The Visual Computer manuscript No.
(will be inserted by the editor)

Xinyu Zhang · **Minkyoung Lee** · **Young J. Kim**+

Interactive Continuous Collision Detection for Non-Convex Polyhedra

http://graphics.ewha.ac.kr/FAST

Abstract We present a highly interactive, continuous collision detection algorithm for rigid, general polyhedra. Given initial and final configurations of a moving polyhedral model, our algorithm creates a continuous motion with constant translational and angular velocities, thereby interpolating the initial and final configurations of the model. Then, our algorithm reports whether the model under the interpolated motion collides with other rigid polyhedral models in environments, and if it does, the algorithm reports its first time of contact (TOC) with the environment as well as its associated contact features at TOC.

Our algorithm is a generalization of conservative advancement [20] to general polyhedra. In this approach, we calculate the motion bound of a moving polyhedral model and estimate the TOC based on this bound, and advance the model by the current TOC estimate. We iterate this process until the inter-distance between the moving model and the other objects in the environments becomes below a user-defined distance threshold.

We pose the problem of calculating the motion bound as a linear programming problem and provide an efficient, novel solution based on the simplex method. Moreover, we also provide a hierarchical advancement technique based on bounding volume traversal tree to generalize the conservative advancement for non-convex models.

Our algorithm is relatively simple to implement and has very small computational overhead of merely performing discrete collision detection multiple times. We extensively benchmarked our algorithm in different scenarios, and in comparison to other known continuous collision detection algorithm, the performance improvement ranges by a factor of $1.4 \sim 45.5$ depending on benchmarking scenarios. Moreover, our algorithm can perform CCD at $120 \sim 15460$ frames per second on a 3.6 GHz Pentium 4 PC for complex models consisting of $10K \sim 70K$ triangles.

Department of Computer Science and Engineering
Ewha Womans University, Seoul, Korea
E-mail:{*zhangxy*|*minkyounglee*|*kimy*}*@ewha.ac.kr*
Tel.: +82-2-3277-4068 Fax: +82-2-3277-2306
+: Corresponding Author

Keywords Continuous Collision Detection, Convex Decomposition, Conservative Advancement, Dynamics Simulation

1 Introduction

Collision detection (CD) is a problem of testing possible interference between geometric models in space. Many graphics applications require fast and reliable CD to simulate the physical presence of objects in virtual space. These cover a wide range of applications such as physically based animation, virtual environments, virtual characters, geometric modelling, etc. As a result, CD has been extensively studied in the literature and many efficient CD algorithms are known.

At a broad level, depending on how CD algorithms handle the motion of objects, they can be categorized into two types, *discrete CD and continuous CD*. Discrete CD algorithms check for interferences between static objects. When objects are moving, discrete CD algorithms are often called at fixed time intervals to deal with the motion of objects. However, it is well known that when objects are moving at a high speed or objects are thin, discrete CD algorithm can easily miss a collision between sampled time instances [24]. Meanwhile, continuous CD (CCD) algorithms take the object's continuous motion into account and accurately report the first time of contact (TOC) between moving objects if there occurs a collision. Recently, CCD has drawn much attention from different communities because of the need for correctly dealing with dynamic nature in applications. The major strength of using CCD in dynamic applications lies in a fact that the result of CCD always guarantees non-penetration between moving objects. For example, in rigid body dynamics, finding an accurate TOC between simulated bodies is crucial to enforce non-penetration constraints in the simulation [3, 25], and CCD is directly applicable to finding a TOC. In 6DOF haptic rendering, a recent technique based on CCD enables God-object type haptic rendering for object/object interactions thanks to the non-penetration condition imposed by CCD [22]. In sampling-based robot mo-

tion planning such as probabilistic roadmap method (PRM), it is crucial to find a continuous, collision free path between two configurations of a moving robot, and CCD plays an important role in finding one [28]. However, the major limitation in existing CCD algorithms is that they are, in general, much slower than discrete CD algorithms. This often limits the applicability of CCD algorithms to many potential applications.

1.1 Main Results

In this paper, we present a fast method to perform CCD between moving, general polyhedra. Our algorithm outperforms known CCD methods by utilizing the connectivity information residing in polyhedra and exploiting the coherence in the continuous motion. Given initial and final configurations of a moving polyhedral model, our algorithm creates a continuous motion with constant translational and angular velocities, there interpolating the initial and final configurations of the model. Then, our algorithm reports whether the model under the interpolated motion collides with other rigid polyhedral models in environments, and if it does, the algorithm reports its first time of contact (TOC) with the environment as well as its associated contact features at TOC.

The main ingredient of our algorithm is a generalization of conservative advancement [20] to general polyhedra. In this approach, we calculate the motion bound of a moving polyhedral model and estimate the possible TOC based on this bound and the inter-distance between the model and other objects in the environment, and advance the model by the current TOC estimate. We iterate this process until the inter-distance becomes less than a user-defined distance threshold. We pose the problem of calculating the motion bound as a linear programming problem and provide a novel, efficient solution based on the simplex method. Moreover, we also provide a novel, hierarchical advancement technique based on bounding volume traversal tree to generalize conservative advancement for non-convex models.

Our algorithm is relatively simple to implement in the sense that one needs to make only minor modifications to existing discrete CD library to implement our algorithm. As a result, our algorithm has very small computational overhead of iterating discrete CD multiple times. We extensively benchmarked our algorithm in different scenarios, and in comparison to other known continuous collision detection algorithm, the performance improvement ranges by a factor of $1.4 \sim 45.5$ depending on benchmarking scenarios. Moreover, our algorithm can perform CCD at $120 \sim 15460$ frames per second on a 3.6 GHz Pentium 4 PC for complex models consisting of $10K \sim 70K$ triangles.

1.2 Organization

The rest of this paper is organized as follows. In Sec. 2, we briefly survey work relevant to ours. In Sec. 3, we explain the preliminary concepts to understand our algorithm and provide an overview of our algorithm. In Sec. 4, we explain how we efficiently calculate the bound of motion and generalize conservative advancement to general polyhedra in Sec. 5. In Sec. 6, we show the performance results of our algorithm in different benchmarking scenarios and conclude the paper in Sec. 7.

2 Previous Work

In this section, we give a brief survey of the earlier work related to continuous collision detection and distance calculation.

2.1 Discrete Collision Detection and Distance Calculation

Most of the prior work on collision detection has focused on discrete CD algorithms. We refer the readers to see [18] for an excellent survey on the field. These include specialized algorithms for convex polytopes that exploit motion coherence between successive time steps, general algorithms for polygonal or spline models that pre-compute a spatial partitioning or bounding volume hierarchies (BVH).

Calculating Euclidean distance between polygonal models has been extensively studied in the literature. For convex polytopes, an efficient theoretical algorithm [8], linear programming-based approach [29], Minkowski-sum and convex optimization approach [11], *Voronoi marching* exploiting motion coherence [19] and multi-resolution approach [9, 13] have been proposed. For non-convex models, typically BVH has been employed to deal with the non-convexity in the models. A convex hull tree-based scheme combined with Voronoi marching has been introduced and shown to work at highly interactive rates for complex polyhedral models [10]. A hybrid BVH scheme using different types of swept sphere volumes has been presented to calculate distance between generic polygonal models [17].

2.2 Continuous Collision Detection

A few algorithms have been proposed for CCD. More specifically, there are five different approaches presented in the literature: algebraic equation solving approach [6, 7, 14, 23], swept volume approach [1], adaptive bisection approach [24, 28], kinetic data structures (KDS) approach [2, 15, 16], and Minkowski sum-based approach [5]. However, most of these approaches do not work in real-time or work for only simple or convex models. Only the adaptive bisection approach has shown interactive performance for relatively complex polygonal models; however, it is designed to handle models in *polygon soups* and does not take advantage of connectivity information residing in polyhedral models and motion coherence in CCD computations.

Recently, CCD for articulated models have been proposed [26, 27]. [26] is applicable to simple, capsule-shaped

avatar models in VR environments and shows interactive performance, whereas [27] works for articulated models but shows only near real-time performance.

3 Overview

We start this section by introducing the notations and definitions used throughout this paper. Then, we will briefly explain the basic idea of conservative advancement (CA) and distance calculation methods that our CCD algorithm is based on. Based on these concepts, we provide an overview of our algorithm later.

3.1 Notations and Definitions

We use a bold symbol to differentiate a vector quantity from a scalar value (e.g., the origin, **o**). Let $\mathscr{A}$ and $\mathscr{B}$ be two polyhedron. Without loss of generality, we assume that $\mathscr{A}$ is a moving object under rigid transformation $\mathbf{M}$ and $\mathscr{B}$ is fixed in space since we can find relative rigid transformation with respect to each other. Further, we assume that the initial and final configurations of $\mathscr{A}$ are given as $\mathbf{q}_0$ and $\mathbf{q}_1$ at its corresponding time, $t = 0$ and $t = 1$, respectively. We use $\mathscr{A}(t)$ as a placement of $\mathscr{A}$ under rigid transformation $\mathbf{M}$ at time t; i.e., $\mathscr{A}(t) = \mathbf{M}(t)\mathscr{A}$. Then, the problem of CCD can be formulated as checking whether Eq. 1 is non-empty:

$$\{\, t \in [0,1] \mid \mathscr{A}(t) \cap \mathscr{B} \neq \emptyset \}. \tag{1}$$

Furthermore, if Eq. 1 is non-empty, we want to compute the minimum value of t that satisfies this equation, known as the time of contact, t_{TOC}.

3.2 Conservative Advancement for Convex Polytopes

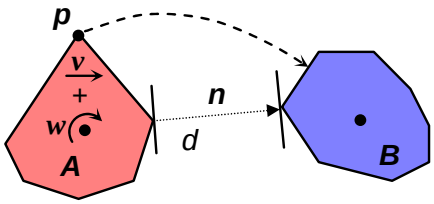

Fig. 1 Conservative Advancement.

The CA is a simple technique that computes an upper bound of t_{TOC} by repeatedly advancing $\mathscr{A}$ by Δt toward $\mathscr{B}$ while avoiding collision [20,21]. Here, Δt is calculated based on the closest distance $d(\mathscr{A}(t), \mathscr{B})$ between $\mathscr{A}(t)$ and $\mathscr{B}$, and the upper bound μ of the motion along $d(\mathscr{A}(t), \mathscr{B})$ traced by $\mathscr{A}(t)$ per unit second. More specifically, in Fig. 1, let $\mathbf{p}$ be a point on $\mathscr{A}$, and let $\mathbf{n}$ be the closest direction vector that realizes $d(\mathscr{A}(t), \mathscr{B})$. Moreover, let μ be an upper bound of the distance along $d(\mathscr{A}(t), \mathscr{B})$ travelled by any $\mathbf{p}$ on $\mathscr{A}$ per second. Then, $\mathscr{A}$ can advance from time t to time $t + \Delta t$ without collision:

$$\begin{aligned} \mu \Delta t &\leq d(\mathscr{A}(t), \mathscr{B}) \\ \Delta t &\leq \frac{d(A(t), B)}{\mu} \end{aligned} \tag{2}$$

The proof is simple, and we refer the readers to see [21] for detail.

In Eq.2, the smaller value of μ or the larger value of $d(\mathscr{A}(t), \mathscr{B})$, the larger CA step size Δt we can have. Thus, we need a tight upper bound of motion μ and tight lower bound of distance $d(\mathscr{A}(t), \mathscr{B})$.

Notice, however, that the technique presented in [21] works only for convex objects. Later in Sec. 5, we will provide novel techniques to generalize CA to non-convex objects and also provide a tighter upper bound of μ by considering both the linear and rotational motions in Sec. 4.

3.3 Distance Calculation using Convex Decomposition

It is easier to calculate the distance between convex polytopes than computing it for general, non-convex polyhedra. In particular, Voronoi Marching [9,19] exploits motion coherence involved in distance calculation and is a good choice for CA as CA requires repeated calculations of distance and thus the motion coherence is high between successive computations.

For distance calculation between general polyhedra, a BVH-based technique using *convex hull tree* is known to work well [10]. As preprocess, the algorithm decomposes a given, general polyhedron $\mathscr{A}$ into convex pieces $c_i^{\mathscr{A}}$ and recursively builds a convex hull tree where each node in the tree corresponds to a convex hull of its children nodes. Note that the convex decomposition scheme employed in [10] is merely surface decomposition in which the union of $c_i^{\mathscr{A}}$ covers only the boundary of $\mathscr{A}$, and this is sufficient for distance calculation.

At run-time, starting from the root nodes of two convex hull trees, the algorithm simultaneously traverses the two trees while performing Voronoi marching on the tree nodes and calculating their distance as well as their associated closest features. The recursive traversal continues until the closest distance cannot be further reduced. This tree traversal process can be also thought of as implicitly building a bounding volume traversal tree (BVTT) between two convex hull trees. The concept of BVTT will be explained in more detail in Sec. 5.1.

3.4 Our Approach

In our algorithm, we assume that two non-convex polyhedra $\mathscr{A}$ and $\mathscr{B}$ are given and only the initial and final configurations of $\mathscr{A}$ are known as $\mathbf{q}_0, \mathbf{q}_1$. As in many applications, since the actual governing motion of $\mathscr{A}$ is unknown [24], we use an arbitrary in-between motion that interpolates $\mathbf{q}_0, \mathbf{q}_1$.

As a result, we can express a continuous motion $\mathbf{M}(t)$ in terms of time that governs the motion of $\mathscr{A}$.

In order to find t_{TOC} between non-convex polyhedra $\mathscr{A}$ and $\mathscr{B}$, a naive approach based on convex decomposition and CA can be devised as in Alg. 1.

Algorithm 1 Naive CCD using Convex Decomposition
Input: A moving polyhedron $\mathscr{A}$ and its initial and final configurations $\mathbf{q}_0, \mathbf{q}_1$, a fixed polyhedron $\mathscr{B}$.
Output: Calculate t_{TOC}.

1: As preprocess, decompose $\mathscr{A}$ and $\mathscr{B}$ into convex pieces, $c_i^{\mathscr{A}}, c_j^{\mathscr{B}}$, as in [10].
2: Compute an interpolating motion $\mathbf{M}(t)$ from $\mathbf{q}_0, \mathbf{q}_1$.
3: **repeat**
4: **for** each pair of convex pieces $(c_i^{\mathscr{A}}, c_j^{\mathscr{B}})$ **do**
5: Calculate their distance $d(c_i^{\mathscr{A}}, c_j^{\mathscr{B}})$ using [9] and find their closest direction vector $\mathbf{n}$.
6: Calculate the motion bound μ of $c_i^{\mathscr{A}}$ using $\mathbf{n}$.
7: Calculate $\Delta t_{i,j} = \frac{d(c_i^{\mathscr{A}}, c_j^{\mathscr{B}})}{\mu}$ based on Eq. 2.
8: **end for**
9: Find $\Delta t := \min(\Delta t_{i,j})$.
10: Advance $\mathscr{A}(t)$ by Δt.
11: **until** $d(\mathscr{A}(t), \mathscr{B})$ becomes less than a user-provided threshold
12: **return** t_{TOC}

There are two severe problems in Alg. 1, however. First of all, if the numbers of convex pieces in $\mathscr{A}, \mathscr{B}$ are $\eta_{\mathscr{A}}, \eta_{\mathscr{B}}$, each CA step (step 4) requires $O(\eta_{\mathscr{A}}\eta_{\mathscr{B}})$ time. Additionally, the motion bound μ (step 6) can become overly conservative and many CA steps (step 11) would be required. These two problems can result in poor performance of the naive algorithm. To address the first issue, instead of calculating the distance between decomposed convex pieces of $\mathscr{A}, \mathscr{B}$, we calculate the distance between $\mathscr{A}, \mathscr{B}$ themselves using the technique presented in Sec. 3.3 and remember the nodes in convex hull trees that realize this distance, called *front* nodes. The union of these nodes is a super set of the union of convex pieces thanks to the convex hull property of convex hull tree. Notice that the distance between the front nodes has been already calculated during the BVH traversal scheme in Sec. 3.3. We perform CA only to these front nodes (Sec. 5). Moreover, to efficiently calculate μ between these front nodes, we take the closest vector direction $\mathbf{n}$ between the nodes and project its translational and rotational motion onto $\mathbf{n}$ (Sec. 4). Overall, our algorithm works as in Alg. 2.

4 Motion Bound Calculation

Since our algorithm does not assume a closed form of the continuous motion of a moving object, we first explain how we model the continuous motion when only its initial and final configurations are given, and also show how we can efficiently calculate the motion bound μ of a moving object under such a motion.

Algorithm 2 CCD using Hierarchical Advancement
Input: $\mathscr{A}, \mathscr{B}$ and $\mathbf{q}_0, \mathbf{q}_1$
Output: Calculate t_{TOC}.

1: As preprocess, decompose $\mathscr{A}$ and $\mathscr{B}$ into convex pieces and construct their convex hull trees, $CHT_{\mathscr{A}}, CHT_{\mathscr{B}}$.
2: Compute an interpolating motion $\mathbf{M}(t)$ from $\mathbf{q}_0, \mathbf{q}_1$.
3: **repeat**
4: Calculate $d(\mathscr{A}(t), \mathscr{B})$ between $\mathscr{A}(t)$ and $\mathscr{B}$ using $CHT_{\mathscr{A}}, CHT_{\mathscr{B}}$ and remember the pairs of front nodes $h_i^{\mathscr{A}}, h_j^{\mathscr{B}}$ in the trees that were traversed to calculate $d(\mathscr{A}(t), \mathscr{B})$. During the traversal, $d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})$ and closest direction vector $\mathbf{n}(i,j)$ are stored {Alg. 3 in Sec. 5}.
5: **for** each pair of $(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})$ **do**
6: Retrieve already calculated $d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})$ and $\mathbf{n}(i,j)$.
7: Calculate the motion bound $\mu(i,j)$ by projecting both the translational and rotation motion of $h_i^{\mathscr{A}}$ onto $\mathbf{n}(i,j)$.
8: Calculate $\Delta t_{i,j} = \frac{d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})}{\mu(i,j)}$.
9: **end for**
10: Find $\Delta t := \min(\Delta t_{i,j})$.
11: Advance $\mathscr{A}(t)$ by Δt.
12: **until** $d(\mathscr{A}(t), \mathscr{B})$ becomes less than a user-provided threshold
13: **return** t_{TOC}

4.1 Motion Interpolation

In many applications involving motion such as physically based animation, rapid prototyping, virtual environments and robot motion planning, its governing motion is often either unknown or cannot be represented as a closed form. This is because the motion employed in these application is controlled interactively by a user [24], or purely random [28], or should be calculated numerically [4]. Therefore, in our CCD algorithm, we only assume that the initial and final configurations of $\mathscr{A}$ are known ($\mathbf{q}_0, \mathbf{q}_1$), and need to find a continuous, rigid motion $\mathbf{M}$ that interpolates $\mathbf{q}_0, \mathbf{q}_1$. We further assume that $\mathbf{q}_0$ is collision-free; otherwise, we trivially report that t_{TOC} is zero by performing discrete CD at $\mathbf{q}_0$.

Different types of interpolation motions have been used in other CCD algorithms: screwing motion [14,24], ballistic motion [21] and linear motion in configuration space [28, 5]. In our case, we choose the linear motion with constant translational and angular velocities because of its simplicity. In fact, some application such as robot motion planning requires linear motion in configuration space [28].

Let us represent the configurations $\mathbf{q}_0, \mathbf{q}_1$ as their translational ($\mathbf{T}$) and rotational components ($\mathbf{R}$): $\mathbf{q}_0 = (\mathbf{R}_0, \mathbf{T}_0), \mathbf{q}_1 = (\mathbf{R}_1, \mathbf{T}_1)$. We want to represent the interpolating motion $\mathbf{M}(t)$ at time t as:

$$\mathbf{M}(t) = \begin{pmatrix} \mathbf{R}(t) & \mathbf{T}(t) \\ (0,0,0) & 1 \end{pmatrix} \tag{3}$$

Then,

$$\mathbf{T}(t) = \mathbf{T}_0 + t\mathbf{v}$$
$$\mathbf{R}(t) = \cos(\omega t).\mathbf{A} + \sin(\omega t).\mathbf{B} + \mathbf{C}$$

Here, $\mathbf{v} = \mathbf{T}_1 - \mathbf{T}_0$ is the constant translational velocity of the center of mass (COM) of $\mathscr{A}$. Furthermore,

$$\mathbf{A} = \mathbf{R}_0 - \mathbf{u}\mathbf{u}^T.\mathbf{R}_0$$
$$\mathbf{B} = \mathbf{u}^*.\mathbf{R}_0$$
$$\mathbf{C} = \mathbf{u}\mathbf{u}^T.\mathbf{R}_0$$

Here, $(\mathbf{u}, \omega)$ is the constant angular velocity of $\mathscr{A}$, extracted from $\mathbf{R}_1\mathbf{R}_0^{-1}$ and $\mathbf{u}^*$ is the skew symmetric matrix; i.e., if $\mathbf{u} = (u^x, u^y, u^z)^T$, then

$$\mathbf{u}^* = \begin{pmatrix} 0 & -u^z & u^y \\ u^z & 0 & -u^x \\ -u^y & u^x & 0 \end{pmatrix}$$

4.2 Motion Projection

Given the interpolating motion, $\mathbf{M}(t)$, now we explain a novel technique of how we can calculate the tight upper bound μ of motion. For now, we assume that both the moving $\mathscr{A}$ and fixed objects $\mathscr{B}$ are convex, and the motion bound calculation technique explained here will be applied to non-convex objects in Sec. 5 by decomposing $\mathscr{A}, \mathscr{B}$ into convex objects, calculating the motion bound for each convex pair, and taking its minimum.

Let $\mathbf{p}_i$ be a point on $\mathscr{A}$, and as $\mathscr{A}$ undergoes $\mathbf{M}$, $\mathbf{p}_i$ will trace out a trajectory $\mathbf{p}_i(t)$ in 3D Euclidean space. Consider its velocity $\dot{\mathbf{p}}_i(t) = \mathbf{v} + \omega \times \mathbf{r}_i(t)$, where $\mathbf{v}, \omega$ are the linear velocity of the center of mass (COM) and angular velocity of $\mathscr{A}$, respectively, and $\mathbf{r}_i = \mathbf{p}_i - R_{COM}$ where R_{COM} is the position of the COM. Notice that $\mathbf{v}, \omega$ are constant between the time interval of $[0,1]$ in our case. Then, the *undirected motion bound* μ_u of $\mathscr{A}$ can be expressed as:

$$\mu_u = \max_i \int_0^1 ||\dot{\mathbf{p}}_i(t)||\, dt \tag{4}$$

μ_u can be an upper bound of motion that Eq. 2 uses, but we get a tighter upper bound μ by projecting $\mathbf{p}_i(t)$ onto $\mathbf{n}$, which is the closest direction vector between $\mathscr{A}$ and $\mathscr{B}$:

$$\begin{aligned} &\max_i \int_0^1 |\dot{\mathbf{p}}_i(t) \cdot \mathbf{n}|\, dt \\ &\leq \max_i \int |\mathbf{v} \cdot \mathbf{n} + \omega \times \mathbf{r}_i \cdot \mathbf{n}|\, dt \\ &\leq |\mathbf{v} \cdot \mathbf{n}| + \max_i \int |\omega \times \mathbf{r}_i \cdot \mathbf{n}|\, dt \\ &\leq |\mathbf{v} \cdot \mathbf{n}| + \int \max_i (|\omega \times \mathbf{r}_i \cdot \mathbf{n}|)\, dt \\ &\leq |\mathbf{v} \cdot \mathbf{n}| + \max_{i,t} (|\omega \times \mathbf{r}_i \cdot \mathbf{n}|) \\ &= \mu \end{aligned}$$

Since $\mathbf{v}, \mathbf{n}, \omega$ are constants during the time interval of Δt, calculating μ boils down to maximizing $|\omega \times \mathbf{r}_i \cdot \mathbf{n}|$. Further, it is equivalent to maximizing $|(\mathbf{n} \times \omega) \cdot \mathbf{r}_i|$ since $(\omega \times \mathbf{r}_i) \cdot \mathbf{n} = (\mathbf{n} \times \omega) \cdot \mathbf{r}_i$. Moreover, because $\mathbf{n} \times \omega = \mathbf{c_1}$ is a constant and $\mathbf{r}_i$ exists only on the surface of a convex polytope, maximizing $|\mathbf{c} \cdot \mathbf{r}_i|$ becomes a linear programming problem. As a result, we have:

$$\mu = \max_i |\mathbf{c_1} \cdot \mathbf{r}_i| + c_2 \tag{5}$$

subject to $\mathbf{r}_i \cdot \mathbf{n}_k \leq d_k, k = 1 \ldots |\mathscr{A}|$

where $\mathbf{c_1} = \mathbf{n} \times \omega$, $c_2 = |\mathbf{v} \cdot \mathbf{n}|$, and $|\mathscr{A}|$ is the number of faces in a convex polytope $\mathscr{A}$, and $\mathbf{x} \cdot \mathbf{n}_k = d_k$ is the plane equation for the kth face in $\mathscr{A}$.

There are many known methods to efficiently solve the linear programming in Eq. 5 [29]. In particular, the simplex method or the *hill climbing method* works well in this simple case. The main idea is, given a query direction $\mathbf{c}_1$, we want to find a vector $\mathbf{r}_i$ on the surface of a convex polytope $\mathscr{A}$ whose direction is closest to the direction of $\mathbf{c}_1$. This type of query is also known as support mapping of $\mathbf{c}_1$ [30] or extremal vertex query along $\mathbf{c}_1$ [9]. A simple method that works well in practice is the use of a lookup table as presented in [9]. In this approach, we sample the possible directions of $\mathbf{c}_1$, and precalculate its extremal vertex $\mathbf{r}_i$, and store their associations in a lookup table. At run-time, given $\mathbf{c}_1$, we look up the closest vector $\hat{\mathbf{c}}$ to $\mathbf{c}_1$ in the table and use its associated extremal vertex as a starting point in the hill climbing to further maximize Eq. 5.

5 Hierarchical Advancement

By plugging μ in Eq.5 into the step (6) in the naive CCD algorithm, Alg. 1, we can devise a CCD algorithm for non-convex polyhedra and accelerate its performance considerably. However, due to the quadratic complexity of $O(\eta_{\mathscr{A}}\eta_{\mathscr{B}})$ in Alg. 1, the algorithm does not scale well to complicated non-convex models.

In this section, we present a novel, efficient scheme based on the concept of bounding volume traversal tree (BVTT) that can handle the quadratic complexity in Alg. 1. Now we relax the convexity restrictions of $\mathscr{A}, \mathscr{B}$ and assume that they are non-convex polyhedra.

5.1 Bounding Volume Traversal Tree

As preprocess, our algorithm decomposes each of the given non-convex polyhedra into convex pieces, and recursively builds a bounding volume hierarchy (BVH) where each BV node corresponds to the convex hull of its children nodes (i.e., convex hull tree), as explained in Sec. 3.3.

Given two BVHs ($CHT_{\mathscr{A}}, CHT_{\mathscr{B}}$) of $\mathscr{A}, \mathscr{B}$, when we calculate the distance between $\mathscr{A}, \mathscr{B}$ at given time t, starting from the root nodes of $CHT_{\mathscr{A}}, CHT_{\mathscr{B}}$, pairwise distance calculation between the nodes $h_i^{\mathscr{A}}, h_j^{\mathscr{B}}$ in $CHT_{\mathscr{A}}, CHT_{\mathscr{B}}$ is recursively performed based on Voronoi marching. Each recursion step requires distance calculation $d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})$ between some pair of nodes $h_i^{\mathscr{A}}, h_j^{\mathscr{B}}$ in $CHT_{\mathscr{A}}, CHT_{\mathscr{B}}$, and $d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})$ is compared to the global minimum d of previously calculated $d(h_k^{\mathscr{A}}, h_l^{\mathscr{B}})$'s during the recursion. If $d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}}) < d$,

the recursive traversal for $h_i^{\mathscr{A}}, h_j^{\mathscr{B}}$ continues; otherwise, it stops.

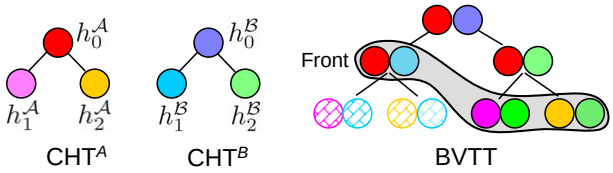

Fig. 2 Bounding Volume Traversal Tree. Two BVHs and their BVTT during distance query. Only solid-colored nodes are actually generated in BVTT.

The bounding volume traversal tree (BVTT) is a special recursion tree generated during the distance query, as illustrated in Fig. 2. BVTT was originally introduced by [17] to measure the cost of a BVH traversal scheme. In our case, each node $h(i,j)$ in a BVTT corresponds to a pair of convex hull nodes $h_i^{\mathscr{A}}, h_j^{\mathscr{B}}$. Furthermore, $d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})$ and the closest direction vector $n(i,j)$ that realizes $d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})$ are also stored at $h(i,j)$. The front in a BVTT refers to a collection of the leaf nodes in the tree, generated during the distance query at run-time. In practice, the size of front is much smaller than the size of all possible combinations between convex pieces in $\mathscr{A}, \mathscr{B}$. We take advantage of this fact to reduce the quadratic complexity in Alg. 1. Algorithms for distance calculation between non-convex polyhedra as well as building its corresponding BVTT are explained in Alg. 3.

Algorithm 3 BVTT
Input: BVH nodes h_i, h_j, Current distance $d_{Current}$
Output: Return the closest distance $d(h_i, h_j)$

```
1: {Initially, BVTT(CHT_A.root, CHT_B.root, ∞) is called. }
2: {A BVTT node h(i, j) is created.}
3: Perform Voronoi marching on h_i, h_j to find d(h_i, h_j) and its closest
   vector n(i, j).
4: if d(h_i, h_j) < d_Current then
5:    if h_j is not a leaf node in CHT_B then
6:       d_1 = BVTT(h_i, h_j.LeftChildTree).
7:       d_2 = BVTT(h_i, h_j.RightChildTree).
8:       return min(d_1, d_2).
9:    else if h_i is not a leaf node in CHT_A then
10:      d_1 = BVTT(h_i.LeftChildTree, h_j).
11:      d_2 = BVTT(h_i.RightChildTree, h_j).
12:      return min(d_1, d_2).
13:   else
14:      {Mark h(i, j) as front and n(i, j) is stored at it.}
15:      return d(h_i, h_j)
16:   end if
17: else
18:   {Mark h(i, j) as front and n(i, j) is also stored at it.}
19:   return d
20: end if
```

5.2 Conservative Advancement for General Polyhedra

Based on the BVTT, we can devise an efficient hierarchical algorithm to perform CA between non-convex polyhedra $\mathscr{A}, \mathscr{B}$. The main idea of this method is based on the following observations:

1. The union of the front nodes $h(i,j)$ in BVTT is a superset of the union of all the pairwise combinations of convex pieces $(c_i^{\mathscr{A}}, c_j^{\mathscr{B}})$ in $\mathscr{A}, \mathscr{B}$.
2. The front nodes $h(i,j)$'s are the nodes that are discovered during closest distance query, so that these nodes have a higher probability than other convex nodes in BVTT to be collided at t_{TOC} or to be reused for successive distance query at next CA step.

The observation (1) guarantees that our algorithm based on performing CA on the front is conservative. Thanks to the observation (2), our algorithm is not overly conservative and as experimentally shown in Sec. 6, it is quite efficient in practice (i.e., few CA iterations). In fact, the observation (2) has been similarly used in discrete distance query when high motion coherence exists. In other words, the *witness convex pieces* that realize closest distance at time step t tend to be a witnesses again for the next time step $t+\Delta t$ when the underlying motion has high coherence [10]. In our case, since our algorithm requires successive invocation of CA steps, it has a similar effect of having high motion coherence and thus the observation (2) is sound.

Based on the aforementioned observations, our algorithm applies an atomic, convex CA operation to each node $h(i,j)$ in the front, estimates its individual time of contact $\Delta t_{TOC}(i,j)$, and takes its minimum as Δt_{TOC}. Then, we advance $\mathscr{A}$ by Δt_{TOC} and iterate another CA step until $d(\mathscr{A}(t), \mathscr{B})$ becomes smaller than a user-provided distance threshold. Finally, at t_{TOC}, we perform one-time discrete distance query to find all the relevant contact features such as face/vertex and edge/edge. This algorithm has been summarized earlier in Alg. 2. The major strength of our hierarchical CA algorithm can be also summarized as follows:

– The main bottleneck in Alg. 2 is the computation of $d(h_i^{\mathscr{A}}, h_j^{\mathscr{B}})$ and $\mathbf{n}(i,j)$. However, these are already calculated as part of distance calculation between $\mathscr{A}$ and $\mathscr{B}$, as presented in Alg. 3. As a result, the hierarchical CA has very little overhead over discrete distance calculation $d(\mathscr{A}(t), \mathscr{B})$ at time t.
– Typically, since the size of the front nodes is small, the step (5) in Alg. 2 has a much lower number of iterations compared to $O(\eta_{\mathscr{A}}\eta_{\mathscr{B}})$.

6 Results and Discussions

Now we present our implementation results and discuss its performance.

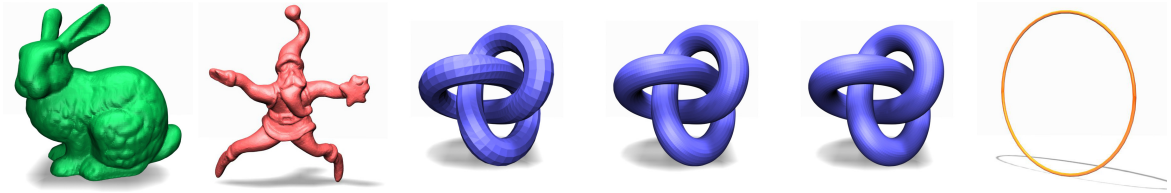

Fig. 3 Benchmarking Models. From left to right, bunny (70K tri.), Santa (37K tri.), torus-knots (2.8K, 11K, 34K tris.), a ring (10K tri.).

6.1 Implementation Results

We have implemented our CCD algorithm using C++ on Windows XP, equipped with Intel P4 3.6GHz CPU and 2GB main memory. We have used a public-domain proximity library, SWIFT++[1] for distance calculation and convex decomposition for non-convex polyhedra, and have modified it to serve our purpose. We also take advantage of the extremal vertex query function provided in SWIFT++ to implement Eq. 5. Thus, in some sense, our implementation can be considered as a continuous version of SWIFT++.

6.2 Benchmarking

In order to measure the performance of our algorithm, we employ different benchmarking models in various complexities (ranging from 10K to 70K triangles) as shown in Fig. 3. These models are highly non-convex, and we use them in different benchmarking scenarios as below. The user-controlled threshold of distance to find t_{TOC} is 0.001 throughout all the experiments.

1. **Santa vs Thin Board** (Fig. 4): Given two configurations of $\mathbf{q}_0, \mathbf{q}_1$ of a highly non-convex Santa model with 37888 triangles, separated by approximately 5 times the bounding box size of Santa, a board (12 triangles) is initially located below $\mathbf{q}_0$. By varying the position of the board toward $\mathbf{q}_1$, we calculate t_{TOC} of the Santa model when it tries to reach from $\mathbf{q}_0$ to $\mathbf{q}_1$. In the figure, the red, blue, and green Santas denote the Santa model at configurations, $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}(t_{TOC})$, respectively.
2. **Bunny vs Bunny** (Fig. 5): Each bunny consists of 69664 triangles. As shown in the figure, 5, one of the bunny is shot from a random configuration $\mathbf{q}_0$ (red) toward a random configuration $\mathbf{q}_1$ (blue) against a fixed bunny (yellow) with constant translational and angular velocities. We perform this test for more than 250 trials, and between [0,100] steps of the test trial, $\mathbf{q}_1$ has only translation relative to $\mathbf{q}_0$. Afterwards, plus the translational motion, $\mathbf{q}_1$ has $\frac{\pi}{3}$ rotation around X axis plus $\frac{\pi}{3}$ rotation around Y axis relative to $\mathbf{q}_0$.
3. **Torusknot vs Torusknot** (Fig. 7): It is similar to the bunny vs bunny benchmarking scenario . However, instead, we use torus-knots in different complexities, 2880, 11520 and 34560 triangles.
4. **Rigid Body Dynamics for Bunnies** (Fig. 8-Left): Using HAVOK^{TM}[2], we perform rigid body dynamics to simulate the falling of a red bunny against a blue one, each consisting of 26K triangles, due to gravity. Each simulation step provides $\mathbf{q}_0, \mathbf{q}_1$ of the blue moving bunny, plug these into our CCD algorithm, and measure the performance. Moreover, in order to create artificial interpenetration at $\mathbf{q}_1$ from this simulation (the simulation itself is collision-free all the time), we slightly modify the $\mathbf{q}_1$ before impact, and thus we can ask our CCD algorithm to find t_{TOC}.
5. **Rigid Body Dynamics for Rings** (Fig. 8-Right): We conduct similar dynamics experiments for a ring consisting of 10K triangles.

The resulting performance of our algorithm, named as FAST[3], for the above benchmarks is shown in Fig.'s 4, 5, 7 and 8: it includes timings (in milli-seconds) and the number of CA iterations of our algorithm (FAST). We also measure the performance of a known CCD method, CONTACT [24], for each benchmarking scenario and compare it with ours. CONTACT can be thought of as a continuous version of the OBB algorithm [12] and uses a screwing formulation for motion interpolation. As a result, our algorithm and CONTACT do not perform the exactly same CCD operation but both of the algorithms are able to calculate t_{TOC} as well as contact features. In our benchmarking scenarios, our approach outperforms CONTACT by a factor of $1.4 \sim 45.5$ depending on benchmarking scenarios.

6.3 Analysis

Now we briefly analyze the computational complexities of our algorithm (Alg. 2). For each CA step (4-12),

– The complexity of step (4) is contact-dependent, and in the worst case, it can take $O(|\mathscr{A}||\mathscr{B}|)$ where $|\mathscr{A}|, |\mathscr{B}|$ are the number of faces in $\mathscr{A}, \mathscr{B}$, respectively. However, in practice, its computational cost is much lower than the quadratic complexity, since our implementation based on SWIFT++ utilizing BVH and high motion coherence between successive CA iterations. The use of motion coherence is made possible thanks to the technique known as *front tracking* [10].

[1] *http://www.cs.unc.edu/ geom/SWIFT++*

[2] *http://www.havok.com/*

[3] It is available to download from *http://graphics.ewha.ac.kr/FAST*

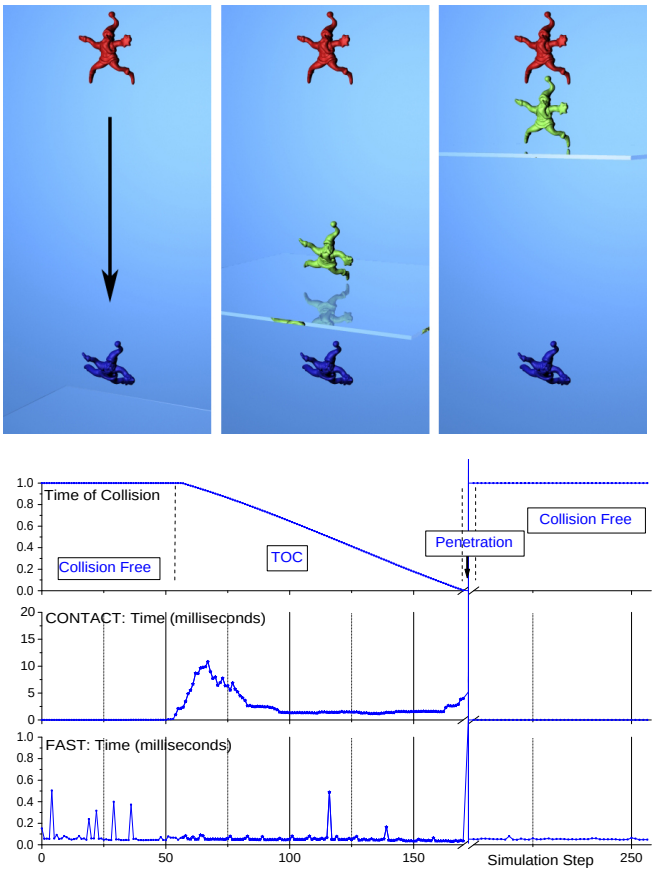

Fig. 4 Benchmarking 1: Santa vs Thin Board. The average CCD query performance is 15460 FPS.

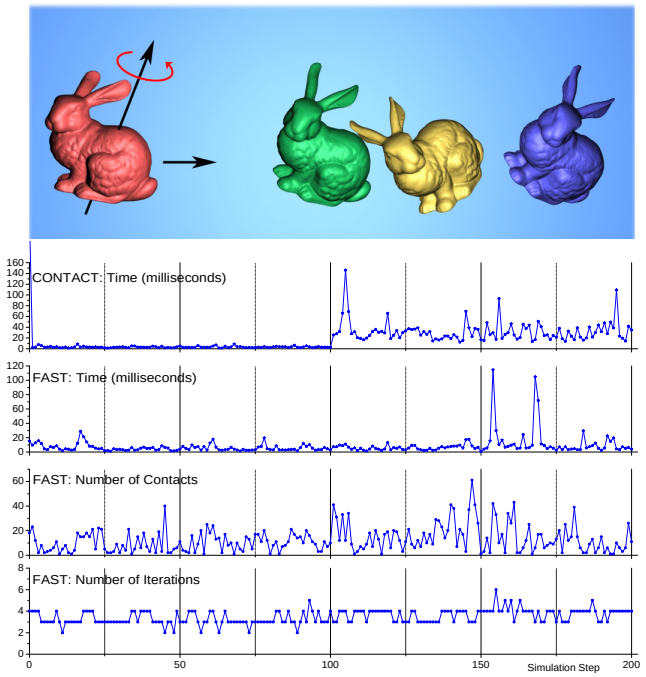

Fig. 5 Benchmarking 2: Bunny vs Bunny. The average CCD performance is 125 FPS.

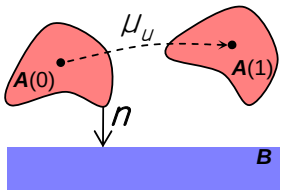

Fig. 6 $\mathscr{A}$ is hovering above $\mathscr{B}$.

– The number of loops in step (5) is also contact-dependent, and in the worst case, it is $O(\eta^{\mathscr{A}}\eta^{\mathscr{B}})$. Inside the loop, the step (7) of performing motion bound is dominated by the linear programming required by Eq. 5. In the worst case, linear programming can take a linear time in terms of the number of faces in a convex hull node, $h_i^{\mathscr{A}}$. However, since we are using a table lookup-based method, in practice, this operation takes an almost constant time.

Bounding the number of CA steps is in general quite difficult since there are many parameters involved with this computation such as the relative configurations of $\mathscr{A},\mathscr{B}$, the initial and final configurations $\mathbf{q}_0,\mathbf{q}_1$ of $\mathscr{A}$, the distance and contact features. However, based on our experiments, the typical number of iterations is 4 on average.

6.4 Comparison

We qualitatively compare the cons and pros of our CCD algorithm with other methods, in particular bisection-based algorithms such as [24,28], because these techniques are known to be faster than other known methods explained in Sec. 2.

CONTACT [24]: As mentioned earlier, CONTACT is a continuous version of OBB so that it can handle generic polygon soups unlike ours. However, it has two problems: 1) the continuous OBB test can be overly conservative when the underlying motion has large rotations, 2) CONTACT does not perform adaptive bisection such that it may suffer from a large number of bisection steps especially when objects are far apart. In our CCD algorithm, even though the motion has large rotation, our motion projection technique can handle such a case. Moreover, by taking advantage of the closest direction vector, our algorithm performs fewer advancement iterations.

Dynamic Collision Checker [28]: The dynamic collision checker performs similarly to our CCD algorithm in that it relies on motion bound calculation and distance calculation. Compared to ours, however, the algorithm has a less tight upper bound of (undirected) motion bound without using the information of closest direction vector (essentially μ_u in Eq. 4) and less tight lower bound of distance calculation based on PQP [17]. As a result, it will have worse performance than ours. In particular, as illustrated in Fig. 6, when

$\mathscr{A}$ hovers above $\mathscr{B}$, since the undirected motion bound μ_u calculated by the dynamic collision checker does not consider the closest direction vector **n**, many iterations will be required to satisfy Eq. 2 to realize that $\mathscr{A}(t)$ is collision-free. However, this algorithm is able to handle polygon soups.

7 Conclusion

In this paper, we have presented a highly interactive CCD algorithm for complex, non-convex polyhedra. The algorithm is based on efficient calculation of motion bound and hierarchical advancement. There are a few limitations in our algorithm. First of all, our algorithm is applicable to only 2-manifold, polyhedral models, not to polygon soups. Secondly, the major bottleneck of our algorithm is distance calculation between two non-convex polyhedra, and this algorithm heavily depends on a convex decomposition scheme. As a result, a high number of convex pieces in the convex decomposition can degrade the CCD performance. For future work, there are many avenues that we will like to pursue. First of all, we would like to apply our fast CCD algorithm to constraint-based rigid dynamics, 6DOF haptic rendering, and robot motion planning to significantly improve their performance. In particular, we expect high performance improvement from a PRM-based motion planning method that requires finding a collision-free path. Finally, we will like to extend our CCD algorithm to articulated bodies such as [27].

Acknowledgements

This project was supported in part by the grant 2004-205-D00168 of KRF, the STAR program of MOST and the ITRC program. We would like to thank Stephane Redon for sharing the CONTACT software with us.

References

1. Abdel-Malek, K., Blackmore, D., Joy, K.: Swept volumes: Foundations, perspectives, and applications. International Journal of Shape Modeling (2002)
2. Agarwal, P.K., Basch, J., Guibas, L.J., Hershberger, J., Zhang, L.: Deformable free space tiling for kinetic collision detection. pp. 83–96 (2001)
3. Baraff, D.: Fast contact force computation for nonpenetrating rigid bodies. In: A. Glassner (ed.) Proceedings of SIGGRAPH '94, pp. 23–34. ACM SIGGRAPH (1994). ISBN 0-89791-667-0
4. Baraff, D., Witkin, A.: Physically-Based Modeling. ACM SIGGRAPH Course Notes (2001)
5. van den Bergen, G.: Ray casting against general convex objects with application to continuous collision detection. Journal of Graphics Tools (2004)
6. Canny, J.F.: Collision detection for moving polyhedra. IEEE Trans. PAMI **8**, 200–209 (1986)
7. Choi, Y.K., Wang, W., Liu, Y., Kim, M.S.: Continuous collision detection for elliptic disks. IEEE Transactions on Robotics (2006)
8. Dobkin, D.P., Kirkpatrick, D.G.: Determining the separation of preprocessed polyhedra – a unified approach. In: Proc. 17th Internat. Colloq. Automata Lang. Program., *Lecture Notes in Computer Science*, vol. 443, pp. 400–413. Springer-Verlag (1990)
9. Ehmann, S., Lin, M.C.: Accelerated proximity queries between convex polyhedra using multi-level voronoi marching. Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems pp. 2101–2106 (2000)
10. Ehmann, S., Lin, M.C.: Accurate and fast proximity queries between polyhedra using convex surface decomposition. Computer Graphics Forum (Proc. of Eurographics'2001) **20**(3), 500–510 (2001)
11. Gilbert, E.G., Johnson, D.W., Keerthi, S.S.: A fast procedure for computing the distance between complex objects. Internat. J. Robot. Autom. **4**(2), 193–203 (1988)
12. Gottschalk, S., Lin, M., Manocha, D.: OBB-Tree: A hierarchical structure for rapid interference detection. In: H. Rushmeier (ed.) SIGGRAPH 96 Conference Proceedings, Annual Conference Series, pp. 171–180. ACM SIGGRAPH, Addison Wesley (1996). Held in New Orleans, Louisiana, 04-09 August 1996
13. Guibas, L., Hsu, D., Zhang, L.: *H-Walk*: Hierarchical distance computation for moving convex bodies. Proc. of ACM Symposium on Computational Geometry (1999)
14. Kim, B., Rossignac, J.: Collision prediction for polyhedra under screw motions. In: ACM Conference on Solid Modeling and Applications (2003)
15. Kim, D., Guibas, L., Shin, S.: Fast collision detection among multiple moving spheres. IEEE Trans. Vis. Comput. Graph. **4**(3), 230–242 (1998)
16. Kirkpatrick, D., Snoeyink, J., Speckmann, B.: Kinetic collision detection for simple polygons. pp. 322–330 (2000)
17. Larsen, E., Gottschalk, S., Lin, M., Manocha, D.: Fast proximity queries with swept sphere volumes. Tech. Rep. TR99-018, Department of Computer Science, University of North Carolina (1999)
18. Lin, M., Manocha, D.: Collision and proximity queries. In: Handbook of Discrete and Computational Geometry (2003)
19. Lin, M.C.: Efficient collision detection for animation and robotics. Ph.D. thesis, University of California, Berkeley, CA (1993)
20. Mirtich, B.: Timewarp rigid body simulation. SIGGRAPH 00 Conference Proceedings pp. 193–200 (2000)
21. Mirtich, B.V.: Impulse-based dynamic simulation of rigid body systems. Ph.D. thesis, University of California, Berkeley (1996)
22. Ortega, M., Redon, S., Coquillart, S.: A six degree-of-freedom god-object method for haptic display of rigid bodies. In: IEEE International Conference on Virtual Reality (2006)
23. Redon, S., Kheddar, A., Coquillart, S.: An algebraic solution to the problem of collision detection for rigid polyhedral objects. Proc. of IEEE Conference on Robotics and Automation (2000)
24. Redon, S., Kheddar, A., Coquillart, S.: Fast continuous collision detection between rigid bodies. Proc. of Eurographics (Computer Graphics Forum) (2002)
25. Redon, S., Kheddar, K., Coquillart, S.: Gauss' least constraints principle and rigid body simulation. Proceedings of International Conference on Robotics and Automation (2002)
26. Redon, S., Kim, Y.J., Lin, M.C., Manocha, D.: Interactive and continuous collision detection for avatars in virtual environments. In: Proceedings of IEEE Virtual Reality
27. Redon, S., Kim, Y.J., Lin, M.C., Manocha, D.: Fast continuous collision detection for articulated models. In: Proceedings of ACM Symposium on Solid Modeling and Applications (2004)
28. Schwarzer, F., Saha, M., Latombe, J.C.: Exact collision checking of robot paths. In: Workshop on Algorithmic Foundations of Robotics (WAFR) (2002)
29. Seidel, R.: Linear programming and convex hulls made easy. pp. 211–215 (1990)
30. van den Bergen, G.: Proximity queries and penetration depth computation on 3d game objects. Game Developers Conference (2001)

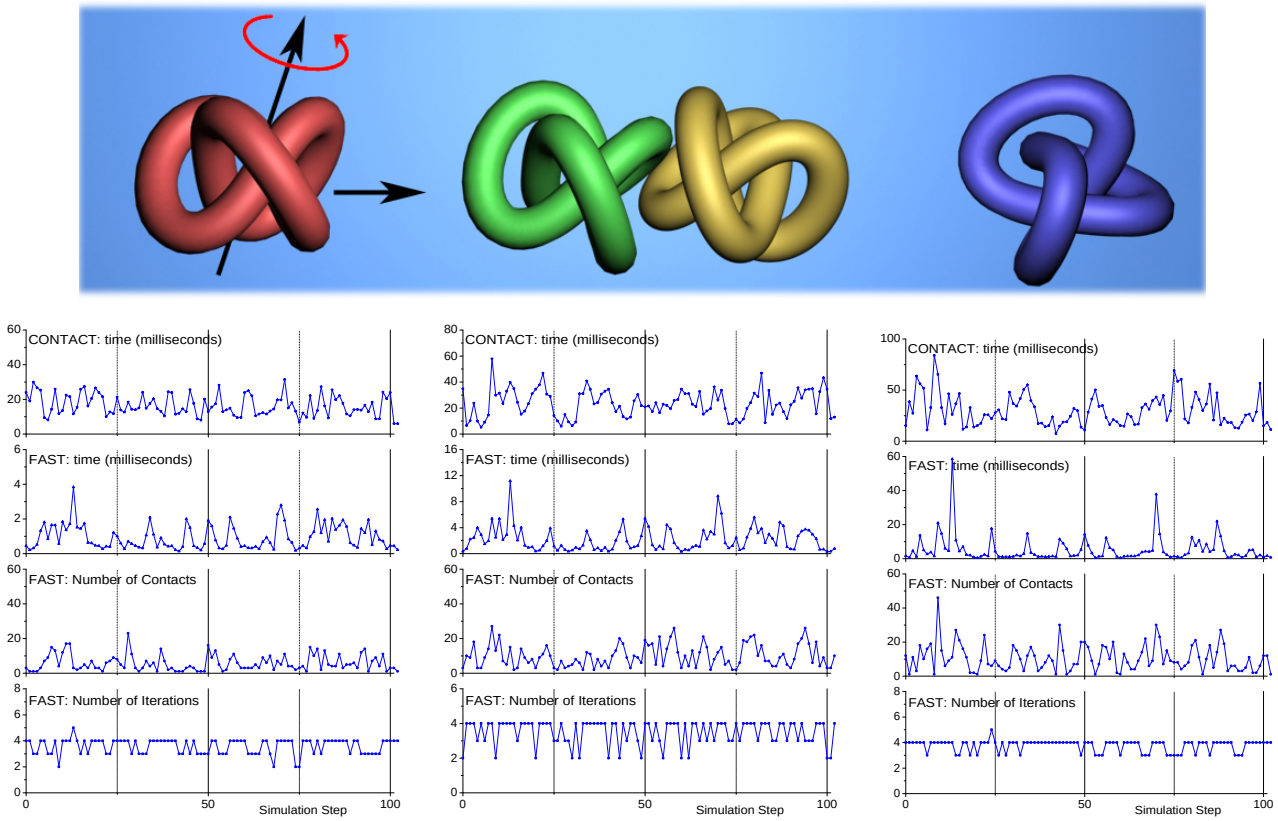

Fig. 7 Benchmarking 3: Torusknots vs Torusknots in different Complexities. 2.8K, 11K, 34K triangles from left to right. The average CCD query performance is 1055, 449, 188 FPS, respectively.

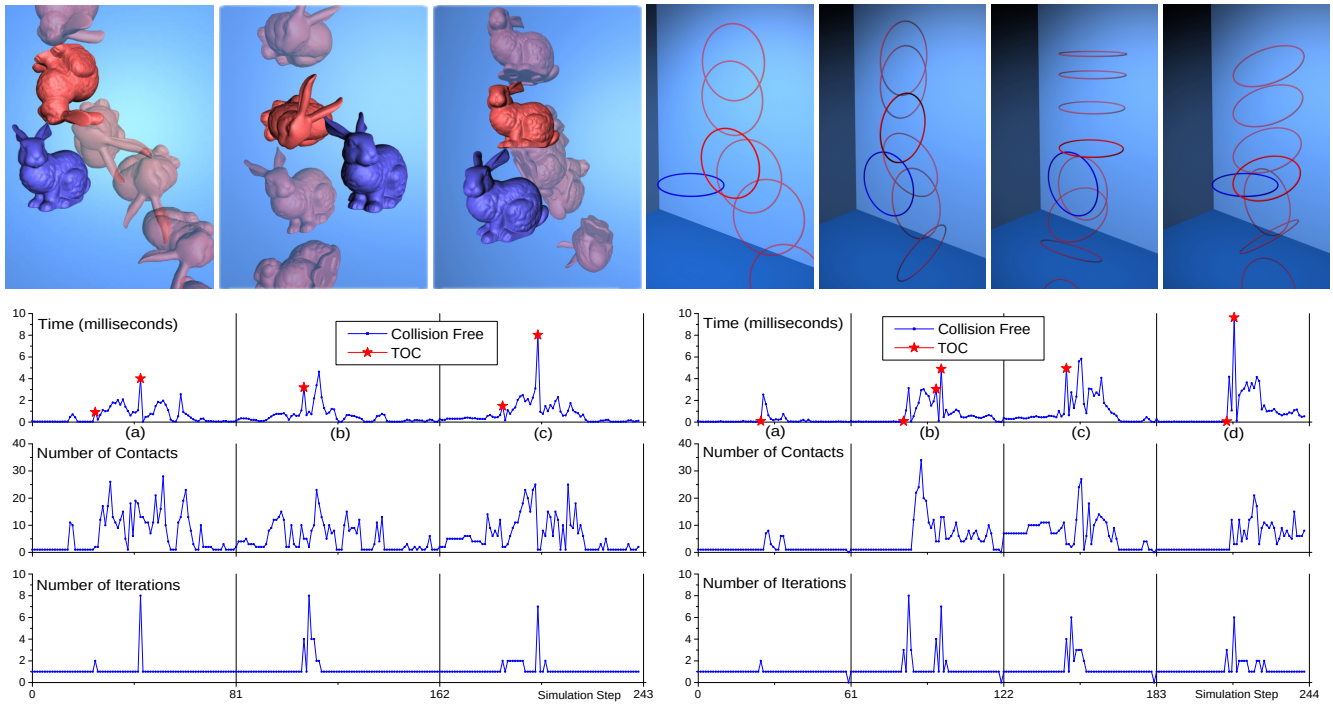

Fig. 8 Benchmarking 4 and 5: Rigid Body Dynamics for Bunnies and Rings. The average CCD query performance is 1555 (bunny) and 1357 (ring) FPS.