Hierarchical and Controlled Advancement for Continuous Collision Detection of Rigid and Articulated Models

Min Tang, *Member, IEEE,* and Dinesh Manocha, *Fellow, IEEE,* and Young J. Kim, *Member, IEEE*

Abstract—We present fast CCD algorithm for general rigid and articulated models based on conservative advancement. We have implemented the CCD algorithm with two different acceleration techniques which can handle rigid models, and have extended one of them to articulated models. The resulting algorithms take a few milliseconds for rigid models with tens of thousands of triangles, and a few milliseconds for articulated models with tens of links. We show that the performance of our algorithms is much faster than existing CCD algorithms for polygon-soup models and it is also comparable to competing CCD algorithms that are limited to manifold models. The preliminary version of this paper appeared in [1].

Index Terms—Continuous Collision Detection, Conservative Advancement, Distance Computation.

1 Introduction

Collision detection and distance computation are important problems in computer graphics, robotics, CAD, etc. In particular, reliable and fast collision detection algorithms are required to enforce non-penetration constraints in motion planning and physics-based animation. Collision detection has been extensively studied over the past two decades. Most early approaches focused on static objects, while recent research has considered moving objects. In some scenarios, the entire trajectory of an object is known in advance, but in most applications, we only know the position of the objects at a few discrete locations in space. For example, in sampling-based motion planning, randomized planners generate collision-free configurations using sampling algorithms, from which they construct a continuous collision-free path from the initial configuration to the goal.

At a broad level, dynamic collision detection algorithms can be subdivided into two categories: *discrete* and *continuous*. Discrete algorithms only check for collisions at sample configurations using static collision detection algorithms. In consequence, they may miss a collision that occurs between two successive configurations. This is sometimes known as the tunneling problem, because it commonly occurs when a rapidly moving object passes undetected through a thin obstacle. Continuous collision detection (CCD) algorithms avoid the tunneling problem by interpolating a continuous motion between successive configurations and checking for collisions along the whole of that motion. If a collision occurs, the first time of contact (ToC) between the moving objects is reported.

CCD algorithms have been used in virtual environments [2], as well as for local planning in sampling-based motion planning [3], [4], [5], [6], where it is used to find the first time of contact and then apply responsive forces in dynamics simulation [7]. The main drawback of CCD algorithms are that they are typically much slower than their discrete counterparts, which limits their applicability. Thus many well-known libraries for sample-based motion planning, such as MSL (http://msl.cs.uiuc.edu/msl/), mostly use discrete collision checking for local planning. Moreover, the fastest CCD algorithms [8], [9] are only applicable to well-behaved polyhedral models, with the exception of our previously reported technique [1]. Thus these approaches are not generally suitable for the 'polygon-soup' models with arbitrary geometry and topology that are widely used in computer graphics, robotics, and physically-based modeling. The preliminary version of this paper appeared in [1].

1.1 Main Results

In this paper, we present a simple algorithm that performs CCD for rigid and articulated models undergoing rigid motions at interactive rates. We make no assumptions about the geometric representation of the model, as long as the model is triangulated (i.e. a polygon-soup). Our new algorithm is an extension of the conservative advancement (CA) technique, which was initially proposed for convex polytopes [10], [11], and is described in Sec. 3. The CA formulation includes two main components: a distance computation

• *Min Tang and Young J. Kim are with the Department of Computer Engineering, Ewha Womans University, Seoul, South Korea. Dinesh Manocha is with the Department of Computer Science at the University of North Carolina at Chapel Hill, U.S.A. Young J. Kim is the corresponding author. E-mail: {tangmin,kimy}@ewha.ac.kr and dm@cs.unc.edu*

and a motion-bound calculation. The former can be performed by any algorithm that computes the separation distance of polygonal models. We use the well-known algorithm based on swept sphere volumes (SSV), which is available as part of the PQP library [12]. We couple the distance computation with a new, efficient analytic method of calculating the motion bound of SSVs, which is described in Sec. 4.2. We also present techniques to improve the performance of our CCD algorithm using two different techniques: controlled advancement (C^2A), described in Sec. 4.3.2, and hierarchical CA (HCA) that is described in Sec. 4.3.3. We compare these two techniques in Sec. 4.3.4.

In Sec. 5, we extend our CCD algorithm to articulated models articulated models, since, unlike other algorithms [9], our algorithm is not restricted to manifold objects. We have implemented our algorithm and measured its performance on several benchmark models of different complexities. Our experiments show that our algorithms can perform CCD in fractions of a millisecond on either rigid or articulated models consisting of tens of thousands of triangles. This is faster than the previous CCD algorithm for rigid models, which can handle only polyhedral models [8]; moreover, our algorithm offers very non-trivial performance improvements over previous work on polygon-soup, rigid models. Our algorithm maintains this performance improvement when applied to objects where the manifold assumption is not satisfied [9], which previous algorithms were unable to achieve at all.

2 Previous Work

We briefly survey prior work on continuous collision detection for both rigid, articulated and deformable models, and for motion-bound computations.

2.1 Continuous Collision Detection

At a broad level, CCD algorithms can be classified into: algebraic equation solvers [13], [14], [15], [16], swept-volume formulations [17], adaptive bisection approaches [7], [3], kinetic data structures (KDS) [18], [19], [20], Minkowski sum formulations [21] and conservative advancement [10], [11], [8], [1], [22].

Most of these approaches are unable to perform fast CCD queries on general polygonal models, although some can handle polygon-soup models [16], [3]. Redon et al. [16] use a continuous version of the separating axis theorem to extend the static OBBtree algorithm [23] to CCD, and demonstrate real-time performance on polygonal models. But the algorithm becomes overly conservative when there is a large rotation between two configurations. In practice, there is a faster algorithm [8] for polyhedral models. [1] uses controlled advancement to detect collision for polygon-soup model, our paper extends this work by using different acceleration techniques and also is applicable articulated models. FCL [24] is a reimplemention of [1], which forms part of a generic collision detection library. A variant CCD query, known as a connection collision query, was used for local planning in sampling-based motion planners [6].

For articulated models, Zhang et al. have extended their approach [8] to articulated models [9], but each link in a model must be a polyhedron, which limits its applicability. Redon et al. [25] describe an extension of their previous algorithm [16] to articulated models, but this algorithm is rather slow for complicated objects. A conservative condition is used in [3] to guarantee a collision-free motion between two configurations, but such a condition is likely to become overly conservative when an object slides over another object.

CCD algorithms for deformable objects find all the times of contact for pairs of overlapping primitive during the motion, while the CCD algorithms for rigid model and articulated model only finds the first time of contact. Most of CCD algorithms for deformable objects focus on how to reduce self-collision tests [26], [27] and redundant elementary tests [28]. Since CCD for rigid models needs not detect self-collision and elementary tests are unnecessary, the CCD algorithms designed for deformable bodies may not be effectively applied to rigid or articulated models.

2.2 Motion Bound Calculation

Schwarzer et al. [3] proposed a method to bound the trajectory of an object moving with constant velocities of translation and rotation. An upper bound on the motion trajectory can be computed by taking the weighted sum of differences between all the configuration parameters along the trajectory. Although the bound is used for local planning in the MPK motion planning library, it is quite conservative and is aimed mainly at models with combinations of prismatic and revolute joints.

The trajectory of linear swept spheres (LSS) can be bounded using interval arithmetic [2], and the resulting bound has been used to detect collisions between a moving robot and the virtual environment. However, this method does not consider how closely the robot and the obstacles are placed to one another (i.e. it is an undirected motion bound), and thus the bound calculation does not utilize the motion trajectory effectively.

Lin [10] and Mirtich [11] propose a ballistic motion as a motion trajectory for CCD, and compute a directional motion bound along the direction of least distance between two convex polytopes by bounding the ballistic rotational velocity. Zhang et al. [8] use an extremal vertex query to find a directional motion bound for an object moving with a constant velocity of translation and rotation. Tang et al. [6] propose a directional motion bound for screw motions. Pan et

al. [29] find a directional motion bound for a cubic B-spline motion trajectory.

3 Overview

We first introduce our notation and the terminology used throughout this paper. Next, we briefly explain the basic idea of conservative advancement (CA) and give an overview of our algorithm.

3.1 Preliminaries

Let $\mathcal{A}$ and $\mathcal{B}$ be two rigid polygon-soup models in 3D, where $\mathcal{A}$ is moving under a rigid transformation $\mathbf{M}(t)$. Without loss of generality, we can assume that $\mathcal{B}$ is fixed. The initial and final configurations of $\mathcal{A}$ are $\mathbf{q}_0$ and $\mathbf{q}_1$ at times $t = 0$ and $t = 1$ respectively. We also define $\mathcal{A}(t) = \mathbf{M}(t)\mathcal{A}(0)$. The CCD problem is that of deciding whether there is a feasible solution to:

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

If this equation has a solution, we also compute the minimum value of t that satisfies it. This is the first time of contact, which is called τ.

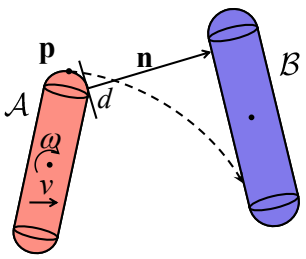


Fig. 1. **Conservative advancement of linear swept sphere.**

CA (conservative advancement) is a simple technique that computes a lower bound on the value of τ for two convex objects $\mathcal{A}$ and $\mathcal{B}$ by repeatedly advancing $\mathcal{A}$ by Δt_i towards $\mathcal{B}$ while avoiding collisions [10], [11]. Successive values of Δt_i are calculated based on a lower bound on the closest distance $d(\mathcal{A}(t), \mathcal{B})$ between $\mathcal{A}(t)$ and $\mathcal{B}$, and an upper bound μ on the motion of $\mathcal{A}(t)$ projected on to $d(\mathcal{A}(t), \mathcal{B})$ (as Fig. 1), as follows:

$$\Delta t_i \quad \leq \quad \frac{d(\mathcal{A}(t), \mathcal{B})}{\mu}. \tag{2}$$

The first time of contact τ can be obtained by summing the time-steps Δt_i until $d(\mathcal{A}(\tau), \mathcal{B})$ becomes less than some user-specified threshold. CA only works for convex objects. We extend it to arbitrary polygon-soup models using swept sphere volume (SSV) hierarchies.

3.2 Our Approach

We assume that only the initial and final configurations of $\mathcal{A}$ are given as $\mathbf{q}_0, \mathbf{q}_1$. Using these configurations, we compute a continuous motion $\mathbf{M}(t)$ to interpolate $\mathbf{q}_0, \mathbf{q}_1$ with constant translational and rotational velocities. If we know the actual motion of $\mathcal{A}$ *a priori*, for instance [20], we can approximate it in a piecewise linear manner. A similar interpolation method has been used previously [7], [8]. When the simulation time-step is small, the differences between the actual objects' motions and the interpolated paths are negligible [7].

We precompute a bounding volume hierarchy (BVH), here the SSV hierarchy, for the input polygon-soup model. SSV can have one of the three forms: a point-swept sphere (PSS), a line-swept sphere (LSS) or a rectangle-swept sphere (RSS). These bounding volumes contain sets of polygon primitives. The root-level SSV node bounds the entire set of primitives, and the SSV hierarchy is recursively built by partitioning the polygonal primitives and bounding each partition with an SSV node [12].

At runtime, we apply CA to the nodes in the SSV hierarchy in a selective manner. As formulated in Eq. 2, CA requires that a closest distance and a motion bound be calculated. Distance between SSVs can be a byproduct of the SSV algorithm. In Sec. 4.2 we will show how to compute a tight upper bound μ on the motion of an SSV. The efficiency of our CCD algorithm depends on applying CA to a good choice of nodes in the hierarchy. We use two strategies to select nodes: controlled CA (C^2A) and hierarchical CA (HCA).

One approach is to select the nodes at which the BVH traversal terminates during the closest distance query; there are the *front nodes*. Experimentally, these nodes are more likly to realize τ than the others. However, the front nodes are likely to be deep in the hierarchy, and thus the number of front nodes can be rather large. This affects the performance of our CCD algorithm, since we need to apply CA to these nodes repeatedly. In Sec. 4.3.2, we propose a scheme, called controlled CA (C^2A), to control the depth of the front nodes during the CA iterations, thereby improving the performance of the CCD algorithm significantly. The main idea is that during the first few CA iterations, we do not need to compute the closest distance exactly, but in later CA iterations we perform exact distance computations. The choice of the level of distance approximation determines the depth of the front nodes and the number of nodes which become involved in our computation.

An alternative strategy is to use the nodes visited by the BVH traversal while determining the minimum ToC. The ToC between two objects is the minimum ToC of all mutual pairs of primitives. We can perform a ToC query by traversing the BVHs for the two objects. During the traversal, ToCs between leaf nodes

(primitive pairs) are calculated and used to reduce the current minimum ToC τ_{cur} for the entire objects. The ToC between a pair of bounding volumes (BVs) is a lower bound on the ToC for all their enclosed primitive pairs, since the BVs will collide before their bounded triangles. If the ToC between a BV pair is later than τ_{cur}, then the BV pair and all the primitives that they bound can be culled, since no ToC between the primitive pairs that they enclose can reduce τ_{cur}. In this scheme, the ToC is obtained by hierarchically traversing BVHs. We call this method hierarchical conservative advancement (HCA), and details will be given in Sec. 4.3.3.

4 CCD FOR RIGID MODELS

4.1 Motion Interpolation

Given initial $\mathbf{q}_0$ and final configurations $\mathbf{q}_1$ for $\mathcal{A}$, CCD algorithms require a motion $\mathbf{M}(t)$ that interpolates $\mathbf{q}_0$ and $\mathbf{q}_1$. In our case, we use a linearly interpolating motion in configuration space. Without loss of generality, we assume that the global frame $\mathbf{o}$ is the same as the local frame attached to $\mathcal{A}$ at $\mathbf{q}_0$. In this case, $\mathbf{q}_0 = (\mathbf{I}, \mathbf{0})$ and $\mathbf{q}_1 = (\mathbf{R}_1, \mathbf{T}_1)$ in $SE(3)$, where $\mathbf{R}_1$ is the rotation matrix, and $\mathbf{T}_1$ is the translation vector. The interpolating motion $\mathbf{M}(t), t \in [0,1]$ is a linear motion with a constant translational velocity $\mathbf{v} = \mathbf{T}_1$, and constant angular velocity $\omega = (\mathbf{u}, \theta)$ where the quaternion component of $\mathbf{q}_1$ is $[cos(\frac{1}{2}\theta), sin(\frac{1}{2}\theta)\mathbf{u}]$.

4.2 Motion Bound Calculation

We now present our algorithm to compute a motion bound μ on swept sphere volumes (SSVs) and triangle primitives undergoing a rigid transformation $\mathbf{M}(t)$ as mentioned above, all of which pass through the origin of body frame. In this section, we assume that all the vectors are defined in world frame.

A bounding volume SSV consists of the 3D Minkowski sums of a sphere with a point, line segment and rectangle. These are called PSS, LSS and RSS respectively. Fig. 3 is a 2D illustration of SSV. Let α and β denote a BV or a triangle primitive of $\mathcal{A}$ and $\mathcal{B}$, respectively. Further, let $\mathbf{p}_i$ be a point on α, $\mathbf{n}$ be the closest direction between α, β, and $\mathbf{r}_i$ be a vector from the origin of the local body frame $\mathbf{o}_\mathcal{A}$ attached to $\mathcal{A}$, to $\mathbf{p}_i$. As mentioned in [8], the maximum length of the trajectory of α (*i.e.* the motion bound) projected on to the direction of $\mathbf{n}$ is:

$$\begin{aligned}\mu &= \max_i \left(\max_{t\in[0,1]} \dot{\mathbf{p}}_i(t) \cdot \mathbf{n} \right) \\ &\leq \mathbf{v}\cdot\mathbf{n} + \max_i \left(\max_{t\in[0,1]} |\omega \times \mathbf{n} \cdot (\mathbf{R}(t)\mathbf{r}_i)| \right) \\ &= \mathbf{v}\cdot\mathbf{n} + \max_i \left(\max_{t\in[0,1]} |\mathbf{L}(t)\cdot\mathbf{r}_i| \right) \\ &\leq \mathbf{v}\cdot\mathbf{n} + \|\omega\times\mathbf{n}\| \max_i \left(\|\mathbf{r}_i^L\| \right),\end{aligned} \tag{3}$$

where $\mathbf{L}(t) = \mathbf{R}^{-1}(t)(\mathbf{n}\times\omega)$, and $\mathbf{R}$ is the rotational component of $\mathbf{M}(t)$. Thus, $\mathbf{R}(0) = \mathbf{I}$ and $\mathbf{R}(1) = \mathbf{R}_1$. Since ω is the rotation axis of $\mathbf{R}(t)$, $\mathbf{R}^{-1}(t)\omega = \omega$. The first term in this equation is constant, and our goal is to obtain the second term. Since

$$\begin{aligned}\mathbf{L}(t) &= \mathbf{R}^{-1}(t)(\mathbf{n}\times\omega) \\ &= \mathbf{R}^{-1}(t)\mathbf{n} \times \mathbf{R}^{-1}(t)\omega \\ &= \mathbf{R}^{-1}(t)\mathbf{n} \times \omega,\end{aligned} \tag{4}$$

the vectors $\mathbf{L}(t)$ are coplanar and form a plane L with a normal ω. The maximum value of the second term in Eq. 3 can be obtained by determining the maximum of the projections of $\mathbf{r}_i$ onto L, which we call $\mathbf{r}_i^L$. An example of the projection of a point bounded by a PSS on L is shown in Fig. 2.

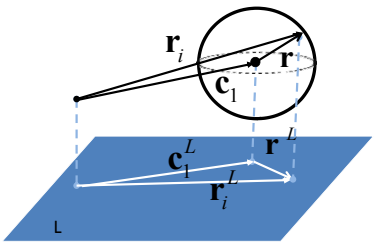


Fig. 2. **Projection $\mathbf{r}_i^L$ of $\mathbf{r}_i$ bounded by a PSS.**

Lemma 4.1 *Let α be an SSV. Then the directional motion bound μ of α is expressed as:*

$$\mu \leq \mathbf{v}\cdot\mathbf{n} + \|\omega\times\mathbf{n}\| \left(r + \max_i \left(\|\mathbf{c}_i^L\| \right) \right),$$

where

$$\|\mathbf{c}_i^L\| = \frac{\|\mathbf{c}_i \times \omega\|}{\|\omega\|},$$

r is the radius of the sphere used to construct the SSV, and $\mathbf{c}_i$ are the vectors from $\mathbf{o}_\mathcal{A}$ to the endpoints of the generator primitives of SSV. These are a point, line, and rectangle for the Minkowski sums PSS, LSS and RSS respectively, for which $i = 1,\ 1\ldots2,$ and $1\ldots4$.

Proof: We start by representing $\mathbf{r}_i$ analytically for the three kinds of SSV as follows (see also Fig. 3):

- For a PSS (Fig. 2), $\mathbf{r}_i = \mathbf{c_1} + \mathbf{r}$, where $\mathbf{c}_1$ is the vector from $\mathbf{o}_\mathcal{A}$ to the center of the PSS, and $\mathbf{r}$ is some point on a sphere centered at the origin with a radius of r.
- For an LSS, $\mathbf{r}_i \in (s\mathbf{c_1} + (1-s)\mathbf{c_2} + \mathbf{r})$ and $s \in [0,1]$, where $\mathbf{c_1}$ and $\mathbf{c_2}$ are the vectors form $\mathbf{o}_\mathcal{A}$ to the endpoints of the line segment used to construct LSS. The definition of $\mathbf{r}$ follows that of the PSS.
- For an RSS, $\mathbf{r}_i \in (s\mathbf{c_1} + (1-s)\mathbf{c_2} + \mu(\mathbf{c_3} - \mathbf{c_1})) + \mathbf{r}$, $s \in [0,1]$ and $\mu \in [0,1]$ where $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{c_3}$ are the vectors from $\mathbf{o}_\mathcal{A}$ to the three defining corners of the rectangle used to construct the RSS. The definition of $\mathbf{r}$ follows that of the PSS.

Furthermore, we can say that $\mathbf{r}_i = \mathbf{r} + \mathbf{k}$ for all three types of the SSV, where $\mathbf{k}$ is the vector from the origin of the body frame of $\mathcal{A}$ to a point on the generator of SSV. Hence:

$$\begin{aligned} &|\mathbf{L}(t) \cdot \mathbf{r}_i| = \left|\mathbf{L}(t) \cdot \mathbf{r}_i^L\right| \\ &\leq \|\mathbf{L}(t)\| \cdot \left\|\mathbf{r}_i^L\right\| \\ &\leq \|\mathbf{L}(t)\|\, r + \|\mathbf{L}(t)\| \left\|\mathbf{k}^L\right\|, \end{aligned} \tag{5}$$

where $\mathbf{k}^L$ is the projection of $\mathbf{k}$ on to the plane L. The projection of a point onto a plane is always a point; the projection of a line segment onto a plane can be a line or a point; and the projection of a rectangle onto a plane is a quadrangle or a line segment. Since the projection of $\mathbf{k}$ to L has to be inside the projection of the generator primitive of the SSV on to L, we obtain the following relationship:

$$\left\|\mathbf{k}^L\right\| \leq \max(\left\|\mathbf{c}_i^L\right\|) \;, \quad \left\|\mathbf{c}_i^L\right\| \;= \frac{\|\mathbf{c}_i \times \omega\|}{\|\omega\|} \tag{6}$$

By combining Eq.s 3, 5, and 6, we obtain the result of the lemma. □

We can compute a motion bound for a triangle primitive by projecting its vertices onto L, so as to bound the range of projection.

Lemma 4.2 *Let α be a triangle primitive. Its directional motion bound can be expressed as follows:*

$$\mu \leq \mathbf{v} \cdot \mathbf{n} + \|\omega \times \mathbf{n}\| \left(\max\left(\left\|\mathbf{c}_i^L\right\|\right)\right),$$

where

$$\left\|\mathbf{c}_i^L\right\| = \frac{\|\mathbf{c}_i \times \omega\|}{\|\omega\|}$$

$\mathbf{c}_i$ are the vertices of α, and $i = 1 \ldots 3$.

Proof: Similar to Lemma 4.1 □

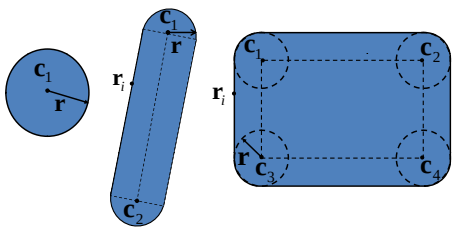


Fig. 3. **Swept sphere volume in 2D.** *From left to right: PSS, LSS and RSS.*

4.3 Acceleration Techniques

We first analyze the cost function for performing CCD on rigid models and then present two techniques to reduce the cost function.

4.3.1 Cost Analysis

The following formula expresses the cost of running our CCD algorithm:

$$T = 2T_{BVH} + N_{CA} \times T_{CA}, \tag{7}$$

where T_{BVH} is the cost of constructing a BVH for a polygonal model, N_{CA} is the total number of CA iterations for all bounding volume pairs to which CA is applied, and T_{CA} is the cost of evaluating the CA equation (Eq. 2). In our case, T_{BVH} and T_{CA} are constants, and so we need to reduce N_{CA}. We propose two schemes to achieve the goal: controlled conservative advancement (C^2A) and hierarchical CA (HCA).

4.3.2 Controlled Conservative Advancement (C^2A)

As explained in Sec. 3.2, we apply CA operations to the front nodes of the BVH that are computed during the closet distance query. If $N_{BV,i}$ is the number of front nodes in the ith iteration, and N_τ is the total number of CA iterations required to find τ, then N_{CA} can be expressed as follows:

$$N_{CA} = \sum_{i=1}^{N_\tau} N_{BV,i}. \tag{8}$$

Balancing N_τ and $N_{BV,i}$: In order to reduce $N_{BV,i}$, we control the depth of the front nodes by terminating the BVH traversal early during the closest-distance query, as shown in Fig. 4. Early termination only yields an approximate distance, although it is typically smaller than the actual closest distance. Therefore, the advancement time-step Δt_i from Eq. 2 is also looser. In our experiments, we have observed that when $i < j$, then $\Delta t_i \gg \Delta t_j$, especially when $i = 1, 2$ (i.e. during the first few iterations of CA), and a small value of $d(\mathcal{A}(t), \mathcal{B})$ yields a useful value of Δt_i. However, decreasing $N_{BV,i}$ for some i's may result in more CA iterations, making it hard to reduce the number of iterations N_τ and also $N_{BV,i}$. We need to find the balance between N_τ and $N_{BV,i}$ that minimizes T. Thus, to prevent an excessive number of iterations, for each iteration we check whether the approximate distance is smaller than some threshold value, or N_τ is larger than another threshold. In either of these cases, we abandon early termination and traverse all the way to the leaf nodes to compute the closest distance.

There are various ways of triggering early termination during the closest-distance query. Our method is to provide a spurious small distance value (instead of a large value or a stored distance from the previous computation) when the recursive function is initially called. The recursive distance query then terminates early, because it cannot decrease the spurious distance. Finally, when the entire recursive traversal is complete, we collect the front nodes where the recursion stops and use them for the CA computations.

The pseudocodes of our CCD algorithms for rigid models are given as Algorithms 1 and 2. In our implementation, we use $w = 0.3 \sim 0.5$ to control advancement during the first few iterations, but it is reset to 1 toward the end of iterations.

Algorithm 1 $\mathbf{C^2A}$: Controlled Conservative Advancement
Input: BVH root nodes $BVH_{\mathcal{A}}.root$, $BVH_{\mathcal{B}}.root$
Output: Time of contact τ
{Initially $d = \infty$, $\tau = 0.0$, and $w < 1.0$ }

while $d > \varepsilon$ **do**
  if the number of CA iterations $> R_{max}$ or $d < D_{threshold}$ **then**
    $w = 1$;
  end if
  $\tau += CA_{BVH}(BVH_{\mathcal{A}}.root, BVH_{\mathcal{B}}.root, d, 1.0, w)$
end while
return τ

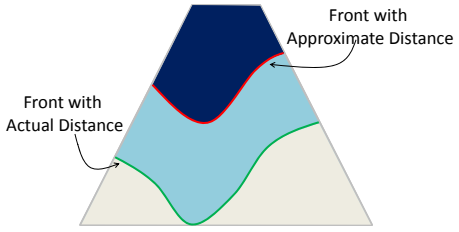


Fig. 4. **Controlling the depth of the front nodes.** *Front nodes obtained by terminating the recursion early during the closest-point query with approximate (red) and exact (green) distance values.*

Early Rejection: We use a simple early-rejection scheme to further improve the performance of CCD. During the CA iterations, if we find a node n of bounding volume pairs whose advancement time-step Δt_i is greater than some upper value Λ_i, we prune away the node and its children nodes. We initially set $\Lambda_1 = 1$, but at the kth iteration we set $\Lambda_k = 1 - \sum_{i=1}^{k-1} \Delta t_i$. When $\sum \Delta t_i > 1$, which implies that the node n does not collide during $[0, 1]$, and thus we can prune it away.

4.3.3 Hierarchical Conservative Advancement (HCA)

The C^2A algorithm requires a weight w, which controls the rate of advancement. It is difficult to deterministically choose w because the optimal value is likely to depend on the particular scenarios. Therefore we developed a new advancement algorithm, called hierarchical conservative advancement (HCA), which does not require a predefined weight, although its performance on rigid objects is similar to that of the C^2A algorithm with an optimal weight.

In general, the ToC of two rigid triangle-soup models can be easily obtained by taking the minimum

Algorithm 2 CA_{BVH}: Conservative Advancement for BVH
Input: BVH nodes $n_{\mathcal{A}}, n_{\mathcal{B}}$, current closest distance d_{cur}, current advancement step Δt_{cur}, CA controlling variable w
Output: Updated d_{cur} and Δt_{cur}

if $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ are leaf nodes **then**
  $d = \mathbf{Distance}(n_{\mathcal{A}}, n_{\mathcal{B}})$; {Using the PQP library}
  if$(d < d_{cur})$ $d_{cur} = d$;
  $\Delta t = \mathbf{CalculateCAStep}(d, n_{\mathcal{A}}, n_{\mathcal{B}})$; {Using Eq. 2}
  if$(\Delta t < \Delta t_{cur})$ $\Delta t_{cur} = dt$;
  return $d_{cur}, \Delta t_{cur}$
end if
if $n_{\mathcal{A}}$ is not a leaf node **then**
  $A = n_{\mathcal{A}}.leftchild$; $B = n_{\mathcal{A}}.rightchild$; $C = D = n_{\mathcal{B}}$;
else
  $A = B = n_{\mathcal{A}}$; $C = n_{\mathcal{B}}.leftchild$; $D = n_{\mathcal{B}}.rightchild$;
end if
$d_1 = \mathbf{Distance}(A, C)$; $d_2 = \mathbf{Distance}(B, D)$;
$\Delta t_1 = \mathbf{CalculateCAStep}(d_1, A, C)$;
$\Delta t_2 = \mathbf{CalculateCAStep}(d_2, B, D)$;
if $d_2 < d_1$ **then**
  if $d_2 < w d_{cur}$ **then**
    $CA_{BVH}(B, D, d_{cur}, \Delta t_{cur}, w)$;
  else
    if$(\Delta t_2 < \Delta t_{cur})$ $\Delta t_{cur} = \Delta t_2$;
  end if
  if $d_1 < w d_{cur}$ **then**
    $CA_{BVH}(A, C, d_{cur}, \Delta t_{cur}, w)$;
  else
    if$(\Delta t_1 < \Delta t_{cur})$ $\Delta t_{cur} = \Delta t_1$;
  end if
else
  if $d_1 < w d_{cur}$ **then**
    $CA_{BVH}(A, C, d_{cur}, \Delta t_{cur}, w)$;
  else
    if$(\Delta t_1 < \Delta t_{cur})$ $\Delta t_{cur} = \Delta t_1$;
  end if
  if $d_2 < w d_{cur}$ **then**
    $CA_{BVH}(B, D, d_{cur}, \Delta t_{cur}, w)$;
  else
    if$(\Delta t_2 < \Delta t_{cur})$ $\Delta t_{cur} = \Delta t_2$;
  end if
end if

ToC for every possible mutual pair of triangles. But since the number of such pairs can be huge, culling is necessary.

We can cull BVH nodes and their descendants when the minimum ToC of all their enclosed triangle pairs is greater than the current minimum ToC τ_{cur} for the entire body, because these pairs can not affect the ToC of the entire models. But since we traverse the BVH

from top to bottom, the minimum ToC of all the triangle pairs enclosed to $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ ($\tau^*(n_{\mathcal{A}}, n_{\mathcal{B}})$) cannot be determined when the node pair are visited, but only when their leaf nodes are reached. Therefore we substitute a lower bound $\underline{\tau}^*(n_{\mathcal{A}}, n_{\mathcal{B}})$ for $\tau^*(n_{\mathcal{A}}, n_{\mathcal{B}})$, and the new culling condition becomes $\underline{\tau}^*(n_{\mathcal{A}}, n_{\mathcal{B}}) \geq \tau_{cur}$, since $\tau^*(n_{\mathcal{A}}, n_{\mathcal{B}}) \geq \underline{\tau}^*(n_{\mathcal{A}}, n_{\mathcal{B}}) \geq \tau_{cur}$. Moreover, the ToC of a node pair $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$, $\tau(n_{\mathcal{A}}, n_{\mathcal{B}})$ provides the required lower bound on $\tau^*(n_{\mathcal{A}}, n_{\mathcal{B}})$ ($\tau^*(n_{\mathcal{A}}, n_{\mathcal{B}}) \geq \tau(n_{\mathcal{A}}, n_{\mathcal{B}})$), since the ToC of leaf nodes (triangles) must be greater than that of their ancestor. To get the tighter lower bound, $\underline{\tau}^*(n_{\mathcal{A}}, n_{\mathcal{B}})$ is set to the maximum value of the ToC between the node pair $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$, and between their ancestors. Note that we need the maximum operator here since our BVH does not ensure that ancestor BVs bound their descendent BVs (i.e. wrapped hierarchy) such that the ToC of ancestor nodes may have a small value than those of their descendents.

The pseudocode of our HCA algorithm is shown as Algorithm 3. The function $\mathbf{ToC}(n_{\mathcal{A}}, n_{\mathcal{B}}, t_{min})$ calculates the ToC between the nodes $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ ($\tau(n_{\mathcal{A}}, n_{\mathcal{B}})$), where t_{min} is a lower bound on $\tau(n_{\mathcal{A}}, n_{\mathcal{B}})$. Since each BVH node is convex, its motion bound can be calculated as in Eq. 5, and the advancement Δt_i of CA can be computed using Eq. 2. The ToC of two nodes can then be calculated by repeating the CA algorithm until the distance between them becomes less than a pre-defined threshold. Moreover, since t_{min} is a lower bound, we can speed up the ToC computation by stating CA at t_{min}, so that $\tau = t_{min} + \sum \Delta t_i$.

The function $\underline{\mathbf{ToC}}(n_{\mathcal{A}}, n_{\mathcal{B}}, t_{min})$, which is equivalent to $\underline{\tau}^*(n_{\mathcal{A}}, n_{\mathcal{B}})$, calculates a lower bound on ToC for all the triangle pairs enclosed by the BV node pair $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$, as follows:

$$\underline{\mathbf{ToC}}(n_{\mathcal{A}}, n_{\mathcal{B}}, t_{min}) = \max\left(t_{min}, \tau(n_{\mathcal{A}}, n_{\mathcal{B}})\right), \qquad (9)$$

where t_{min} is the maximum ToC of all the ancestors of node pair $(n_{\mathcal{A}}, n_{\mathcal{B}})$.

Algorithm 3 HCA: Hierarchical Conservative Advancement

Input: BVH nodes $n_{\mathcal{A}}, n_{\mathcal{B}}$, current minimum ToC τ_{cur}, lower bound on ToC for the triangle pairs bounded by $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ t_{min}

Output: Time of contact τ

{Initial call: HCA($BVH_{\mathcal{A}}.root$, $BVH_{\mathcal{B}}.root$, 1.0, 0)}

if $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ are leaf nodes **then**
  $t = \mathbf{ToC}(n_{\mathcal{A}}, n_{\mathcal{B}}, t_{min})$;
  if$(\tau_{cur} < t)$ $\tau_{cur} = t$;
  return τ_{cur}
end if
if $n_{\mathcal{A}}$ is not a leaf node **then**
  $A = n_{\mathcal{A}}.leftchild$; $B = n_{\mathcal{A}}.rightchild$; $C = D = n_{\mathcal{B}}$;
else
  $A = B = n_{\mathcal{A}}$; $C = n_{\mathcal{B}}.leftchild$; $D = n_{\mathcal{B}}.rightchild$;
end if
$t_1 = \underline{\mathbf{ToC}}(A, C, t_{min})$;
$t_2 = \underline{\mathbf{ToC}}(B, D, t_{min})$;
if $t_2 < t_1$ **then**
  if $t_2 < \tau_{cur}$ **then**
    HCA(B, D, τ_{cur}, t_2);
  end if
  if $t_1 < \tau_{cur}$ **then**
    HCA(A, C, τ_{cur}, t_1);
  end if
else
  if $t_1 < \tau_{cur}$ **then**
    HCA(A, C, τ_{cur}, t_1);
  end if
  if $t_2 < \tau_{cur}$ **then**
    HCA(B, D, τ_{cur}, t_2);
  end if
end if
return 1.0

4.3.4 Comparisons between C^{2}A and HCA

For HCA , the term N_{CA} in Eq. 7 can be replaced by:

$$N_{CA} = \sum_{k=1}^{N_{BV}} N_{\tau,k}, \qquad (10)$$

where N_{BV} is the total number of BV pairs to which CA is applied and $N_{\tau,k}$ is the number of CA iterations for the kth node pair. We can see from Eqs. 8 and 10 that C^{2}A only needs to perform CA iteration once for each node pair during BVH traversal, but needs to iteratively do BVH traversal; conversely, HCA performs the BVH traversal once and repeats CA iterations for each node pair. In C^{2}A, the variable w controls the number of times N_{CA} that CA must be performed, although the number of runs of CA is fixed in HCA. For this reason, with C^{2}A method, one needs to choose a proper value for the w parameter to get the optimal performance, even though the preset value of $0.3 \sim 0.5$ works nicely in our experiments. The HCA method does not require tuning any parameter, but the number of configuration update can be higher than C^{2}A. Thus, if the cost of configuration update is costly, for instance, when a screw motion is used for underlying interpolating motion as opposed to the linear one, C^{2}A may show better performance than HCA.

We need to determine the configuration of each BV node pair before running CA. But C^{2}A can compute the configuration directly from the configuration of the object, and CA iteration is only run once for each node pair during BVH traversal; thus the number of configuration updates is the same as the number of iterations (i.e. N_τ in Eq.8). Since HCA runs CA iterations on each node pair, the number of configu-

ration updates is N_{CA}. Thus C^{2}A performs far fewer configuration updates than HCA, which can impact its performance.

5 CCD FOR ARTICULATED MODELS

Our CCD algorithm for articulated models is based on the CATCH algorithm [9]. However, CATCH cannot handle articulated models made up of arbitrary polygonal models, while ours, using the technique presented in Sec. 4, can handle these models. We will first look at the motion bound computation, which is essential for CA-based CCD algorithm, and then explain how to improve the performance of the CCD algorithm for articulated models.

5.1 Notations

We will use the following notation to describe articulated models. An articulated model $\mathcal{A}$ is made up of m links $\mathcal{A}_i, i = 0, ..., m-1$. We assume that $\mathcal{A}$ does not contain any kinematic loop, and that $\mathcal{A}_{i-1}$ is the parent of $\mathcal{A}_i$. This convention is used just for notation. Our algorithm is applicable to any articulated model with no loops. $\{i\}$ denotes the reference frame of $\mathcal{A}_i$. $\{0\}$ is the world reference frame. Both the superscript and subscript in front of a symbol denote frame numbers; for instance, ${}^{j}_{i}\mathbf{R}(t) \in SO(3)$ is the orientation matrix of frame $\{i\}$ relative to frame $\{j\}$ at time $t, t \in [0,1]$. The subscript of a symbol (e.g. i) denotes a variable defined for $\mathcal{A}_i$; for instance, ${}^{j}\mathbf{v}_i$ and ${}^{j}\omega_i$ are respectively the linear and rotational velocities of $\mathcal{A}_i$ with respect to frame $\{j\}$.

5.2 Motion Bound

The motion bound μ on $\mathcal{A}_i$ at time t is:

$$\mu = \max_{\mathbf{p}\in\mathcal{A}_i, t\in[0,1]} \dot{\mathbf{p}}(t) \cdot \mathbf{n}, \tag{11}$$

$\dot{\mathbf{p}}(t)$ can be expressed as follows [9]:

$$\dot{\mathbf{p}}(t) = \sum_{j=1}^{i} \left(\begin{array}{l} {}^{0}_{j-1}\mathbf{R}(t)^{j-1}\,\mathbf{v}_i \\ \quad +{}^{0}_{j-1}\mathbf{R}(t)^{j-1}\,\omega_j \times \left(\sum_{k=j}^{i} {}^{0}_{k-1}\mathbf{R}(t)^{k-1}\,\mathbf{L}_k \right) \end{array} \right). \tag{12}$$

μ can be expressed as Eq. 14. While the motion bound in CATCH takes no account of the projection direction except for the root link, our bound (Eq. 14) is obtained by projecting the motion onto the direction of minimum distance $\mathbf{n}$ for every link. Thus our motion bound is tighter. In Eq. 14, ${}^{k-1}\mathbf{L}_k$ is the displacement vector from the origin of $\mathcal{A}_k$ to the origin of $\mathcal{A}_{k-1}$, and $\|{}^{k-1}\mathbf{L}_k\|$ can be calculated as [9]. An upper bound of $\|\mathbf{n} \times {}^{0}_{j-1}\mathbf{R}(t)^{j-1}\,\omega_j\|$ can be obtained using Taylor models, but their computation cost can be high. To calculate the upper bound more rapidly, we discretize the direction vector

$$\mathbf{n} = \begin{bmatrix} \sin\varphi\cos\theta \\ \sin\varphi\sin\theta \\ \cos\varphi \end{bmatrix}, \tag{13}$$

and precompute a lookup table for the bound. In more detail, the Taylor model of ${}^{0}_{j-1}\mathbf{R}(t)^{j-1}\,\omega_j$ can be evaluated when the dynamic BV is built for $\mathcal{A}_j$. Then a Taylor model of $\mathbf{n}$ can be constructed. A lookup table is generated for an upper bound of $\|\mathbf{n} \times {}^{0}_{j-1}\mathbf{R}(t)^{j-1}\,\omega_j\|$ for each link over a grid of θ and φ intervals.

5.3 Acceleration Technique

A straightforward method to get the ToC for articulated models is to compute the individual ToCs, τ_i, for every potential collision link pair PCL_i, and then determine which is the earliest. However, this method can be very expensive, since the number of these pairs can be very high depending on the number of links in the body. Culling methods can be employed to decrease the number of these pairs, as well as the number of CA iterations required for each pair. We will now summarize the culling methods used in CATCH [9] and then explain how we improve on these techniques.

5.3.1 *Improved Temporal Culling*

There are two main culling methods used in CATCH: dynamic BVH culling and temporal culling. Dynamic BVH culling involves the construction of a BVH for each articulated model using Taylor models and then the elimination of pairs of link whose dynamic BVs are not colliding. Our algorithm uses this technique without modification to improve the performance of CCD. Temporal culling has two steps:

1) **Collision-time sorting**: a single iteration of CA is performed for each PCL_i, yielding $\underline{\tau}_i$, which is a lower bound on the ToC for that pair. The pairs are then sorted into ascending order of $\underline{\tau}_i$.
2) **Temporal culling**: For each pair PCL_i, the minimum ToC for the link pairs upon which CCD has been performed, $\overline{\tau}_i = \min_{j<i} \tau_j$ is used as an upper bound for the pair. Further iterations of CA are then performed for PCL_i. If the estimated ToC after kth iteration becomes greater than $\overline{\tau}$, no more iterations are performed on PCL_i, since this pair cannot contribute to the ToC τ of the entire bodies.

Since the bounding volumes we used to perform CA are not as tight as the convex hulls used by CATCH [9], the number of BV nodes that need to be traversed during CA iterations can be high. We reduce the number of nodes visited by the first step of temporal culling, i.e. the collision-time sort. We do

$$
\begin{aligned}
\mu &= \max_{\mathbf{p}\in\mathcal{A}_i, t\in[0,1]} \dot{\mathbf{p}}(t)\cdot\mathbf{n} \\
&= \max_{\mathbf{p}\in\mathcal{A}_i, t\in[0,1]} \sum_{j=1}^{i}\left({}^{0}_{j-1}\mathbf{R}\,(t)^{j-1}\,\mathbf{v}_j\cdot\mathbf{n} + {}^{0}_{j-1}\mathbf{R}\,(t)^{j-1}\,\omega_j \times \left(\sum_{k=j}^{i} {}^{0}_{k-1}\mathbf{R}\,(t)^{k-1}\,\mathbf{L}_k\right)\cdot\mathbf{n}\right) \\
&\le \max_{\mathbf{p}\in\mathcal{A}_i, t\in[0,1]} \left(\sum_{j=1}^{i} {}^{0}_{j-1}\mathbf{R}\,(t)^{j-1}\,\mathbf{v}_j\cdot\mathbf{n} + \sum_{j=1}^{i}\left(\mathbf{n}\times {}^{0}_{j-1}\mathbf{R}\,(t)^{j-1}\,\omega_j\right)\cdot\left(\sum_{k=j}^{i} {}^{0}_{k-1}\mathbf{R}\,(t)^{k-1}\,\mathbf{L}_k\right)\right) \\
&\le^{0} \mathbf{v}_1\cdot\mathbf{n} + \max_{\mathbf{p}\in\mathcal{A}_i, t\in[0,1]} \left(\sum_{j=2}^{i}\left\|{}^{j-1}\mathbf{v}_j\right\| + \sum_{j=1}^{i}\left(\mathbf{n}\times {}^{0}_{j-1}\mathbf{R}\,(t)^{j-1}\,\omega_j\right)\cdot\left(\sum_{k=j}^{i} {}^{0}_{k-1}\mathbf{R}\,(t)^{k-1}\,\mathbf{L}_k\right)\right) \\
&\le^{0} \mathbf{v}_1\cdot\mathbf{n} + \sum_{j=2}^{i}\left\|{}^{j-1}\mathbf{v}_j\right\| + \max_{\mathbf{p}\in\mathcal{A}_i, t\in[0,1]} \left(\sum_{j=1}^{i}\left(\left\|\mathbf{n}\times {}^{0}_{j-1}\mathbf{R}\,(t)^{j-1}\,\omega_j\right\|\left(\sum_{k=j}^{i}\left\|{}^{k-1}\mathbf{L}_k\right\|\right)\right)\right)
\end{aligned}
\tag{14}
$$

this by applying HCA to only a sub-hierarchy of BVH of each potential collision link pair. This yields a lower bound $\underline{\tau_i}$ on the ToC for each pair, and then all the PCLs are sorted into ascending order of $\underline{\tau_i}$. The detail of our two-step culling method is as follows:

1) Compute a lower and an upper bound for each PCL_i:
   a) Randomly pick a pair of triangles $\{Tri_{\mathcal{A}}, Tri_{\mathcal{B}}\}$ from PCL_i and compute their ToC, which becomes an upper bound $\overline{\tau_i}$ on the ToC for PCL_i. Alternatively, if the underlying simulation environment (e.g. physics-based animation) allows us to exploit motion coherence, we may reuse the triangle pair that realizes the ToC of PCL_i from the previous frame of the simulation.
   b) The HCA method is applied to a sub-hierarchy of the BVHs of PCL_i, namely those with a depth less than a pre-defined threshold value, so as to obtain a lower bound $\underline{\tau_i}$ on the ToC for PCL_i. If $\underline{\tau_i}$ is greater than the minimum of all the potential collision link pairs' upper bound on ToC, i.e. $\underline{\tau_i} > \min_{\forall PCL_j\in\mathbf{L}} \overline{\tau_j}$, then PCL_i can be culled, since in this case $\tau_i \ge \underline{\tau_i} > \min_{\forall PCL_j\in\mathbf{L}} \overline{\tau_j} \ge \tau$ (which means that τ_i, the ToC for PCL_i is later than the ToC τ for entire articulated models).
2) In the second step, we sort all the remaining potential collision link pairs into ascending order using $\underline{\tau_i}$, since a pair with a smaller $\underline{\tau_i}$ has a higher probability to realize the final ToC τ, and further CA iterations should be preferentially applied to such PCL_i. We compute an initial upper bound τ_{cur} on ToC for the entire articulated models as $\overline{\tau} = \min_{\forall PCL_i\in\mathbf{L}} \overline{\tau_i}$ and then attempt to reduce τ_{cur} until it converges on the ToC τ for the entire models by executing HCA for each PCL_i. However, if $\underline{\tau_i}$ is greater than τ_{cur}, we do not need to compute τ_i and PCL_i can be culled, since $\tau_i \ge \underline{\tau_i} > \tau_{cur} \ge \tau$.

The pseudocode of our CCD algorithm for articulated models using temporal culling is given as Alg. 4. The function $\mathbf{HCA}_{sub}$ is used to get a lower bound on the ToC by limiting the depth of traversal T_{depth} of BVHs. The computation of $\mathbf{HCA}_{sub}$ is a simple modification of **HCA** (Alg. 3), effected by changing the decision condition for a leaf node (line 1 in Alg. 3). In $\mathbf{HCA}_{sub}$, we decide whether a BV node is a leaf node by comparing its depth with a pre-defined threshold T_{depth}. $\mathbf{HCA}_{sub}$ does not visit BV nodes at a depth greater than T_{depth}. In the function **HCA** (line 26 in Alg. 4), we use τ_{cur} as an initial upper bound on τ_i. If the lower bound on the ToC for a BV pair is found to be greater than τ_{cur} during the traversal, then we can cull that pair, as well as all its children, since their ToCs cannot reduce the ToC of the full articulated models.

5.3.2 Precomputed Transformations

As we discussed in Sec. 4.3.4, the performance of HCA may be compromised by a complicated configuration computation. For articulated models, computing the configuration of a BV node is complicated (i.e. Eq. 12). To reduce the number of configuration updates, we pre-compute the configuration $\mathbf{q}(t)$ for each link with a given time resolution $T_t < 1$, $\{\mathbf{q}(t)|t = T_t \times i, i = 0, ..., \lfloor\frac{1}{T_t}\rfloor - 1\}$, allowing us to obtain the configurations of each BV at the time intervals. We use these precomputed configurations to compute the ToC for non-leaf node pairs approximately. The ToC after the ith advancement is approximated by $\tau'(n_{\mathcal{A}}, n_{\mathcal{B}}) = T_t \times \left(\left\lfloor\frac{t_{\min}}{T_t}\right\rfloor + \sum_i \left\lfloor\frac{\Delta t_i}{T_t}\right\rfloor\right)$, where t_{min} is a known lower bound on $\tau(n_{\mathcal{A}}, n_{\mathcal{B}})$. Then the configurations of $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ at τ' are precomputed. The CA will be performed iteratively until the distance between $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ becomes less than a threshold, or Δt_i becomes less than the time resolution T_t. Since we use a floor operation to compute τ', $\tau'(n_{\mathcal{A}}, n_{\mathcal{B}})$ is a lower bound on $\tau(n_{\mathcal{A}}, n_{\mathcal{B}})$. The availability of τ' allows us to omit ToC computations for non-leaf node pairs ($\tau(n_{\mathcal{A}}, n_{\mathcal{B}})$ in Eq. 9), because τ' is also a lower bound on the ToC of all their constituent triangle pairs (i.e. $\underline{\tau}^*(n_{\mathcal{A}}, n_{\mathcal{B}})$). In **HCA**, since $\underline{\tau}^*(n_{\mathcal{A}}, n_{\mathcal{B}})$ (lines 11, and 12 in Alg. 3)

are used only for culling purposes, the substitution will not affect the accuracy of the ToC for the entire model.

Algorithm 4 CCD with Temporal Culling:
Input: a list **L** including all PCL_i
Output: ToC τ for the whole articulated model

{**First step:** upper and lower bounds computation}
$\underline{\tau} = \overline{\tau} = 1.0$;
for ($PCL_i \in$ **L**) **do**
  $\{Tri_{\mathcal{A}}, Tri_{\mathcal{B}}\} \in PCL_i$;
  $\overline{\tau_i} = \mathbf{ToC}(Tri_{\mathcal{A}}, Tri_{\mathcal{B}}, 0.0)$;
  if $\overline{\tau_i} = 0.0$ **then**
    return $\tau = 0.0$;
  end if
  if $\overline{\tau_i} < \overline{\tau}$ **then**
    $\overline{\tau} = \overline{\tau_i}$;
  end if
  $\{\mathcal{A}, \mathcal{B}\} = PCL_i$;
  $\underline{\tau_i} = \mathbf{HCA}_{sub}(BVH_{\mathcal{A}}.root, BVH_{\mathcal{B}}.root, \overline{\tau}, 0.0, T_{depth})$;
end for
for ($PCL_i \in$ **L**) **do**
  if $\underline{\tau_i} > \overline{\tau}$ **then**
    $\mathbf{L}.remove(PCL_i)$;
  end if
end for
{**Second step:** Sort and cull}
Sort L in ascending order of $\underline{\tau_i}$;
$\tau_{cur} = \overline{\tau}$;
for ($PCL_i \in$ **L**) **do**
  if $\underline{\tau_i} < \tau_{cur}$ **then**
    $\{\mathcal{A}, \mathcal{B}\} = PCL_i$;
    $\tau_i = \mathbf{HCA}(BVH_{\mathcal{A}}.root, BVH_{\mathcal{B}}.root, \tau_{cur}, \underline{\tau_i})$;
    if $\tau_i < \tau_{cur}$ **then**
      $\tau_{cur} = \tau_i$;
    end if
  end if
end for
return τ_{cur}

6 Results and Discussions

We now describe the implementation of our CCD algorithms and present the results from various benchmarks, including both rigid and articulated models.

We implemented our CCD algorithms using C++ on a Windows 7 PC, equipped with an Intel Core Q9450 2.66GHz CPU and 2.75Gb of memory. A modified version of the public-domain library PQP was used for to construct SSV hierarchies and distance queries. We implemented our CCD algorithm for articulated model based on the public-domain library CATCH (http://graphics.ewha.ac.kr/CATCH/).

6.1 Rigid Models

We benchmarked our algorithms using the models shown in Fig. 5. These contain from 1k to 105k triangles. Performance for these benchmarks is shown in Table 1.

We used the same benchmarking setup as [8] to measure the performance of our algorithms: one of the models moves from a random configuration $\mathbf{q}_0$ toward another random configuration $\mathbf{q}_1$ against another model fixed in space (see Fig. 7). We detect the first time of contact between the moving and fixed objects by repeating each test 200 times and averaged the results. We used three benchmark polygon-soup models; Club vs. Club, Gear vs. Gear and Hammer vs. CAD workpiece. For Club vs. Club, all the motion between $\mathbf{q}_0$ and $\mathbf{q}_1$ yielded collisions (i.e. $\tau < 1$); in Gear vs. Gear, two thirds of the trials resulted in collisions. Hammer vs. CAD workpiece produced configurations similar to those resulting from the Gear vs. Gear scenario. As shown in Table 1, controlled advancement improves the performance of C^{2}A by a factor of between 1 and 28, and the HCA algorithm shows a similar improvement. Fig. 6 shows the number of front nodes (N_{BV}) in each trial. The use of controlled advancement in the C^{2}A algorithm achieves a particulary good reduction in N_{CA}, by 82.3% on average.

We also compare the performance of our C^{2}A algorithm with that of FAST [8] using the same benchmarks: Bunny vs. Bunny, Bunny Dynamics, and Torusknot vs. Torusknot (as shown in 7). C^{2}A does not take advantage of the connectivity information in these models, though FAST exploits it. Despite this, C^{2}A achieves similar performance to FAST. This also means that our algorithm is an order of magnitude faster than competing CCD algorithms for polygon-soup models such as [7], since FAST shows a similar speedup when manifold models are used. Finally, our software implementation is available for download at http://graphics.ewha.ac.kr/C2A.

6.2 Articulated Models

We compared the performance of our algorithm with that of CATCH [9] on three benchmarks: an exercising mannequin, a walking mannequin on a chessboard, and a grasping hand, as shown in Figs. 8 and 9. The first two benchmarks were also used in [9]. The mannequin in the exercising and walking benchmarks is composed of 15 links and 20k triangles, and the chessmen in the walking benchmark are made up of 101k triangles. In the above two scenarios, the CCD algorithm is used to detect the first time of contact between the mannequin and the chessmen or the links of the mannequin. In the grasping benchmark, a hand model moves towards a sphere from different directions and then closes its fingers with different velocities to grasp it. The hand model consists of 21

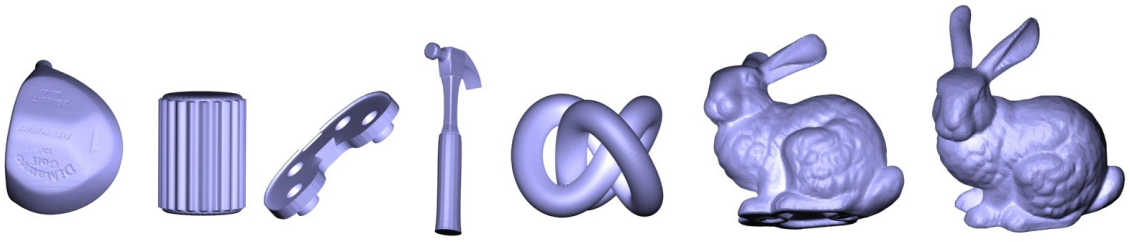


Fig. 5. **Benchmark Models.** *From left to right (with triangle counts): Gear (25.6k), Club (10.5k), CAD workpiece (2.6k), Hammer (1.7k), Torusknot (34.6k), Bunny1 (70k), Bunny2 (26k).*

Benchmark	Number of triangles	C^2A without controlled advancement (ms)	C^2A			HCA (ms)	FAST (ms)
			Colliding confgs.(ms)	Collision-free confgs.(ms)	Average (ms)		
Club vs. Club	104.8k (each)	14.97	3.6	–	3.6	3.4	–
Gear vs. Gear	25.6k (each)	55.49	2.96	0.0048	1.98	1.68	–
Hammer vs. CAD piece	1.7k, 2.6k	7.68	2.82	0.0052	1.89	1.86	–
Bunny vs. Bunny	69.7k (each)	8.64	4.11	0.048	2.77	2.77	4.01
Bunny Dynamics	26.4k (each)	0.41	5.54	0.11	0.22	0.15	0.31
Torusknot vs. Torusknot	34.6k (each)	6.8	2.81	0.41	2.01	2.68	1.96

TABLE 1
Benchmarks for rigid models.

links and 8k triangles, and the sphere model consists of 2k triangles. Our CCD algorithm is used to find the first contact between the hand and sphere during the approaching motion of the hand; it is then used again to find the ToC between the fingers and the sphere, as well as between the fingers themselves as they close.

We also plugged our algorithm into the sampling-based motion planner MPK library [30], where it is used to check whether a local path between sampled configurations is collision-free. We tested this combination on a scenario in which a Puma robot moves around a Beetle car, using models from the MPK library (shown in Fig. 10). The Puma is composed of 8 links and 1k triangles, and the Beetle is modeled with 4k triangles.

The performance statistics for these benchmarks are shown in Table 2. The performance of our algorithm is better than CATCH for exercising, walking and grasping benchmarks, despite the fact our algorithm does not require models to be manifolds, which is the limitation that CATCH imposes in order to speed up its performance. Moreover, our algorithm is an order of magnitude improvement of fastest algorithms for articulated models composed of polygon-soups such as [25], since CATCH shows a similar improvement under the manifold assumption. We cannot compare our algorithm to CATCH for the Puma/Beetle benchmark, since they are polygon-soup models (*i.e.*, non-manifold).

6.3 Parameter Tuning

In our CCD algorithms, there are different parameters used for both rigid and articulated models. In the section, we discuss and provide empirical solutions

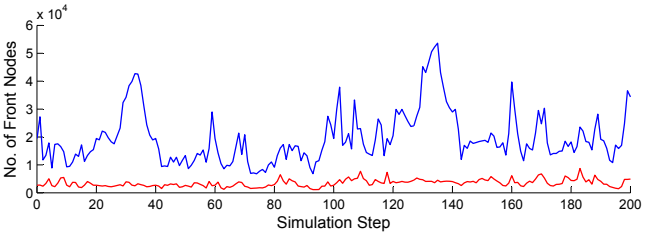


Fig. 6. **Number of front nodes.** *Red and blue lines show the number of front nodes, with and without controlled conservative advancement, respectively, for the Club vs. Club benchmark.*

Benchmark	Our Algorithm (ms)			CATCH (ms)
	Colliding configs.	Collision-free configs.	Average	
Exercising	1.62	0.77	0.88	0.90
Walking	1.08	0.94	0.99	1.03
Grasping	9.79	-	9.79	10.23
Puma	0.87	0.35	0.38	-

TABLE 2
Benchmarks for articulated models.

how to tune these parameters to get the best CCD performance.

For rigid models, we use a distance threshold parameter to stop the CA iterations, when the distance between the models becomes less than some threshold. This threshold exists for all CA-based methods such as [1], [6], [8], [9], [11], [22], and it controls the accuracy of CCD results. In our experiments for both rigid and articulated models, we set this value to 0.001. In general, the higher the distance threshold is, the less time the CA iterations take. Thus, depend-

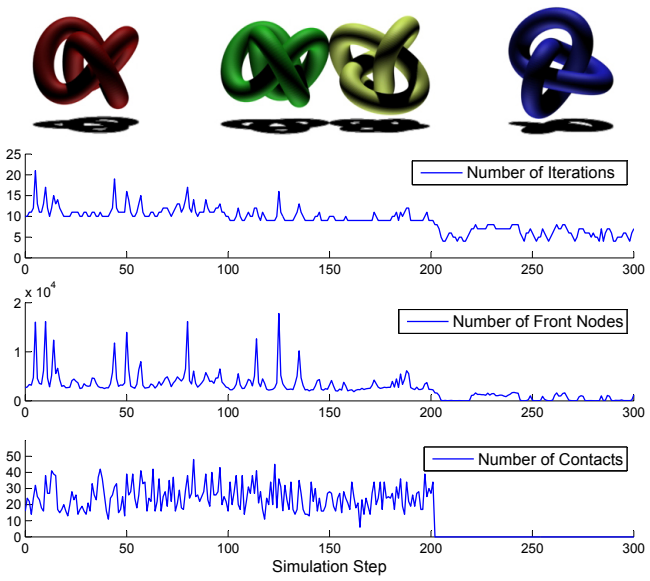
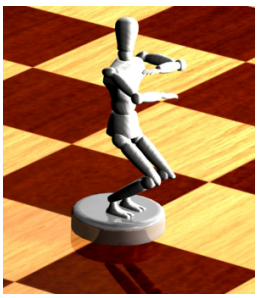


Fig. 7. **Torusknot vs Torusknot.** *Top: the red, blue, yellow, and green torusknots represent the initial* $\mathbf{q}_0$ *and final* $\mathbf{q}_1$ *configurations of* $\mathcal{A}$*, the configuration of* $\mathcal{B}$*, and the configuration of* $\mathcal{A}$ *at* τ*, respectively. From the second row to bottom: the number of CA iterations* N_τ*, the number of front nodes* N_{BV}*, and the number of contacts for the benchmark.*

(a) Walking mannequin



(b) Exercising mannequin

Fig. 8. **Mannequin benchmarks.** *The mannequin in both benchmarks consists of 15 links and 20k triangles. The chessmen are composed of 101k triangles.*

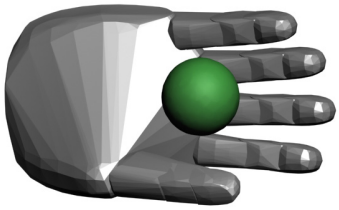


Fig. 9. **Grasping.** *The hand model is composed of 21 links and 8k triangles, and the sphere of 2k triangles.*

ing on application demands, the distance threshold should be determined. We also use a maximum iteration number to limit the CA iterations both for rigid and articulated models. The threshold is used to avoid the scenario when the number of CA iterations is too high. For all our benchmarks, the threshold is set to 50.

For articulated models, two more thresholding parameters, T_{depth} and T_t are used. T_{depth} is used in HCA_{sub} (Ln 13 in Algorithm 4) to limit the traversal depth in bounding volume hierarchy (BVH), and is used to get a lower bound of time of contact (ToC) for each potential collision link (PCL). With a higher value of T_{depth}, the lower bound of ToC becomes closer to the exact ToC and we can compute the ToC more quickly in the second step in Algorithm 4. However, the higher value of T_{depth} can make the lower bound computation itself slower, since we need to traverse the BVH more deeply. Thus, there is a trade-off to set T_{depth}. In all of our benchmarks, we set T_{depth} as $\frac{BVH_{depth}}{3}$, where BVH_{depth} is the maximum depth for the whole BVH. Another parameter is the time resolution T_t, explained in Sec. 5.3.2, at which the link configurations are pre-computed. A small value of T_t will increase the cost for the transformations, while a large one will decrease the culling efficiency, resulting in the CCD performance degradation. In our benchmarks, we set $T_t = 0.01$.

7 Conclusions

We have presented two CCD algorithms, C^{2}A and HCA, for rigid polygon-soup models. These algorithms are based on tight motion-bound calculations for swept-sphere volumes (SSV) and on the adaptive use of conservative advancement to control hierarchical traversal of the BVH. Different culling techniques are used to improve the performance of each algorithm. We also extended the CCD algorithm to articulated models.

One of the limitations of all these algorithms is that distance calculation based on SSV is still a bottleneck. Another limitation is that there are no theoretical guarantees on the bound of the number of CA iterations. We plan to apply our CCD algorithms to a range of graphical applications, including 6-DoF haptic and physic dynamics, and extending them to deformable models.

Acknowledgements

This research was supported in part by NRF in Korea (No.2012R1A2A2A01046246, No.2012R1A2A2A06047007). Dinesh Manocha was

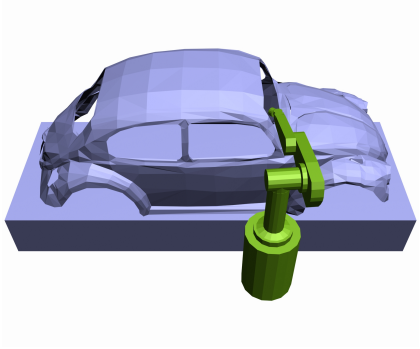


Fig. 10. **Puma and Beetle.** *The Puma robot is composed of 8 links and 1k triangles, and the Beetle model of 4k triangles.*

supported in part by ARO Contract W911NF-12-1-0430, NSF awards 100057, 1117127, 1305286 and Intel.

REFERENCES

[1] Min Tang, Young J. Kim, and D. Manocha, "C^{2}A: Controlled conservative advancement for continuous collision detection of polygonal models," in *Proc. of Int. Conf. on Robotics and Automation*, 2009.

[2] S. Redon, Young J. Kim, Ming C. Lin, Dinesh Manocha, and Jim Templeman, "Interactive and continuous collision detection for avatars in virtual environments," in *Proc. of IEEE Virtual Reality 2004*, Washington, DC, USA, 2004, VR '04, pp. 117–, IEEE Computer Society.

[3] F. Schwarzer, M. Saha, and J.-C. Latombe, "Exact collision checking of robot paths," in *Workshop on Algorithmic Foundations of Robotics*, Dec. 2002.

[4] S. Redon and M. Lin, "Practical local planning in the contact space," in *Proc. of Int. Conf. on Robotics and Automation*, 2005.

[5] L. Zhang and D. Manocha, "Constrained motion interpolation with distance constraints," in *Wkop. on Algorithmic Foundations of Robotics*, 2008.

[6] Min Tang, Young J. Kim, and D. Manocha, "CCQ: effcient local planning using connection collision query," in *Workshop on Algorithmic Foundations of Robotics*, 2011, vol. 68, pp. 229–247.

[7] S. Redon, A. Kheddar, and S. Coquillart, "Fast continuous collision detection between rigid bodies," *Proc. of Eurographics (Computer Graphics Forum)*, 2002.

[8] Xinyu Zhang, Minkyoung Lee, and Young J. Kim, "Interactive continuous collision detection for non-convex polyhedra," *The Visual Computer*, pp. 749–760, 2006.

[9] Xinyu Zhang, Stephane Redon, Minkyoung Lee, and Young J. Kim, "Continuous collision detection for articulated models using taylor models and temporal culling," *ACM Trans. on Graphics (Proc. of SIGGRAPH 2007)*, vol. 26, no. 3, pp. 15, 2007.

[10] M. C. Lin, *Efficient Collision Detection for Animation and Robotics*, Ph.D. thesis, University of California, Berkeley, CA, Dec. 1993.

[11] B. V. Mirtich, *Impulse-based Dynamic Simulation of Rigid Body Systems*, Ph.D. thesis, University of California, Berkeley, 1996.

[12] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," Tech. Rep. TR99-018, Department of Computer Science, University of North Carolina, 1999.

[13] J. F. Canny, "Collision detection for moving polyhedra," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 8, pp. 200–209, 1986.

[14] Y.-K. Choi, W. Wang, Y. Liu, and M.-S. Kim, "Continuous collision detection for elliptic disks," *IEEE Trans. on Robotics*, 2006.

[15] B. Kim and J. Rossignac, "Collision prediction for polyhedra under screw motions," in *ACM Conf. on Solid Modeling and Applications*, June 2003.

[16] S. Redon, A. Kheddar, and S.Coquillart, "An algebraic solution to the problem of collision detection for rigid polyhedral objects," 2000.

[17] K. Abdel-Malek, D. Blackmore, and K. Joy, "Swept volumes: Foundations, perspectives, and applications," *Int. J. of Shape Modeling*, vol. 12, no. 1, pp. 87–127, 2002.

[18] P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang, "Deformable free space tiling for kinetic collision detection," in *Wkop. on Algorithmic Foundations of Robotics*, 2001, pp. 83–96.

[19] D. Kim, L. Guibas, and S. Shin, "Fast collision detection among multiple moving spheres.," *IEEE Trans. Vis. Comput. Graph.*, vol. 4, no. 3, pp. 230–242, 1998.

[20] D. Kirkpatrick, J. Snoeyink, and Bettina Speckmann, "Kinetic collision detection for simple polygons," in *ACM Symposium on Computational Geometry*, 2000, pp. 322–330.

[21] G. van den Bergen, "Ray casting against general convex objects with application to continuous collision detection," *Journal of Graphics Tools*, 2004.

[22] Min Tang, Young J. Kim, and D. Manocha, "Continuous collision detection for non-rigid contact computations using local advancement," in *Proceedings of International Conference on Robotics and Automation*, 2010.

[23] Stefan Gottschalk, Ming Lin, and Dinesh Manocha, "OBB-Tree: A hierarchical structure for rapid interference detection," in *Proc. of SIGGRAPH 96*, 1996, pp. 171–180.

[24] Jia Pan, Sachin Chitta, and Dinesh Manocha, "FCL: A general purpose library for collision and proximity queries," in *Proc. of Int. Conf. on Robotics and Automation*, St. Paul, Minnesota, USA, May 2012.

[25] S. Redon, Young J. Kim, Ming C. Lin, and Dinesh Manocha, "Fast continuous collision detection for articulated models," in *Proc. of ACM Symposium on Solid Modeling and Applications*, 2004.

[26] Min Tang, Sean Curtis, Sung-Eui Yoon, and Dinesh Manocha, "Interactive continuous collision detection between deformable models using connectivity-based culling," in *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, New York, NY, USA, 2008, pp. 25–36, ACM.

[27] Changxi Zheng and Doug L. James, "Energy-based self-collision culling for arbitrary mesh deformations," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2012)*, vol. 31, no. 4, Aug. 2012.

[28] Min Tang, Dinesh Manocha, Sung-Eui Yoon, Peng Du, Jae-Pil Heo, and Ruofeng Tong, "VolCCD: Fast continuous collision culling between deforming volume meshes," vol. 30, pp. 111:1–111:15, May 2011.

[29] Jia Pan, Liangjun Zhang, and Dinesh Manocha, "Collision-free and curvature-continuous path smoothing in cluttered environments," in *Proc. of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[30] F. Schwarzer, M. Saha, and J.-C. Latombe, "Adaptive dynamic collision checking for single and multiple articulated robots in complex environments," *IEEE Trans.on Robotics*, vol. 21, no. 3, pp. 338–353, 2005.

Min Tang is a postdoctoral research fellow in the Department of Computer Science and Engineering at Ewha Womans University. She was a full-time lecturer (2006-2008) in the Department of Computer Science and Engineering at Hohai Universtiy, China. She obtained her BS and PhD degrees in Computer Science from Nanjing University of Science and Technology in 2001 and 2006, respectively. Her research interests include computer graphics, motion planning, and collision detection.

Dinesh Manocha is currently the Phi Delta Theta/Mason Distinguished Professor of Computer Science at the University of North Carolina at Chapel Hill. He received his Ph.D. in Computer Science at the University of California at Berkeley 1992. He has received Junior Faculty Award, Alfred P. Sloan Fellowship, NSF Career Award, Office of Naval Research Young Investigator Award, Honda Research Initiation Award, Hettleman Prize for Scholarly Achievement. Along with his students, Manocha has also received 12 best paper awards at the leading conferences on graphics, geometric modeling, visualization, multimedia and high-performance computing. He has published more than 340 papers and some of the software systems related to collision detection, GPU-based algorithms and geometric computing developed by his group have been downloaded by more than 100,000 users and are widely used in the industry. He has supervised 23 Ph.D. dissertations and is a fellow of ACM, AAAS, and IEEE. He received Distinguished Alumni Award from Indian Institute of Technology, Delhi.

Young J. Kim is an associate professor of computer science and engineering at Ewha Womans University. He received his PhD in computer science in 2000 from Purdue University. Before joining Ewha, he was a postdoctoral research fellow in the Department of Computer Science at the University of North Carolina at Chapel Hill. His research interests include interactive computer graphics, computer games, robotics, haptics, and geometric modeling. He has published more than 70 papers in leading conferences and journals in these fields. He also received the best paper awards at the ACM Solid Modeling Conference in 2003 and the International CAD Conference in 2008, and the best poster award at the Geometric Modeling and Processing conference in 2006. He was selected as best research faculty of Ewha in 2008, and received the outstanding research cases awards from Korean research foundation and Korean Minstry of Knowledge and Economy in 2008 and 2011.