Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2010)
M. Otaduy and Z. Popovic (Editors)

FASTCD: Fracturing-Aware Stable Collision Detection

Jae-Pil Heo[1], Joon-Kyung Seong[1], DukSu Kim[1], Miguel A. Otaduy[2], Jeong-Mo Hong[3], Min Tang[4], Sung-Eui Yoon[1]

[1] KAIST [2] URJC Madrid [3] Dongguk Univ. [4] Zhejiang Univ.

Abstract

We present a collision detection (CD) method for complex and large-scale fracturing models that have geometric and topological changes. We first propose a novel dual-cone culling method to improve the performance of CD, especially self-collision detection among fracturing models. Our dual-cone culling method has a small computational overhead and a conservative algorithm. Combined with bounding volume hierarchies (BVHs), our dual-cone culling method becomes approximate. However, we found that our method does not miss any collisions in the tested benchmarks. We also propose a novel, selective restructuring method that improves the overall performance of CD and reduces performance degradations at fracturing events. Our restructuring method is based on a culling efficiency metric that measures the expected number of overlap tests of a BVH. To further reduce the performance degradations at fracturing events, we also propose a novel, fast BVH construction method that builds multiple levels of the hierarchy in one iteration using a grid and hashing. We test our method with four different large-scale deforming benchmarks. Compared to the state-of-the-art methods, our method shows a more stable performance for CD by improving the performance by a factor of up to two orders of magnitude at frames when deforming models change their mesh topologies.

1. Introduction

Simulating realistic fractures in deforming models is one of the main challenges in computer animation, sculpting in CAD, virtual surgery, etc [OH99, SOG06, BHTF07, WTGT09, PO09]. Simulating such complex phenomena requires collision detection (CD) methods to avoid any inter-collisions among deforming models and self-collisions (i.e. intra-collisions) within each deforming model. Moreover, fractures change the mesh connectivity (i.e. mesh topology), in addition to changing the geometry (e.g., positions of vertices) of the mesh. Also, at a fracture or merge event, multiple parts of the same object can appear in a close proximity (see Fig. 1), causing a higher computation time for CD. As a result, CD including self-collision detection is typically the main computational bottleneck of simulating these complex phenomena [SOG06, BHTF07].

CD methods are commonly accelerated by using bounding volume hierarchies (BVHs) constructed from deforming models. The BVHs are hierarchically traversed to find collisions among models. At the leaf nodes of BVHs, primitive tests are performed between triangles stored in these leaf nodes. In these BVH-based CD methods, detecting self-collisions requires much longer computation time than detecting inter-collisions [GKJ*05]. This is mainly because bounding volumes (BVs) of any neighboring triangles can overlap and BVHs do not provide any culling for these overlapping BVs during the self-collision detection.

Moreover, these BVHs of deforming meshes should be updated as deforming meshes change their geometry and topology. At fracturing events, the geometry and topology of deforming models undergo more drastic changes, compared to deforming events that do not have any fractures. Therefore, BVHs at fracturing events become to have lower culling efficiencies, degrading the performance of CD more significantly. As a result, users may experience noticeable performance degradations at such fracturing events. Especially in interactive applications, providing a stable performance to users is very important. Since the performance of CD for large-scale fracturing simulations can be very slow, such large-scale fracturing has not been widely employed in various interactive applications.

At a high level, CD is classified into discrete CD (DCD) and continuous CD (CCD) [LM03]. DCD finds collisions only at discrete time steps. On the other hand, CCD models a continuous motion for deforming geometry between two discrete time steps and detects collisions at the first time-of-contact in the continuous time interval defined by these two discrete time steps. It is well known that CCD is more expensive than DCD and thus presents more challenges for achieving interactive performances with fracturing models.

To improve the performance of DCD and CCD, many prior approaches have been proposed. They accelerate the performance of CD by designing specialized algorithms for certain types of models (e.g., rigid objects [RKC02] and meshes with fixed topology [GKJ*05]), developing various culling methods [VT94, CTM08, TCYM08], designing efficient BVH update methods [LAM06, OCSG07], or introducing parallel CD methods [KHH*09, TMT09]. However, most of these prior methods have not been tested with fracturing models that have topological changes. Moreover, if we apply these techniques to such models that consist of hundreds of thousands of triangles, they may take prohibitive computation time at fracturing events in practice.

Main results: In this paper we propose a **F**racturing-**A**ware **ST**able **CD** (FASTCD) method for complex and large-scale fracturing models that have geometric and topological changes. First, we introduce a *Dual-Cone Theorem* (Sec. 4) that allows us to check whether a surface can have self-collisions or not. The dual-cone consists of surface normal and binormal cones. We design a BVH-based hierarchi-

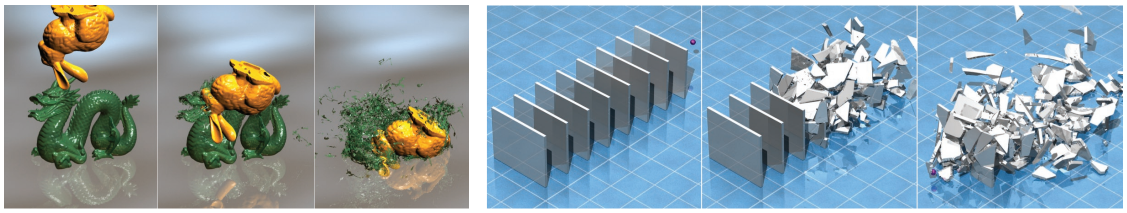

Figure 1: *These figures show two complex fracturing benchmarks that have topological changes. (a) Three frames of a breaking dragon benchmark that consists of 252 K triangles throughout the simulation. (b) Three frames of a breaking-wall benchmark that starts with 42 K triangles and ends with 140 K triangles.*

cal culling method that uses dual-cones for each node. Our dual-cones can be efficiently updated even when deforming models have drastic geometric and topological changes. Our dual-cone theorem is conservative. However, our BVH-based CD method integrated with dual-cones results in an approximate culling method and could miss collisions in theory. However, we found that our method does not miss any collisions in the tested benchmarks. Second, we propose a selective restructuring method that locally modifies only sub-BVHs that have low culling efficiencies (Sec. 5). To measure the culling efficiency of a BVH, we propose a novel cost metric that measures the expected number of BV overlap tests that are performed recursively for CD including self-collision detection. Also, in order to reduce performance degradations at fracturing events, we propose a fast BVH construction that builds multiple levels of the hierarchy in one iteration by using a grid and hashing.

To demonstrate the benefits of our method, we test discrete and continuous CD methods integrated with our methods in four different complex deforming benchmarks that consist of hundreds of thousands of triangles (Sec. 6). Compared to the state-of-the-art techniques, our method improves the performance for CD including self-collision detection by a factor of up to two orders of magnitude at fracturing events when deforming meshes change their topology. This results in a more stable performance of CD with large-scale fracturing models.

2. Related Work

CD has been widely studied and excellent surveys are available [LM03, Eri04, TKH*05].

2.1. Bounding Volume Hierarchies (BVHs)

BVHs have been widely used to accelerate the performance of CD. Some of the common bounding volumes (BVs) include spheres, axis-aligned bounding boxes (AABBs), etc [LM03, Eri04]. Many top-down and bottom-up techniques have been proposed to construct these BVHs from meshes [TKH*05].

BVH update methods: As models deform, their BVHs should be updated. The simplest update method refits each BV by traversing the BVH in a bottom-up manner [TKH*05]. This method runs quite fast, but shows lower culling efficiencies as models deform drastically. Another class of BVH update methods reconstructs the entire BVH from scratch [WH06]. This method shows the best culling efficiency, but runs quite slow for large-scale models. The last, but the most widely used class of BVH update methods selectively restructures subsets of the BVHs [LAM06]. For fracturing models, Otaduy et al. [OCSG07] proposed an AVL-based local restructuring method in order to compute balanced BVHs. Zachmann and Weller [ZW06] proposed event-based kinetic BVHs. This method showed an optimal number of BVH updates for deforming models that have fixed topologies.

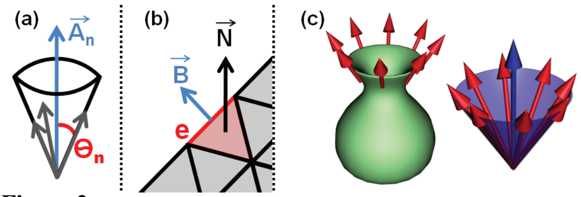

Figure 2: *This figure shows (a) a cone definition, (b) a computed binormal vector from a boundary edge, and (c) an example of a binormal cone computed from a surface.*

Time-critical methods: In order to achieve interactive performance of CD for large-scale models, many approximate CD methods have been proposed. Hubbard [Hub93] introduced the concept of time-critical CD using sphere-BVHs. Many other approximate techniques [OD01, OL03, YSLM04] have been introduced based on the concept of time-critical CD, to provide more stable CD performances at fracturing events.

2.2. Culling Techniques for CD

Volino and Thalmann [VT94] proposed a culling condition to identify regions that do not have any self-collisions for a connected mesh at discrete time steps. This culling method can be combined with BVH-based CD methods [Pro97] and has been extended to CCD [TCYM08].

Another class of culling methods aims to reduce the number of primitive tests for CCD by considering the connectivity of the meshes. These techniques are initially designed for meshes with fixed connectivity [GKJ*05] and are extended to deforming models that have fixed topologies [CTM08, TCYM08].

3. Background

In this section we define terminologies for the rest of the paper and introduce the background of our culling method.

Terminologies: We define a sub-BVH (n) to denote a sub-BVH rooted at the BV node n. Also, n_l and n_r represent the left and right child nodes of the node n. We use a cone to compute the tightest directional bounds of vectors. A cone of vectors is constructed such that the cone (e.g., the violet cone in Fig. 2-(c)) contains all the vectors (e.g., red vectors in Fig. 2-(c)), when the origin of each vector is anchored to a point. We define a cone $C(\vec{A_n}, \theta_n)$ by $\vec{A_n}$, an axis, and θ_n, half of the apex angle of the cone (Fig. 2-(a)). Unless mentioned otherwise, the angle of a cone refers to θ_n. We also define a binormal vector $\vec{B}$ of a boundary edge e of a triangle to be the cross product between the surface normal $\vec{N}$ of the triangle and the boundary edge e. In particular, we assume that the binormal vector $\vec{B}$ points outside the triangle (Fig. 2-(b)). Ideally, a cone is valid, only if the angle of the cone is less than $\frac{pi}{2}$. However, we loosely use the term of a cone, even if it has an angle that is equal or bigger than $\frac{pi}{2}$, for the sake of compact explanations.

3.1. BVH-based CD

For BVHs of deforming models, we merge these BVHs into a single BVH and then perform our CD method with the merged BVH. In this case, inter-collisions among multiple objects and self-collisions within each object can be computed by performing self-collision detection with the merged BVH [TKH*05]. We initially perform a BV overlap test between two child nodes of the root node of the BVH. If there is a collision between two nodes, we refine each of them and perform overlap tests between the refined nodes. We perform this process recursively until there are no collisions or we reach leaf nodes. For leaf nodes, we perform primitive tests between triangles stored in the leaf nodes.

3.2. Sufficient Conditions for Self-Colliding Surfaces

To improve the performance of self-collision detection, which is the most time consuming part of CD for deforming models, Volino and Thalmann [VT94] proposed that if both of the following two conditions are satisfied for a given continuous surface S bounded by a contour C, the surface does not have any self-collisions:

- **Surface normal test:** There is a vector $\vec{V}$ such that $(\vec{N} \cdot \vec{V}) > 0$ for every point on the surface S, where $\vec{N}$ is the normal vector for a point on the surface.
- **Contour test:** The projection of the contour C along the vector $\vec{V}$ does not have any self-intersection on the projected plane.

Provot [Pro97] proposed an efficient method to check the first condition using normal cones that bound surface normal vectors. However, the contour test has $O(N^2)$ time complexity at the worst case, where N is the number of projected edges in the projected plane. Tang et al. [TCYM08] extended these two conditions to CCD by proposing the continuous normal cone theorem and continuous contour tests. They also proposed various optimization methods that improve the performance of the continuous contour tests, by assuming that the topologies of deforming models are fixed and precomputing various data. However, their pre-computations can cause significant performance degradations whenever deforming models change their topologies, as empirically verified in Sec. 6.

4. Dual-Cone Method

In this section we present our culling method using dual-cones for self-collision detection.

4.1. Culling Conditions using Dual-Cone

We first derive a culling condition that can identify regions that do not have any self-collisions for planar surfaces, followed by an extension to 3-D surfaces.

For the sake of simplicity, we assume that input meshes for our culling method are 2-dimensional manifolds with boundaries. Our method is naturally applicable to triangular meshes including non-manifold triangular meshes, by decomposing non-manifold meshes into a set of manifold meshes and ignoring any collisions among the decomposed mesh boundaries.

We use two cones, the surface normal and binormal cones, for a surface S to check whether the surface can have self-collisions. The surface normal cone, SNC $(\vec{A_n}, \theta_n)$, bounds all the surface normal vectors of the surface S. The binormal cone, BNC $(\vec{A_b}, \theta_b)$, bounds all the binormal vectors of the boundary of the surface S. Note that the angle of the surface normal cone of a planar surface is either 0 or $\frac{\pi}{2}$ (i.e. the surface has fold-overs).

Theorem 4.1 (Self-Colliding Planar Surface) Given a planar surface S that has SNC $(\vec{A_n}, \theta_n)$ and BNC $(\vec{A_b}, \theta_b)$, the surface S that has self-collisions satisfies at least one of the following two conditions:

$$\theta_n = \pi/2, \quad (1)$$
$$\theta_b \geq \pi/2. \quad (2)$$

Proof: Let us suppose that the first condition (Eq. 1) is false, i.e. normals of the surface point in one direction. In this case, we will show that the second condition (Ineq. 2) is satisfied.

We define a planar minimal self-colliding surface to be a planar self-colliding surface, whose self-collisions occur only at the boundaries of the surface. One can compute such a planar minimal self-colliding surface S^m from the planar surface S, by trimming the surface S. If we prove that the second condition is satisfied for S^m, then the second condition naturally holds for S, since S^m is a sub-surface of S and the angle of the binormal cone of S is at least the angle of the binormal cone of S^m. For the sake of simplicity, we assume that there is a single self-colliding point on S^m; our proof can be easily extended to planar minimal self-colliding surfaces with multiple self-collisions.

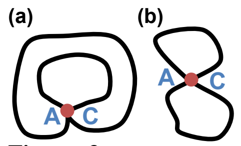

Figure 3: *Planar minimal self-colliding surfaces*

Let two surface points, A and C, be the two colliding points on the boundary of the surface S^m. Since the surface S^m self-collides only at A and C, the surface has two different loops and these two loops do not cross each other in any other points except for A and C. There are only two cases that satisfy this configuration: one of these two loops is inside the other loop (Fig. 3-(a)) or not (Fig. 3-(b)). Note that these two loops are not continuous at A and C; they are *piecewise regular*.

If we compute a binormal cone for a continuous loop, the angle of the binormal cone is π. However, in our piecewise regular loops, the maximum angle deviation between two binormal vectors at these discontinuous points, A and C, is less than π [dC76]. Therefore, the binormal cone for each piecewise regular loop has an angle that is equal to or bigger than $\pi/2$. Due to a space limit, we provide this intuitive derivation that shows the binormal cone of each piecewise regular loop has such angle. Nonetheless, this result can be more formally derived from the *Turning Tangent* theorem [dC76]. □

We now present a lemma, followed by our main theorem, which serves as a culling condition that can check whether 3-D surfaces cannot have any self-collisions.

Lemma 4.1 (Projection of a Cone) An orthographic projection of a cone, $C(\vec{A},\theta)$, along a direction $\vec{P}$ results in a projected 2-D cone, whose angle is less than $\pi/2$ if and only if $|\vec{P}\cdot\vec{A}| < \cos\theta$.

The bi-implication of the Lemma can be easily shown by choosing $\vec{P}$ inside and outside the cone. What Lemma 4.1 indicates is that the projection vector, $\vec{P}$, must come from outside of the cone, in order for the projected cone to span less than $\pi/2$.

Theorem 4.2 (Dual-Cone Theorem) Given a 3-D surface S that has SNC $(\vec{A_n},\theta_n)$ and BNC $(\vec{A_b},\theta_b)$, the surface S that has self-collisions satisfies at least one of the following two conditions:

$$\theta_n \geq \pi/2, \tag{3}$$

$$|\vec{A_n}\cdot\vec{A_b}| \geq \cos\theta_b. \tag{4}$$

Proof: Assume that neither two conditions are satisfied for the 3-D surface S that has self-collisions. Since the angle of the normal cone of the surface is less than $\pi/2$ and an orthographic projection of the surface S in the direction of $\vec{A_n}$ is surjective [dC76], the planar surface projected from the surface S is a self-colliding planar surface, whose surface normal vectors point in one direction; hence it negates the first condition (Eq. 1) of Theorem 4.1. Also, since the second condition (Ineq. 4) is not satisfied either, the binormal cone of the projected surface should span less than $\pi/2$, according to Lemma 4.1; this also negates the second condition (Ineq. 2) of Theorem 4.1. Since both conditions of Theorem 4.1 are not satisfied, we can apply the contraposition of Theorem 4.1. Therefore, we can conclude that the planar surface projected from the surface S does not have any self-collisions, which leads to a contradiction. □

The dual-cone theorem (Theorem 4.2) identifies two necessary conditions for a 3-D surface that has self-collisions. Therefore, the contraposition of the theorem provides a culling algorithm that detects regions that cannot have any self-collisions. In other words, if $(\theta_n < \pi/2)$ and $(|\vec{A_n}\cdot\vec{A_b}| < \cos\theta_b)$, then we can decide that the surface cannot have any self-collisions.

Our dual-cone culling algorithm tests two simple inequalities for a connected 3-D surface. However, our dual-cone method is a very conservative culling technique; it may not cull meshes although they do not have any self-collisions. For example, the angle of the binormal cone of a planar surface with a boundary always spans more than $\pi/2$, preventing any cullings for the surface, according to Theorem 4.2. Therefore, in order to achieve a high culling ratio, we propose an approximate culling technique that integrates our dual-cone method with a BVH-based CD method.

4.2. BVH-based Culling

Our dual-cone culling method can be easily applied to the BVH-based CD method mentioned in Sec. 3.1. For this, we compute the surface normal and binormal cones for a sub-mesh contained in each BV. Also, a (surface normal or binormal) cone of an intermediate BV node can be computed by merging two cones of its two child BV nodes [Pro97]. As a result, once we compute dual-cones for leaf nodes of a BVH, we can compute dual-cones of intermediate nodes by traversing the BVH in a bottom-up manner. Therefore, dual-cones associated with the BVH can be quite efficiently updated even for fracturing models that have topological changes.

At runtime, during the BVH traversal, we test our dual-cone culling method for each BV node and may cull a sub-mesh contained in the node from self-collision detection. Although an input mesh has one connected component, a sub-mesh contained in a BV node may have multiple components. Therefore, we pre-compute whether a sub-mesh contained in a BV node is connected as one component by traversing the sub-mesh during the BVH construction and record the information in the node. If the node has multiple components, we do not perform our culling method on the node during the BVH-based CD.

Internal boundary edges: The BVH-based CD method combined with our method mentioned above is an approximate approach, i.e. it may miss collisions. Suppose that we have a penetrating pipe (Fig. 4-(a)). To build a BVH for the object, we may partition triangles of the object with a horizontal partitioning plane as shown in Fig. 4-(a) and then compute two child BVs with two partitioned sub-objects. Because of the high curvature in the original object, we cannot cull the object and check two child BVs that contain the partitioned sub-objects. However, since the lower child BV (shown in a pink color in Fig 4-(a)) contains a near-flat connected sub-object (Fig. 4-(b)), whose boundary (shown in red thick curves) is also near-flat, our dual-cone method culls the BV from self-collision detection, although there are self-collisions within the partitioned sub-object. This inaccurate culling is caused by ignoring *internal* boundary edges (shown in blue thick curves in Fig. 4-(b)) that are incident on two triangles that are partitioned to two different BVs. Our BVH-based CD method does not check any collisions among them, since they are not in the original boundary of the object.

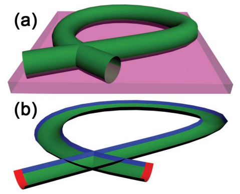

Figure 4: *A penetrating pipe*

To address this problem, we can also compute separate dual-cones for these internal boundaries and design a conservative BVH-based CD method. However, maintaining such internal boundary edges and additional dual-cones can significantly lower the performance. Note that the example configuration that leads our method to miss collisions occurs very rarely. This is because if we have complex internal boundaries formed by a partitioning plane, it is likely that we have a high curvature for the partitioned objects in many cases. Moreover, we found that our simple BVH-based CD method without considering these internal boundaries does not miss any collisions in our tested benchmarks. Therefore, we decide to use the simple BVH-based CD method without any modifications.

As a result, our method has $O(1)$ time complexity, but is approximate in theory. This is a different design choice from the exact contour test of Volino and Thalmann's method that has a quadratic worst-case time complexity. Also, one can ignore internal boundary edges and perform the exact contour tests only for real boundary edges. However, this method still has the non-linear time complexity, although it can run faster than the original Volino and Thalmann's method. Moreover, the dual-cones of our method can be updated quite efficiently even for fracturing models. We compare these different techniques including our method and demonstrate the performance benefits of our method with fracturing models in Sec. 6.2.

4.3. Extension to CCD

To extend our culling method to CCD, we need to compute the surface normal and binormal cones with the deforming triangles in the time interval between two discrete time steps. Tang et al. [TCYM08] computed the surface normal cone in such time interval. We take a similar approach used for the surface normal cone, to extend the binormal cone for CCD. The detailed derivation is given in the supplementary report.

5. Mesh and BVH Update Method

In this section we propose our mesh and BVH update methods for efficient CD for fracturing models.

5.1. Mesh Update Interface

We define a *body* to be a set of triangles; also, a body can have only one triangle. We use the body as a unit data structure for mesh deformations. We assume that simulation applications update meshes based on the following API:

- **Add** $(\cdot)$: Add a new body into a scene.
- **Delete** $(\cdot)$: Delete an existing body.
- **Split** $(\cdot)$: Split an existing body into a set of bodies.
- **Merge** $(\cdot)$: Merge a set of existing bodies to a body.
- **Move** $(\cdot)$: Modify the position of each vertex of a body.

All the functions other than **Move**$(\cdot)$ modify the topology of a deforming model. After each API function is called, we first update the BVH to reflect the changes of the mesh (Sec. 5.3) and restructure only the sub-BVHs (Sec. 5.4) that have a low culling efficiency, which is defined in the following section.

5.2. Culling Efficiencies of BVHs

We define our traversal cost metrics to measure the expected numbers of BV overlap tests that are performed recursively to identify self-collisions and inter-collisions respectively during the BVH-based CD method. Intuitively speaking, if a sub-BVH has a higher traversal cost value, the sub-BVH has a lower culling efficiency. For the simple derivation, we measure the cost related only to BV overlap tests; our traversal cost metric can be easily extended to also consider the costs related to the primitives associated with leaf nodes.

We use a traversal cost, $TC_S(n)$, to measure the expected number of BV overlap tests that are recursively performed to identify self-collisions under a node n during the BVH-based CD. A self-collision detection query on a node n requires 1) an inter-collision detection query between two child nodes of the node n and 2) two self-collision detection queries on each of those two child nodes. Therefore, $TC_S(n)$ of the self-collision detection query to the node n is defined as the following:

$$TC_S(n) = TC_I(n_l, n_r) + I_{n_l} \times TC_S(n_l) + I_{n_r} \times TC_S(n_r), \quad (5)$$

where I_{n_l} is an indicator variable that takes a value of 0 or 1, when n_l is culled or not culled respectively for the query based on our dual-cone method; I_{n_r} is defined similarly. Also, $TC_I(n_l, n_r)$ is a traversal cost of an inter-collision query between two nodes n_l and n_r. This measures the number of BV overlap tests that are recursively performed to identify any inter-collisions between those two nodes during the BVH-based CD. Since the term of $TC_I(n_l, n_r)$ is defined for pairs of two arbitrary nodes, evaluating this term is inefficient during a simple BVH traversal, i.e. during a BVH update that is performed by traversing the BVH in a depth-first or breadth-first order. Therefore, we decided to approximate this term by using a traversal cost defined in each single node of a BVH.

For this purpose, we define a traversal cost, $TC_I(n)$, to measure the expected number of BV overlap tests that are recursively performed for an inter-collision query between the node n and an unknown node, m. Note that the node m can be any node in the merged BVH and thus its BV can be arbitrary. To perform the inter-collision query, we first perform a BV overlap test between them. If there is a BV overlap, then the two child nodes of the node n are tested against the node m, to further localize colliding primitives. Therefore, $TC_I(n)$ can be defined as the following:

$$TC_I(n) = 1 + P(n_l)TC_I(n_l) + P(n_r)TC_I(n_r), \quad (6)$$

where $P(n_l)$ represents a probability that the node n_l will be intersected with another node m given its parent node n is intersected; $P(n_r)$ is defined similarly. Fortunately, this probability was previously derived [YM06,Zac02] and following its formulation [YM06], we define $P(n_l)$ as $Vol(n_l)/Vol(n)$, where $Vol(n)$ is the volume of the BV node n; $P(n_r)$ is also defined similarly.

We then approximate $TC_I(n_l, n_r)$ as $TC_I(n_l) + TC_I(n_r)$, since the number, $TC_I(n_l, n_r)$, of BV overlap tests performed recursively by refining two nodes n_l and n_r is highly likely to have a linear correlation with the sum of $TC_I(n_l)$ and $TC_I(n_r)$, each of which measures BV overlap tests performed recursively by refining the nodes n_l and n_r separately. As a result, our final traversal cost $TC_S(n)$ of a self-collision query to the node n is formulated as the following:

$$TC_S(n) \approx TC_I(n_l) + TC_I(n_r) + I_{n_l} TC_S(n_l) + I_{n_r} TC_S(n_r). \quad (7)$$

5.2.1. Validations

We verify how much correlation our traversal cost metrics, $TC_S(n)$ (Eq. 7) and $TC_I(n)$ (Eq. 6), have with the observed numbers of overlap tests in BVHs computed from various models. We perform these validation tests with seven different models including the Stanford bunny model, a club, a gear, and our benchmark models. We use a simple median-based BVH construction method and compute 500 different versions of BVHs of each model by partitioning triangles contained in a node with a randomly-chosen axis-aligned partitioning plane. Then, we compute $TC_S(n_R)$ and $TC_I(n_R)$, where the node n_R is the root node of the BVH. Also, in order to measure the observed number of the BV overlap tests, we perform a self-collision query to n_R of each BVH of each model and perform 100 different inter-collision queries between the model and a second model chosen from the test set, after rotating and translating the second model randomly. We then compute linear correlations between our traversal cost metric values and the numbers of BV overlap tests obtained from different BVHs; we use the average number of overlap tests computed from those 100 different inter-collision queries to compute the linear correlation for the inter-collision query. For $TC_I(n_R)$ of the inter-collision detection, we found that linear correlations range from 0.6 to 0.77 with different models; the average linear correlation is 0.71. Also, for $TC_S(n_R)$ of the self-collision detection, the linear correlation ranges from 0.28 to 0.76; the average is 0.48. Our metric values and observed numbers of BV overlap tests in the Standford bunny and the cloth benchmark are shown in Fig. 10 and Fig. 11 of the supplementary report respectively.

5.3. BVH Update Method

When a simulation changes the topology of a deforming mesh, we first modify the tree structure of the BVH of the mesh locally. To perform these local operations, we need to know a BV node that contains a body. For this, we maintain a pointer in each body that indicates a BV node that contains the body.

Suppose that a simulation calls **Merge** $(\cdot)$. We compute the lowest common ancestor, n_a, from nodes that contain a given set of bodies. Then, we delete all the nodes from the BVH and add the merged body at the lowest common ancestor n_a. Then, starting from n_a, we recursively relocate the merged body to the left or right node of n_a that yields a lower traversal cost (Eq. 7), down to a leaf node in the BVH. Once it reaches a leaf node, n_{leaf}, we compute a sub-BVH for the merged body and let the sub-BVH to be one of the child nodes of the leaf node and use the unused child node to contain the geometry stored in the leaf node n_{leaf}. When a simulation calls **Split** $(\cdot)$ that splits a body, b, into a set of new bodies, we first identify the node that contains the body b. Then, we simply replace the node and its sub-BVH with a sub-BVH computed from the new bodies. When a simulation calls **Add** $(\cdot)$, we have to recursively traverse the BVH from its root node, add the given bodies to leaf nodes, and construct sub-BVHs for these bodies. Note that since we add these bodies without any hints on the relationship between the given bodies and existing bodies in the BVH, the cost required to perform this operation is typically higher than that of **Merge** $(\cdot)$ or **Split** $(\cdot)$. For bodies that are called with **Delete** $(\cdot)$, we remove nodes that contain the bodies.

Once we process all these mesh update functions, then we traverse the BVH in a bottom-up manner to refit BVs of the BVH to reflect geometric changes caused by **Move** $(\cdot)$. During this BV refitting process, we also update our dual-cones and traversal cost metrics associated with each node. This refitting process takes a small portion (e.g., 5% to 7%) of the total CD time in our benchmark models. Although these local modifications to tree structures, the BV-refitting method, and the dual-cone update run quite fast, the culling efficiency of the BVH can be quite low at complex fracturing cases, leading a lower overall runtime performance. Therefore, we perform a selective restructuring method as a final step of our BVH update method.

5.4. Selective Restructuring

We consider two factors to decide whether we reconstruct a sub-BVH or not: 1) a potential performance benefit caused by an improved culling efficiency by reconstructing the sub-BVH and 2) a potential performance loss due to the time spent on reconstructing the sub-BVH. During a simple top-down BVH traversal such as a depth-first or breadth-first traversal, we measure the potential performance benefit and loss, and restructure a sub-BVH only if both of the following two conditions are met: 1) the potential performance benefit is bigger than the performance loss and 2) the potential performance loss is not higher than p% of the overall CD time at the previous frame.

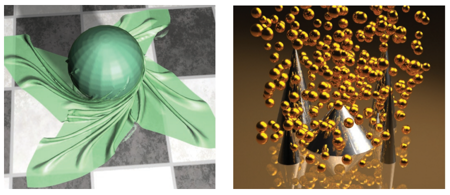

Figure 5: *These images show our cloth and N-body simulation benchmarks, which consist of 92 K and 34 K triangles respectively. These benchmarks have the same and fixed topology during the simulation.*

To quantify these potential improvement and loss, we take the following simple heuristics. To estimate the time that would be spent on reconstructing a sub-BVH (n), we record the number of triangles that are contained in the sub-BVH to the node n and plug it to the time complexity function of our reconstruction method. For this method, we also need to estimate the constant factor in the time complexity function. We can estimate it by running a simple micro-benchmark at the startup of our method in a similar manner proposed in a previous work [YCM07]. For the potential performance improvement, we need to predict how much culling efficiency of a sub-BVH we can improve by reconstructing the sub-BVH. We simply assume that by reconstructing the sub-BVH, its traversal cost metric value can be decreased to the traversal cost metric value of a sub-BVH reconstructed at the last time. By running a simple micro-benchmark, we can estimate a constant factor between the traversal cost metric value and the observed CD time and thus estimate how much running time we can save by reconstructing a sub-BVH.

Our approximations, especially, the approximation of potential performance improvements, are based on drastic simplifications. One problem that we have observed is that when we restructure a large sub-BVH by expecting a higher performance improvement, it may cause a performance degradation, because the performance improvement can be overestimated. In order to avoid this problem that can cause the performance drop, we have the second condition in our selective restructuring method. We use p to be 25 and found it works well in our tested benchmarks.

5.5. Fast BVH Construction

In addition to constructing a BVH for a deforming model at the initial frame, there are two other cases when we need to construct or reconstruct sub-BVHs from scratch: 1) for newly added bodies at any frame and 2) for the sub-mesh contained in the node that is chosen for selective restructuring at any frame. In order to further reduce the construction time for these two cases, we propose a fast BVH construction method.

Model	Num. Frames	Num. Tri. (K)	Rep. Image
N-body simulation	149 (0)	32 to 32	Fig. 5-(b)
Cloth simulation	465 (0)	92 to 92	Fig. 5-(a)
Breaking walls	80 (8)	42 to 140	Fig. 1-(b)
Exploding dragon	250 (4)	252 to 252	Fig. 1-(a)

Table 1: *Each benchmark model is shown with its representative image,* ***Rep. Image****, and the number of total frames,* ***Num. Frames****, with the number of frames that have topological changes. We also report the number of triangles of the model at the initial and the last frames under the column of* ***Num. Tri.***

Given a mesh consisting of N triangles, we compute a uniform grid that has $2^k \times 2^k \times 2^k$ regular cells and encloses the BV of the mesh. We also construct a balanced BVH on top of the grid such that each leaf node of the BVH corresponds to a cell of the grid. We let each cell to record a pointer to its corresponding leaf node in the BVH. Then, we can add each triangle of the mesh directly into a leaf node of the BVH in a constant time, by computing a grid cell that contains the centroid of the triangle and locating the leaf node associated with the cell. This is one iteration of our BVH construction method. Since each leaf node can have multiple triangles, we also apply this same process for each leaf node until each leaf node has less than G triangles. For leaf nodes that have less than G triangles, we simply apply the common median-based BVH construction method [TKH*05].

Our fast BVH construction method has an $O(N \log N)$ time complexity like many other BVH construction methods [TKH*05]. However, our BVH construction method has a lower constant factor and runs quite fast, since each iteration of our method traverses triangles only one time and uses a hashing method instead of using an expensive sorting method. On the other hand, BVHs constructed by our method can have a lower quality compared to BVHs constructed from other $O(N \log N)$ BVH construction methods, since we do not consider the geometric distribution of triangles at each iteration. Moreover, some of the leaf nodes may not have any associated triangles, wasting memory space. To minimize the negative sides of our method, we set k to be 1. Also, we found that setting G ranging from 100 to 1000 works well in our benchmarks.

6. Results and Comparisons

We have implemented DCD and CCD with our methods on an Intel i7 desktop that has one 3.2 GHz quad-core CPU and 2 GB main memory. We use axis-aligned bounding volumes and use feature-based BVHs [CTM08] that avoid any redundant elementary tests for CCD. Also, the feature-based BVHs do not require much pre-computations and suit well for handling fracturing models. We use a single CPU thread for all the tests.

Benchmark models: We have tested our method with four different benchmarks (Table 1) that have different characteristics. Our first and second benchmarks are cloth and N-body simulations (Fig. 5) that consist of 92 K and 34 K triangles respectively, and do not have any topological changes. Our third model is a breaking-wall model (Fig. 1), where a ball breaks 8 different walls. When the ball collides with a wall, the wall is first deformed to make a dent and then breaks into multiple pieces. This benchmark is obtained by simulating fracturing (and denting) brittle materials [BHTF07]. This model starts with 42 K triangles and ends with 140 K triangles, with topological changes due to the fractures at 8 different frames among the total 80 frames. Our fourth model is an exploding dragon benchmark, which has been tested frequently in prior collision detection literature. In this model, a bunny collides with a dragon model and then the dragon model breaks into numerous pieces (Fig. 1). Originally, this benchmark was created with a fixed topology by using pre-cut fracture boundaries, but is modified to create fracture boundaries during the thin-shell simulation and thus have dynamic topology. This model has 252 K triangles throughout the simulation, but has topological changes at four different frames among the total 250 frames.

6.1. Results

We perform DCD and CCD that include self-collision detection with our benchmarks. Our method spends 252 ms and 715 ms for DCD and CCD respectively with the dragon benchmark. Also, our method achieves 97 ms for DCD with the breaking-wall benchmark. Fig. 6 shows a frame rate graph of CCD on the exploding dragon model. Also, Fig. 7 shows a frame rate graph of DCD on the breaking-wall benchmark. For deforming models that do not have any topological changes, our method spends 345 ms and 17 ms for CCD on the cloth and N-body benchmarks respectively. We achieve a very high performance for the N-body benchmark, mainly because it has a smaller model complexity.

We also measure the times spent on different components of our method with the exploding dragon model:1) the mesh update, 2) the BVH update including BV-refitting, dual-cone refitting, and selective restructuring, and 3) the BVH traversal for CD; we report times spent on these components in the order each component appears. In the case of CCD, each component takes 8%, 24%, and 68% of the total CCD time; DCD also has similar percentages for components. Note that when deforming models change their topology, the BVH update takes 35% of the total CCD time, which is not drastically increased from 24% computed from all the frames.

6.2. Comparisons

We compare our method with a CCD technique, **T-CCD**, proposed by Tang et al. [TCYM08], which is one of the state-of-the-art CCD methods for deforming models that have fixed topologies. **T-CCD** performs continuous versions of normal cone and contour tests in addition to employing a selective restructuring method [LAM06], which uses the median-based partitioning and a heuristic metric, the **LM** metric, of identifying sub-BVHs that have low culling efficiencies. We also compare our method with an optimized spatial hashing CD method [THM*03] (**S-Hash**), which is widely used for the finite element method (FEM) based simulations that can support fracturing.

6.2.1. Tests with benchmarks with fixed topologies

Our dual-cone method is mainly designed for fracturing models. However, it can be also used with models that have fixed topologies. To see benefits of our method, we also compare our method over two modified versions, **T-CCD\CCT** and **T-CCD\CCT(Internal)**, of **T-CCD**. **T-CCD\CCT** does not perform the continuous contour test, as used in prior work [Pro97]. **T-CCD\CCT(Internal)** performs continuous contour test only for real boundary edges while ignoring internal boundary edges, as our BVH-based culling method integrated with dual-cones ignores self-collisions among internal boundary edges. These two

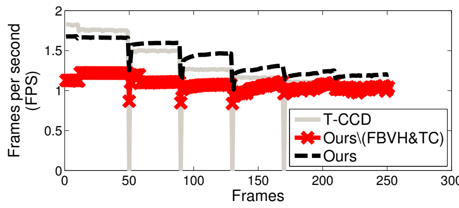

Figure 6: *This figure shows frame rate graphs of CCD for the dragon benchmark with Tang et al.'s method (**T-CCD**), our method (**Ours**), and **Ours\(FBVH&TC)** that does not use our fast BVH construction method and traversal cost metric for the selective restructuring.*

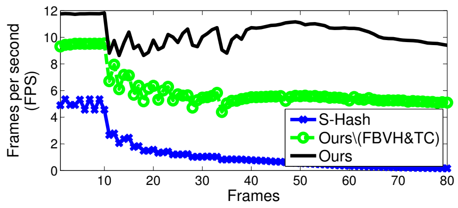

Figure 7: *This figure shows frame rate graphs of DCD for the breaking-wall benchmark with this spatial hashing method (**S-Hash**), our method (**Ours**), and **Ours\(FBVH&TC)** that does not use our BVH construction method and traversal cost metric.*

methods, **T-CCD\CCT** and **T-CCD\CCT(Internal)**, including our method are approximate. We obtained the original source codes of **T-CCD** from its authors and integrated our dual-cone culling method into **T-CCD\CCT**. We call this **T-CCD\CCT+Dual-Cone**. Since Tang et al.'s original implementation does not handle fracturing models, we use the cloth benchmark for the test.

We measure the time spent on performing CCD with different methods in the cloth benchmark (Fig. 8). By using **T-CCD\CCT+Dual-Cone**, we achieve 25% and 6% higher performance than **T-CCD** and **T-CCD\CCT(Internal)** respectively. We also observe only 5% lower performance than **T-CCD\CCT**, because of the low overhead of evaluating our dual-cone culling method. We found that **T-CCD**, **T-CCD\CCT+Dual-Cone**, and **T-CCD\CCT(Internal)** do not miss any collisions, while **T-CCD\CCT** misses collisions during the cloth simulation benchmark. To check whether an algorithm misses collisions or not, we compare the collision results of each method with those of the naive CD method that exhaustively checks collisions between all the pairs of two triangles from the model.

We define a culling ratio of a culling method to be the ratio of the numbers of BV overlaps with and without using each culling method, given the framework of **T-CCD**. We found that our dual-cone method, **T-CCD\CCT+Dual-Cone**, shows a comparable culling ratio (46.5%) to that (48.9%) of the culling methods used in **T-CCD**. Although our method has a bit lower culling ratio, we achieve a higher performance (e.g., 25%) because of its lower computational overhead, while detecting all the collisions.

6.2.2. Tests with benchmarks with varying topologies

To handle fracturing models with varying topologies, we implement our own CD framework (**Ours**), a different CD framework to that of **T-CCD**, since the CD framework of **T-CCD** is not appropriate for handling fracturing models. We compare the CCD performance of our method with that of **T-CCD** for the breaking dragon benchmark. Since **T-CCD** does not handle such fracturing models, we run the **T-CCD** from scratch, whenever the simulation produces a mesh that has different topology from that of the previous frame. Although these two methods are designed for different purposes, we compare them to shed light on their main characteristics with fracturing models that have topological changes.

As can be seen in Fig. 6, our method (**Ours**) shows a more stable performance even when deforming models change their topology. On the other hand, **T-CCD** shows drastic performance degradations at such cases, mainly because they have to reconstruct the BVH from scratch and re-compute various data in order to improve the performance of exact continuous contour tests; **T-CCD\CCT(Internal)** shows the performance similar to that of **T-CCD**. More specifically speaking, our method improves the CCD performance by a factor of 260 times and 200 times over **T-CCD** and **T-CCD\CCT(Internal)** respectively at frames where deforming models change their topologies. The more graceful performance degradation of our method is due to the fast construction of our dual-cones and our selective restructuring method that also uses our fast BVH construction method.

We also compare the DCD performance of our method with that of **S-Hash** [THM*03]. Our method (**Ours**) runs 20 times faster in the breaking-wall models (Fig. 7). The inferior performance of **S-Hash** is mainly because many parts of fracturing models come in a close proximity, causing many grid cells to have multiple triangles. We also achieve 7 times and 56% improvements in the N-body and cloth benchmarks respectively over **S-Hash**.

We also compare our method with its two variations: 1) **Ours\FBVH** that does not use our fast BVH construction method, but uses the median-based partitioning within our CD framework and 2) **Ours\(FBVH&TC)** that does not use our traversal cost metric additionally from **Ours\FBVH**. Instead, it uses the **LM** metric. We found that **Ours** achieves 76% and 27% performance improvements on average in the exploding-wall model (Fig. 7) and in the dragon model (Fig. 6) respectively over **Ours\(FBVH&TC)**, because of both of our traversal cost metric and the fast BVH construction method. Also, we found that because of our fast BVH construction method, **Ours** achieves 84% improvement over **Ours\FBVH** in the dragon model at frames when the deforming models change their topologies. Frame rate graphs of **Ours** and **Ours\FBVH** are shown in Fig. 12 of the supplementary report.

Overhead of supporting fracturing models: Recently, Kim et al. [KHH*09] showed interactive performances for various deformable models that do not have any topological changes by using multiple CPUs and GPUs. This method uses the same type of BVs and BVHs with ours, while it uses the **LM** metric and the median-based partitioning in its selective restructuring method. We measure the overhead of our CD framework that supports fracture models, by comparing the performances of ours and theirs. According to their paper, their method spends 675 ms for the exploding dragon simulation thats uses pre-cut boundaries and thus has a fixed topology throughout the simulation. This result is achieved by using a single CPU thread in a machine that has the same CPU performance to that of our testing machine. In the same configuration, our method with the **LM** method and the median-based partitioning spends 902 ms for

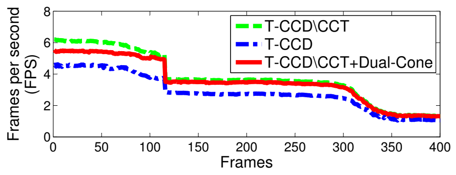

Figure 8: *This figure shows frame rate graphs of CCD with various methods in the cloth simulation benchmark.* ***T-CCD*** *refers to the original Tang el al's CCD framework.* ***T-CCD\CCT*** *does not any contour tests in* ***T-CCD***. ***T-CCD\CCT+Dual-Cone*** *indicates our dual-cone method integrated in* ***T-CCD\CCT***.

the same exploding dragon model that is modified to create fracturing boundaries and have topological changes. Therefore, we can conjecture that our CD framework supporting fracturing models has approximately 30% more overhead compared to the CD framework supporting only deforming models that do not have any topological changes. However, by using our selective restructuring method and the fast BVH construction method, our method becomes to spend 709 ms, which is a comparable performance to the result, 675 ms, achieved with the CD framework that does not support fracturing models.

6.3. Discussions

We discuss various aspects of our method and point out limitations of our method.

Local vs. global BVH restructuring: Otaduy et al. [OCSG07] proposed a BVH restructuring algorithm that maintains balanced BVHs for fracturing models by using AVL trees and applying simple local operations similar to techniques mentioned in Sec. 5.3. However, this method is suited mainly for fractures that occur progressively. For instantaneous fractures with many fracturing pieces like our two fracturing benchmarks, they suggest a full BVH reconstruction instead. Our method shows much higher performance than the full reconstruction, which has been employed in **T-CCD** for fracturing events. In general, the local BVH restructuring method may not show a high culling efficiency for complex and large-scale fracturing models similar to our benchmark models. Instead, our selective restructuring method with culling efficiency metrics can identify sub-BVHs with low-culling efficiency and restructure them in a more global manner, leading to more high-quality BVHs.

Low-level culling techniques: Recently, Tang et al. [TMT10] proposed the deforming non-penetration filters (DNFs) to improve the performance of low-level elementary tests for CCD, in the framework of **T-CCD**. We can take advantage of the performance benefit of the DNF, since it improves the performance of low-level elementary tests used for CCD, while our method belongs to a class of high-level culling methods and improves overall performance for DCD and CCD. To verify this, we have integrated the DNFs within our method and variations of **T-CCD** tested in the paper. We observe that the overall performance of our method is improved and our method still achieves similar improvements over those variations of **T-CCD** integrated with the DNFs, compared to improvements over the variations of **T-CCD** without using the DNFs.

GPU-based culling methods: Sud et al. [SGG*06] proposed a unified GPU-framework for various proximity queries including DCD and CCD for deforming models. In their paper, they tested only with fracturing models that have pre-cut boundaries and the fixed topologies. However, we conjecture that their method can show a similar performance even for fracturing models with dynamic topologies, since their method does not require pre-computations. We contacted Sud et al. to get their binary, but were unable to get their binary. Therefore, we provide the indirect comparison in below. Recently, a hybrid parallel CD method that uses CPUs and GPU [KHH*09] outperforms Sud et al.'s method by about 5 to 10 times. Note that our method can be parallelized in the same manner of the hybrid parallel CD method, since both of these two methods use the same BVH-based CD method. Therefore, we expect that our method would produce a higher performance than Sud et al.'s method, if our method is parallelized with CPUs and GPUs.

Supporting volumetric representations: Our CD method is mainly designed for surface-based deformable simulations. However, it can be also applied to volume representations (e.g., tetrahedral meshes) that are widely used in FEM based simulations. To do that, one has to extract boundary surfaces from the volumetric representation and perform our method with the boundary surfaces. Since our method is much faster than the spatial hashing method for complex and large-scale fracturing models as demonstrated in Sec.6.2, this approach may be a viable option for such models.

Limitations: Our method has certain limitations. As mentioned earlier, our dual-cone method combined with BVHs is approximate and thus may miss collisions, although our method did not miss any collisions in our tested benchmarks. Also, the memory requirement of our method is relatively high. For example, our current implementation uses 92 bytes for each BV node and requires 50 MB for the BVH of the dragon model, the largest model in our tested benchmakrs. Also, our culling efficiency metrics for self-collision and inter-collision queries are probabilistically derived. Therefore, our selective restructuring method with these metrics does not guarantee to always improve the performance of CD.

7. Conclusion and Future Work

We have presented dual-cone culling and selective restructuring methods to improve the performance of CD including self-collision detection among fracturing models. The interplay between the dual-cone method and our selective BVH update method achieved a faster and more stable collision detection performance even at fracturing events of large-scale fracturing models. We believe that achieving fast and stable performance of collision detection for fracturing models can trigger various new research efforts to create more complex and realistic fracturing simulations.

Currently, we used only a single CPU thread and our method did not show interactive performance (e.g., more than 10 frames per second) in two of our testing benchmark models. However, by adopting recent parallel CD methods [KHH*09, TMT09] designed for the BVH-based CD methods, we believe that our method can be interactive even for all the large-scale fracturing models tested in this paper.

There are many avenues for future work, in addition to addressing the limitations of our method. We found that most primitive tests (e.g., 90% of total primitive tests in the exploding dragon benchmark) turned out to be false positives. Therefore, we would like to design a more effective culling method for self-collision queries. Although we achieved a more stable performance with topological changes, our method still shows lower performances at such cases. To fundamentally address this problem, we would like to design an error-bounded multi-resolution CD method for high-quality interactive fracturing simulations. Finally, in concurrent work, Schvartzman et al. [SPO10] proposed an exact, yet effective contour test method for identifying self-collision free regions. We would like to extend this work for fracturing models and CCD.

Acknowledgements

We would like to thank anonymous reviewers and the members of SGLab. for their helpful feedbacks. This work was supported in part by NSFC (No. 60803054), Spanish Dept. of Science and Innovation (projects TIN-2009-07942 and PSE-300000-2009-5), BK, KAIST, KOSEF/MEST (2009-0077290), MKE/IITA u-Learning, KRF-2008-313-D00922, MKE/MCST/IITA[2008-F-033-02], MKE digital mask control, MCST/KOCCA-CTR&DP-2009, MSRA, KMCC, DAPA/ADD (UD080042AD), and the MKE/KEIT/MCST project of semi-realtime renderer.

References

[BHTF07] Bao Z., Hong J.-M., Teran J., Fedkiw R.: Fracturing rigid materials. *IEEE Trans. on Visualization and Computer Graphics 13*, 2 (2007), 370–378. 1, 7

[CTM08] Curtis S., Tamstorf R., Manocha D.: Fast Collision Detection for Deformable Models using Representative-Triangles. *Symp. on Interactive 3D Graphics* (2008), 61–69. 1, 2, 7

[dC76] do Carmo M.: *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, 1976. 4

[Eri04] Ericson C.: *Real-Time Collision Detection*. Morgan Kaufmann, 2004. 2

[GKJ*05] Govindaraju N., Knott D., Jain N., Kabal I., Tamstorf R., Gayle R., Lin M., Manocha D.: Collision detection between deformable models using chromatic decomposition. *ACM Trans. on Graphics 24*, 3 (2005), 991–999. 1, 2

[Hub93] Hubbard P. M.: Interactive collision detection. *Proc. of IEEE Symposium on Research Frontiers in Virtual Reality* (1993), 24–31. 2

[KHH*09] Kim D., Heo J.-P., Huh J., Kim J., Yoon S.-E.: HPCCD: Hybrid parallel continuous collision detection. *Computer Graphics Forum (Pacific Graphics) 28*, 7 (2009). 1, 8, 9

[LAM06] Larsson T., Akenine-Möller T.: A dynamic bounding volume hierarchy for generalized collision detection. *Computers and Graphics 30*, 3 (2006), 451–460. 1, 2, 7

[LM03] Lin M., Manocha D.: Collision and proximity queries. *Handbook of Discrete and Computational Geometry* (2003). 1, 2

[OCSG07] Otaduy M., Chassot O., Steinemann D., Gross M.: Balanced hierarchies for collision detection between fracturing objects. *IEEE Virtual Reality Conf.* (2007), 83–90. 1, 2, 9

[OD01] O'Sullivan C., Dingliana J.: Collisions and perception. *ACM Trans. on Graphics 20*, 3 (2001), pp. 151–168. 2

[OH99] O'Brien J. F., Hodgins J. K.: Graphical modeling and animation of brittle fracture. In *ACM SIGGRAPH* (1999), pp. 137–146. 1

[OL03] Otaduy M. A., Lin M. C.: Sensation preserving simplification for haptic rendering. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)* (2003), 543–553. 2

[PO09] Parker E. G., O'Brien J. F.: Real-time deformation and fracture in a game environment. In *Symp. on Computer Animation* (2009), pp. 165–175. 1

[Pro97] Provot X.: Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface* (1997), 177–189. 2, 3, 4, 7

[RKC02] Redon S., Kheddar A., Coquillart S.: Fast continuous collision detection between rigid bodies. *Computer Graphics Forum 21*, 3 (2002), 279–287. 1

[SGG*06] Sud A., Govindaraju N., Gayle R., Kabul I., Manocha D.: Fast Proximity Computation among Deformable Models using Discrete Voronoi Diagrams. *ACM SIGGRAPH* (2006), 1144–1153. 9

[SOG06] Steinemann D., Otaduy M. A., Gross M.: Fast arbitrary splitting of deforming objects. In *ACM Symp. on Computer Animation* (2006), pp. 63–72. 1

[SPO10] Schvartzman S. C., Pérez A. G., Otaduy M. A.: Star-contours for efficient hierarchical self-collision detection. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH) 29*, 3 (2010). To appear. 10

[TCYM08] Tang M., Curtis S., Yoon S.-E., Manocha D.: Interactive continuous collision detection between deformable models using connectivity-based culling. *ACM Symp. on Solid and Physical Modeling* (2008), 25 – 36. 1, 2, 3, 5, 7

[THM*03] Teschner M., Heidelberger B., Muller M., Pomeranets D., Gross M.: Optimized spatial hashing for collision detection of deformable objects. In *Proc. of Vision, Modeling and Visualization* (2003), pp. 47–54. 7, 8

[TKH*05] Teschner M., Kimmerle S., Heidelberger B., Zachmann G., Raghupathi L., Fuhrmann A., Cani M.-P., Faure F., Magnenat-Thalmann N., Strasser W., Volino P.: Collision detection for deformable objects. *Computer Graphics Forum 19*, 1 (2005), 61–81. 2, 3, 7

[TMT09] Tang M., Manocha D., Tong R.: Multi-core collision detection between deformable models. In *SIAM/ACM Joint Conf. on Geometric and Solid & Physical Modeling* (2009), pp. 355–360. 1, 9

[TMT10] Tang M., Manocha D., Tong R.: Fast continuous collision detection using deforming non-penetration filters. In *Symp. on Interactive 3D Graphics* (2010), pp. 7–13. 9

[VT94] Volino P., Thalmann N. M.: Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum (EuroGraphics Proc.) 13*, 3 (1994), 155–166. 1, 2, 3

[WH06] Wald I., Havran V.: On building fast kd-trees for ray tracing, and on doing that in O(N log N). In *IEEE Symp. on Interactive Ray Tracing* (2006), pp. 61–69. 2

[WTGT09] Wojtan C., Thürey N., Gross M., Turk G.: Deforming meshes that split and merge. *ACM Trans. Graph. 28*, 3 (2009), 1–10. 1

[YCM07] Yoon S., Curtis S., Manocha D.: Ray tracing dynamic scenes using selective restructuring. *Eurographics Symp. on Rendering* (2007), 73–84. 6

[YM06] Yoon S.-E., Manocha D.: Cache-efficient layouts of bounding volume hierarchies. *Computer Graphics Forum (Eurographics) 25*, 3 (2006), 507–516. 5

[YSLM04] Yoon S., Salomon B., Lin M. C., Manocha D.: Fast collision detection between massive models using dynamic simplification. In *Eurographics Symposium on Geometry Processing* (2004), pp. 136–146. 2

[Zac02] Zachmann G.: Minimal hierarchical collision detection. In *ACM symp. on Virtual reality software and technology* (2002), pp. 121–128. 5

[ZW06] Zachmann G., Weller R.: Kinetic bounding volume hierarchies for deforming objects. In *Virtual Reality Continuum and its Applications* (2006), pp. 189–196. 2