:** *The element pair in (a), $(v, a_1)$, is an internal orphan pair. The pyramids in (b) and (c) consist of nothing but orphan pairs.*

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.

### 5.3.1 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 will be part of one or more orphan pairs. Figure 5 shows two such orphan pairs. Figure 5(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 5(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 6(b)) or four-sided pyramid (Figure 6(c)), in which every triangle is adjacent to every other triangle. Additionally, any tube 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 6(a)).

## 5.4 Orphan Set Computation

The number and location of orphans are strictly a function of topology. Deformations which don't change 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 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 5(a) and (b).

With these concepts, we can test each edge and vertex to determine which orphan pairs, if any, it belongs in. Algorithms 1 and 2 show how orphans are identified. We simply execute each function on all vertices and edges, respectively.

If we know *a priori* that there are no interior orphans, then we can limit the search to the boundary of the mesh and reformulate the algorithms above in a more optimized form. For example, a mesh produced through Delauney triangulation, won't suffer from vertices on the interior of triangles. These meshes typically don't have interior orphans and creation of the orphan set can be constrained to the boundary elements. In the extreme, if the mesh is a closed manifold with none of the characteristics required for interior orphans,

**Algorithm 1** FindVFOrphan: Find all orphan pairs built on the vertex $v$

```
Create OIS for v
Create OAS for v
for all Triangle t_a ∈ OAS do
    adjacent = TRUE
    for all Triangle t_i ∈ OIS do
        if t_a not adjacent t_i then
            adjacent = FALSE
            break
        end if
    end for
    if adjacent == TRUE then
        Add orphan pair {v, t_a}
    end if
end for
```

**Algorithm 2** FindEEOrphan: Find all orphan pairs built on the edge $e$

```
Create OIS for e
Create OAS for e
Create ES, the set of all edges in OAS
for all Edge e_a ∈ ES do
    if e_a is incident to e then
        continue
    end if
    Create OIS_a for e_a
    adjacent = TRUE
    for all Triangle pair {t_i, t_a}, t_i ∈ OIS ∧ t_a ∈ OIS_a do
        if t_a not adjacent t_i then
            adjacent = FALSE
            break
        end if
    end for
    if adjacent == TRUE then
        Add orphan pair {e, e_a}
    end if
end for
```

the orphan set would be completely empty.

#### 5.4.1 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 which had an element eliminated by the fracture. Finally, we perform Algorithms 1 and 2 on all of the edges and vertices in the fracture adjacent set.

### 5.5 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. 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.

| **Model** | Tri# | Adjacent pairs | Elementary tests of [Govindaraju et al. 2005] | Orphan Set |
|---|---|---|---|---|
| Cloth | $92K$ | $564K$ | $4.4M$ | $4.8K$ |
| Princess | $40K$ | $269K$ | $2.1M$ | $2K$ |
| N-body | $34K$ | $198K$ | $1.5M$ | 200 |
| Letters | $5K$ | $29K$ | $224K$ | 114 |

**Table 1:** *This table shows the number of elementary tests performed between adjacent triangles for various approaches. The fourth column represents the work performed by [Govindaraju et al. 2005], 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 [Govindaraju et al. 2005].*

### 5.6 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).

## 6 Implementation and Performance

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

### 6.1 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.

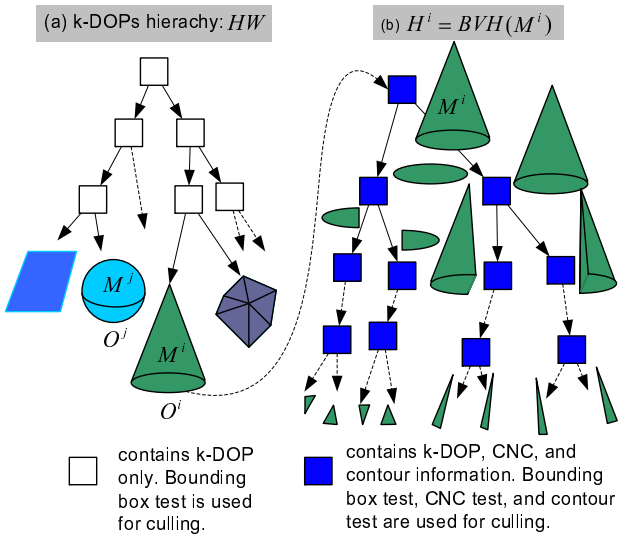


**Figure 7: An example of DT-BVH:** *The first-level is a k-DOPs hierarchy $HW$ shown on the left (a). The leaf node of $HW$ represents a connected mesh. The second-level (shown on the right) hierarchy is build for each connected mesh $M^i$, $H^i = BVH(M^i)$. All the nodes in $H^i$ contain normal cones (CNC) and the contour of the mesh associated with that node.*

The two-level hierarchy is built based on mesh connectivity and bounds on the normals of the triangles. We illustrate the two-level hierarchy, DT-BVH, in Figure 7. The two-levels are:

- 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 and contour information 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 $M^i$.

#### 6.1.1 DT-BVH Construction

Based on the definition of DT-BVH, we use a two-level construction algorithm. We use the connectivity of each mesh, $M^i$, to compute its BVH, $H^i$, 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.

**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, we use a two-level approach to update the hierarchy.

- **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 Section 4.1. The CNCs of the intermediate nodes are computed in a bottom-up manner, 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 [Yoon et al. 2007] 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 [Yoon et al. 2007]. This formulation works well in terms of quickly computing a tree with good culling efficiency and can also handle breaking objects.

**Algorithm 3** SelfCollide(Node $N^i$): Perform self-collision on a node of $H^i$

```
1: if IsLeaf(N^i) then
2:     return // Skip leaf nodes.
3: end if
4: // Continuous normal cone test.
5: if TestCNC() == true then
6:     // Continuous contour test.
7:     if CCT() == NoIntersection then
8:         return // This region can be skipped.
9:     end if
10: end if
11: // Check the descendants.
12: SelfCollide(N^i →left) AND SelfCollide(N^i →right)
13: Collide(N^i →left, N^i →right)
```

#### 6.1.2 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. The pseudo code description of the algorithm is given in Alg. 3. Then, elementary tests are used to check for collisions between the leaf nodes of $H^i$, as shown in Algorithm 4. Finally, we perform orphan tests.

**Algorithm 4** Collide(Node $N_a^i, N_b^i$): Checks for collision between descendants of two nodes of $H^i$.

```
if BoundingBoxTest(N_a^i, N_b^i) == NoOverlap then
    return
end if
// Perform elementary tests on leaf nodes.
if IsLeaf(N_a^i) AND IsLeaf(N_b^i) then
    if N_a^i not adjacent to N_b^i then
        ElmTests(N_a^i, N_b^i) // Perform elementary test.
        return
    end if
end if
// Check the descendants.
if IsLeaf(N_a^i ) then
    Collide(N_a^i, N_b^i →left) AND Collide(N_a^i, N_b^i →right)
else
    Collide(N_a^i →left, N_b^i) AND Collide(N_a^i →right, N_b^i)
end if
```

| Model | Tri # (K) | Query (time $ms$) | Speedup over: the base impl. | Speedup over GPU-based method |
|---|---|---|---|---|
| Cloth | 92 | 290 | $9X$ | $2.4X$ |
| Princess | 40 | 45 | $8.8X$ | $12X$ |
| N-body | 34 | 89 | $14.6X$ | NA |
| Letters | 5 | 9.4 | $12.6X$ | $10X$ |
| Dragon | 253 | 878 | $21.4X$ | NA |

**Table 2: 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 [Sud et al. 2006]. Performance improvement over the base implementation is mainly due to our DT-BVH hierarchy representation and improved culling methods. We observe significant improvement over the base implementation and the GPU-based method of [Sud et al. 2006] due to our improved culling algorithms.*

### 6.2 Benchmarks

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

- **Folding cloth simulation:** We drop a cloth on top of a ball and it curls around resulting in a high number of self-collisions in this 92K triangle model (Fig. 11).
- **Princess:** A dancer with flowing skirt ($40K$ triangles) sits on the ground, resulting in inter- and intra-object collisions (Fig. 8).
- **N-body collision:** A scene with hundreds of balls (34K triangles) that are colliding with each other (Fig. 10). This sequence is generated using a rigid-body simulator.
- **Breaking and deforming letters:** Multiple deforming models of characters ($5K$ triangles) fall into a bowl and break into pieces (Fig. 12).
- **Bunny-Dragon breaking simulation:** We drop a bunny model on top of a dragon model (total $253K$ triangles) and the dragon model decomposes into a high number of smaller pieces (Fig. 9).

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

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

**Table 3: 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.*

### 6.3 Performance

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 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.

**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.

**Culling efficiency:** 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 [Provot 1997] , in both of these implementations. Table 6.2 shows the average CCD time of our algorithm and performance improvement over the base method and GPU-based technique [Sud et al. 2006]. We observe almost one order of magnitude improvement due to the improved culling efficiency.

Table 3 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.

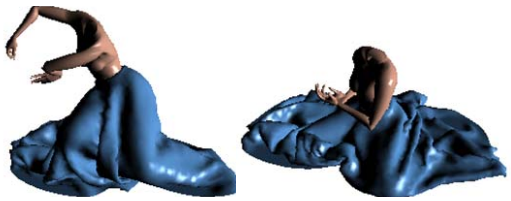

**Figure 8: Princess benchmark:** *A dancer with a flowing skirt. This model has 60K vertices and 40K triangles. Our novel CCD algorithm takes* $45$*ms per frame to compute all the collisions, and is about one order of magnitude faster than prior approaches.*

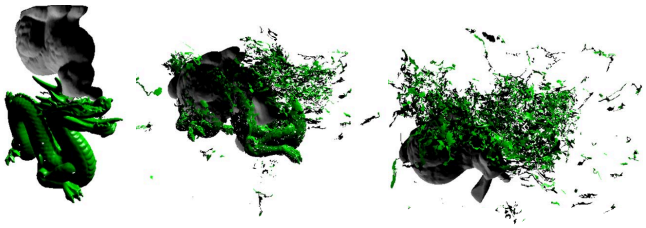

**Figure 9: Dragon benchmark:***In this simulation, a bunny model is dropped on top of the dragon model and the dragon model breaks into many pieces. This model has 193K vertices and 253K triangles. In this scene with changing topologies, our algorithm obtains high culling efficiency and reduces the number of false positives by* $20$ *times, as compared to prior CCD algorithms. The average CCD query time is about* $878$*ms, about an order of magnitude faster than prior algorithms.*

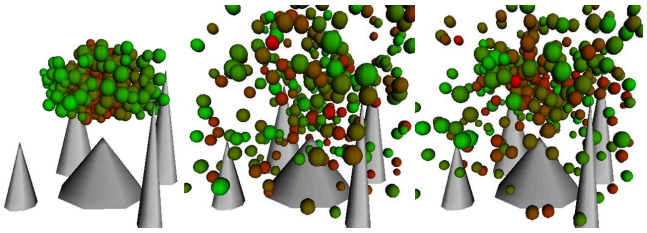

**Figure 10: N-body benchmark:** *In this simulation, multiple balls are colliding with each other. This scene has 18K vertices and 34K triangles. Our culling algorithms reduce the number of elementary test by* $18$ *times and can find all collisions in about* $89$*ms per frame.*

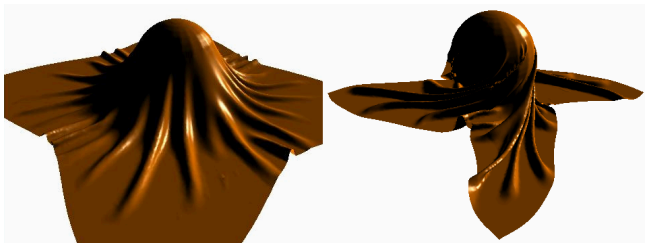

**Figure 11: Cloth benchmark:** *We drop a cloth on top of a rotating ball. This model has 46K vertices and 92K triangles and the simulation results in a high number of self-collision. Our algorithm takes about* $290$*ms on average to perform continuous collision detection. Our culling techniques reduce the number of false positives by* $38$ *times.*

## 7 Analysis and Comparison

In this section, we analyze the performance of our algorithm. 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. In the rest of this section, we compare some of the features and the performance with prior methods.

**GPU-based accelerations:** The GPU-based algorithms use the rasterization hardware to perform occlusion queries [Govindaraju et al. 2005] or compute 3D distance fields [Sud et al. 2006], 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 [Sud et al. 2006] 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. Furthermore, the low-level culling algorithms significantly reduce the number of elementary tests.

**Kinetic BVHs and updates:** [Zachmann and Weller 2006; Weller and Zachmann 2006] 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 [Govindaraju et al. 2005; Hutter and Fuhrmann 2007; Wong and Baciu 2006; Curtis et al. 2008]. 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 [Hutter and Fuhrmann 2007]. We observe that our method performs $8.9$ times and 7 times fewer elementary tests in the cloth and N-body collision benchmarks, respectively. Algorithms [Wong and Baciu 2006; Curtis et al. 2008] 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 is inefficient to adjust the assignment dynamically. While our culling method is more general and more robust to deal with all kinds for deforming scene.

**Improved normal cone tests:** Most prior work in use of normal cones has been limited to discrete collision detection. Recently, [Wong and Baciu 2005] presented a technique to bound the normals of a mesh for continuous motion, using a "canonical cone". However, their formulation is rather conservative and inefficient as compared to our fast culling test based on Bernstein basis representation.

### 7.1 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

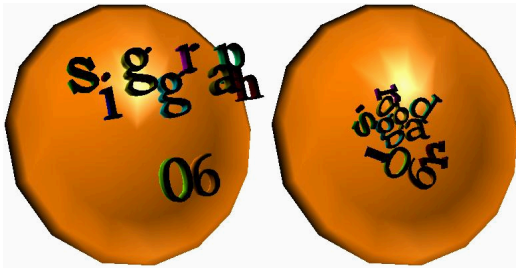

**Figure 12: Letters benchmark:** *Multiple characters interact with a bowl. This model has 3K vertices and 5K triangles.It takes* 9.4*ms on average for CCD, which is almost* 10 *times faster than the running time presented in [Sud et al. 2006].*

the objects break into pieces and loses mesh connectivity, the benefit of high-level and low-level culling techniques decreases. Secondly, our normal bounds CNC test can be quite conservative, especially on models with highly varying curvature. We observe this in cloth simulation benchmarks, after the cloth folds multiple times. Thirdly, our memory requirements can be high due to maintaining various lists.

## 8 Conclusion and Future Work

We present 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 highlighted in Section 7.1. 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 Stephane Redon for many useful discussions and his initial code for elementary tests. We also thank Rasmus Tamstorf, Naga Govindaraju, Avneesh Sud, Russ Gayle and Ming Lin for useful discussions and the benchmarks.

This research is supported in part by 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. Yoon is supported in part by a seed grant from KAIST, ETRI, and IT R&D program of MIC/IITA contract ITAA1100070400010001000300200. Tang is supported in part by National Basic Research Program of China (No. 2006CB303106), Natural Science Foundation of Zhejiang, China (No. Y107403), Doctoral subject special scientific research fund of Education Ministry of China (No. 20070335074), and Future Academic Star fellowship from Zhejiang University.

## References

ANDERSSON, L.-E., STEWART, N. F., AND ZIDANI, M. 2006. Conditions for use of a non-selfintersection conjecture. *Comput. Aided Geom. Des. 23*, 7, 599–611.

BARAFF, D., WITKIN, A., AND KASS, M. 2003. Untangling cloth. *Proc. of ACM SIGGRAPH*, 862–870.

BRADSHAW, G., AND O'SULLIVAN, C. 2004. Adaptive medial-axis approximation for sphere-tree construction. *ACM Trans. on Graphics 23*, 1, 1–26.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treament for collisions, contact and friction for cloth animation. *Proc. of ACM SIGGRAPH*, 594–603.

CURTIS, S., TAMSTORF, R., AND MANOCHA, D. 2008. Fast collision detection for deformable models using representative-triangles. In *SI3D '08: Proceedings of the 2008 Symposium on Interactive 3D graphics and games*, 61–69.

ERICSON, C. 2004. *Real-Time Collision Detection*. Morgan Kaufmann.

FOSKEY, M., GARBER, M., LIN, M., AND MANOCHA, D. 2001. A voronoi-based hybrid planner. *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*.

GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 1996. OBB-Tree: A hierarchical structure for rapid interference detection. *Proc. of ACM Siggraph'96*, 171–180.

GOVINDARAJU, N., LIN, M., AND MANOCHA, D. 2004. Fast and reliable collision detection using graphics hardware. *Proc. of ACM VRST*.

GOVINDARAJU, N., KNOTT, D., JAIN, N., KABUL, I., TAMSTORF, R., GAYLE, R., LIN, M., AND MANOCHA, D. 2005. Collision detection between deformable models using chromatic decomposition. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH) 24*, 3, 991–999.

HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2003. Real-time volumetric intersections of deforming objects. *Proc. of Vision, Modeling and Visualization*, 461–468.

HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2004. Detection of collisions and self-collisions using image-space techniques. *Journal of WSCG 12*, 3, 145–152.

HUBBARD, P. M. 1993. Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*.

HUTTER, M., AND FUHRMANN, A. 2007. Optimized continuous collision detection for deformable triangle meshes. In *Proc. WSCG '07*, 25–32.

JAMES, D. L., AND PAI, D. K. 2004. BD-Tree: Output-sensitive collision detection for reduced deformable models. *Proc. of ACM SIGGRAPH*, 393–398.

KIM, Y., VARADHAN, G., LIN, M., AND MANOCHA, D. 2003. Efficient swept volume approximation of complex polyhedral models. *Proc. of ACM Symposium on Solid Modeling and Applications*, 11–22.

KLOSOWSKI, J., HELD, M., MITCHELL, J., SOWIZRAL, H., AND ZIKAN, K. 1998. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Trans. on Visualization and Computer Graphics 4*, 1, 21–37.

KNOTT, D., AND PAI, D. K. 2003. CInDeR: Collision and interference detection in real-time using graphics hardware. *Proc. of Graphics Interface*, 73–80.

LARSSON, T., AND AKENINE-MÖLLER, T. 2006. A dynamic bounding volume hierarchy for generalized collision detection. *Computers and Graphics 30*, 3, 451–460.

LAUTERBACH, C., YOON, S., TUFT, D., AND MANOCHA, D. 2006. RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. *IEEE Symposium on Interactive Ray Tracing*, 39–46.

LIN, M., AND MANOCHA, D. 2003. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*.

MEZGER, J., KIMMERLE, S., AND ETZMU$\beta$, O. 2003. Hierarchical techniques in cloth detection for cloth animation. *Journal of WSCG 11*, 1, 322–329.

OTADUY, M., CHASSOT, O., STEINEMANN, D., AND GROSS, M. 2007. Balanced hierarchies for collision detection between fracturing objects. In *IEEE Virtual Reality*, 83–90.

PROVOT, X. 1997. Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface*, 177–189.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2002. Fast continuous collision detection between rigid bodies. *Proc. of Eurographics (Computer Graphics Forum) 21*, 3, 279–288.

REDON, S., KIM, Y. J., LIN, M. C., AND MANOCHA, D. 2004. Fast continuous collision detection for articulated models. In *Proceedings of ACM Symposium on Solid Modeling and Applications*, 145–156.

SANNA, A., AND MILANI, M. 2004. CDFast: an algorithm combining different bounding volume strategies for real time collision detection. *SCI Proceedings 2*, 144–149.

SUD, A., OTADUY, M. A., AND MANOCHA, D. 2004. DiFi: Fast 3D distance field computation using graphics hardware. *Computer Graphics Forum (Proc. Eurographics) 23*, 3, 557–566.

SUD, A., GOVINDARAJU, N., GAYLE, R., KABUL, I., AND MANOCHA, D. 2006. Fast proximity computation among deformable models using discrete voronoi diagrams. *Proc. of ACM SIGGRAPH*, 1144–1153.

TESCHNER, M., KIMMERLE, S., HEIDELBERGER, B., ZACHMANN, G., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., STRASSER, W., AND VOLINO, P. 2005. Collision detection for deformable objects. *Computer Graphics Forum 19*, 1, 61–81.

VAN DEN BERGEN, G. 1997. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools 2*, 4, 1–14.

VOLINO, P., AND THALMANN, N. M. 1994. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum (EuroGraphics Proc.) 13*, 3, 155–166.

VOLINO, P., AND THALMANN, N. M. 2000. Accurate collision response on polygon meshes. In *Proc. of Computer Animation*, 154–163.

WELLER, R., AND ZACHMANN, G. 2006. Kinetic separation lists for continuous collision detection of deformable objects. In *Virtual Reality Interactions and Physical Simulation*, 189–196.

WONG, W. S.-K., AND BACIU, G. 2005. Dynamic interaction between deformable surfaces and nonsmooth objects. *IEEE Tran. on Visualization and Computer Graphics 11*, 3, 329–340.

WONG, W. S.-K., AND BACIU, G. 2006. A randomized marking scheme for continuous collision detection in simulation of deformable surfaces. *Proc. of ACM VRCIA*, 181–188.

YOON, S., CURTIS, S., AND MANOCHA, D. 2007. Ray tracing dynamic scenes using selective restructuring. *Proc. of Eurographics Symposium on Rendering*.

ZACHMANN, G., AND WELLER, R. 2006. Kinetic bounding volume hierarchies for deforming objects. In *ACM Int'l Conf. on Virtual Reality Continuum and its Applications*.

ZHANG, L., AND MANOCHA, D. 2008. Motion interpolation with distance constraints. Tech. Rep. TR 08-001, Department of Computer Science, UNC Chapel Hill.

ZHANG, X., REDON, S., LEE, M., AND KIM, Y. J. 2007. Continuous collision detection for articulated models using taylor models and temporal culling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007) 26*, 3, 15.