C^2A: Controlled Conservative Advancement for Continuous Collision Detection of Polygonal Models

(http://graphics.ewha.ac.kr/C2A)

Min Tang, Young J. Kim and Dinesh Manocha

***Abstract*— We present a simple and fast algorithm to perform *continuous collision detection* between polygonal models undergoing rigid motion for interactive applications. Our approach can handle all triangulated models and makes no assumption about the underlying geometry and topology. The algorithm uses the notion of *conservative advancement (CA)*, originally developed for convex polytopes [1], [2]. We extend this formulation to general models using swept sphere volume hierarchy and present a compact formulation to compute the motion bounds along with a novel controlling scheme. We have implemented the algorithm and highlight its performance on various benchmarks. In practice, our algorithm can perform continuous collision queries in few milli-seconds on models composed of tens of thousands of triangles.**

I. Introduction

Collision detection and distance computation are important problems in robotics, computer graphics and physically-based modeling. In particular, reliable and fast collision detection algorithms are required for robot motion planning and dynamics simulation to enforce the non-penetration constraints. Collision detection has been extensively studied over the past two decades by many researchers in the field. Most of the earlier approach on collision detection focused on *static* scenes, where the queried objects are placed stationary in space. Recently, research has focused on *dynamic* scenes, where objects undergo motion. In a few dynamic scenarios, the entire motion trajectory of an object may be known in advance as a function of time. In most practical scenarios, the object's motion is not known a priori and only a few sampled locations are known. For example, in sampling-based robot motion planning, randomized planners generate collision-free configurations of a robot using sampling algorithms and use these samples to compute a continuous collision-free path from the initial to goal configurations.

At a broad level, the dynamic collision detection algorithms can be subdivided into two categories: *discrete* and *continuous*. Discrete algorithms check for collision only at sampled configurations using static collision detection algorithms. In consequence, they may miss collision between two successive configurations, also known as the *tunneling problem*. On the other hand, continuous collision detection (CCD) algorithms can resolve the tunneling problem by first computing a continuous, motion interpolation between successive configurations and use the interpolated motion to check for collisions. Thus, CCD never misses any potential collision between configurations. CCD algorithms are used in sampling-based motion planning to perform the local planning step [3], [4], [5], and in robot dynamics to find the first time of contact to apply responsive forces [6]. However, the major drawback of CCD algorithms are that they are typically slower than the discrete counterpart. Many well-known libraries for sample-based motion planning such as MSL[1] mostly use discrete collision checking for local planning. Most of the prior algorithms for continuous collision detection are limited to polyhedral or articulated models, which take into account the connectivity or topological information. These approaches can be rather slow on general models that are represented as polygon-soup models with no connectivity information.

Main Results: In this paper, we present a simple and fast algorithm that performs CCD for *polygon-soup* models undergoing rigid motion at highly interactive rates. We make no assumptions about the model geometry and topology as long as the model is tessellated. Our CCD algorithm uses an extension of the conservative advancement (CA) technique, which was initially proposed for convex polytopes[1], [2], as described in Sec.III. The CA formulation requires two functional components: distance computation and motion bound calculation. One can use any separation distance computation algorithm for polygonal models. In our case, we adopt the well-known algorithm based on swept sphere volume (SSV) available as part of PQP library [7]. We also present a novel, analytic method to efficiently calculate the motion bound of SSVs, as described in Sec.IV. Moreover, we accelerate the performance of our CCD algorithm by using controlled advancement techniques that are described in Sec.V. We have implemented our algorithm and highlight the performance on different benchmarking models with varying complexities, as shown in Sec.VI. In practice, our algorithm can perform CCD in a fraction of milli-seconds on models represented using tens of thousands of triangles. Furthermore, our new algorithm is faster than the prior specialized CCD algorithms for polyhedral models.

M. Tang and Y. J. Kim are with the Department of Computer Science and Engineering at Ewha Womans University in Seoul, Korea. {tangmin|kimy}@ewha.ac.kr

D. Manocha is with the Department of Computer Science at the University of North Carolina at Chapel Hill, U.S.A. dm@cs.unc.edu

[1] http://msl.cs.uiuc.edu/msl/

II. Previous Work

In this section, we give a brief survey of prior work on continuous collision detection and motion bound computations.

A. Continuous Collision Detection

Different types of CCD algorithms have been proposed in the literature. At a broad level, these algorithms can be classified into the following: algebraic equation solvers [8], [9], [10], [11], swept volume formulations [12], adaptive bisection approach [6], [3], kinetic data structures (KDS) approach [13], [14], [15], Minkowski sum-based formulations [16] and conservative advancement [1], [2], [17].

However, most of these approaches are unable to perform very fast CCD queries on general polygonal models. A few of these algorithms can handle polygon-soup models such as [11], [3]. Redon *et al.* [11] uses a continuous version of the *separating axis theorem* to extend the static OBB-tree algorithm [18] to CCD and demonstrates real-time performance on polygonal models. But the algorithm becomes overly conservative when there is a large rotational motion between two configurations. In practice, this algorithm is slower than Zhang *et al.*'s algorithm [17] for polyhedral models. More recently, Zhang *et al.* have extended their approach to articulated models [19], but each link in the model should be a polyhedron. Redon *et al.* [20] describe an extension of [11] to articulated models and their algorithm suffers from a similar problem. Schwarzer *et al.*'s algorithm [3] is based on a conservative condition to guarantee a collision-free motion between two configurations, but the condition may become overly conservative when an object slides over another object. Moreover, the motion bound is also quite conservative and is mainly designed for simple prismatic and revolute joints, or their combinations.

B. Motion Bound Calculation

Schwarzer *et al.* [3] propose a method to bound the motion trajectory of a moving robot with constant translational and rotational velocities. The upper bound on the motion trajectory is computed by taking the weighted sum of differences between all the configuration parameters along the motion trajectory. This bound is used for local planning in MPK motion planning library.

Redon *et al.* [21] bound the motion trajectory of linear swept spheres (LSS) by using interval arithmetic and the resulting bound has been used for dynamic collision checking between a moving avatar and the virtual environment. However, these methods do not take into account how close a robot and obstacles are displaced from each other (i.e. *undirected motion bound*) and the motion trajectory may not be fully utilized during the bound calculation.

Lin [1] and Mirtich [2] propose a ballistic motion as a motion trajectory for CCD and compute a *directed motion bound* along the closest direction between two convex polytopes by bounding the ballistic rotational velocity. Zhang *et al.* [17] use the extremal vertex query to find a directed motion bound for an object moving with constant translational and rotational velocities.

III. Overview

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

A. Preliminaries

Let $\mathfrak{A}$ and $\mathfrak{B}$ be two polygon-soup models in 3D, where $\mathfrak{A}$ is movable under rigid transformation $\mathbf{M}(t)$. Without loss of generality we assume that $\mathfrak{B}$ is fixed. Further, we assume that the initial and final configurations of $\mathfrak{A}$ are given as $\mathbf{q}_0$ and $\mathbf{q}_1$ at time $t = 0$ and $t = 1$, respectively. We also define $\mathfrak{A}(t) = \mathbf{M}(t)\mathfrak{A}$. The problem of CCD can be formulated as checking whether Eq. 1 has a feasible solution:

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

Furthermore, if Eq. 1 has a solution, we also compute the minimum value of t that satisfies this equation, known as the first time of contact, τ.

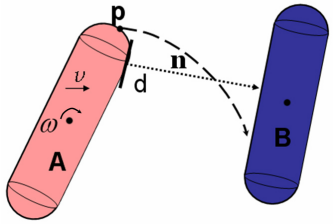

Fig. 1. **Conservative Advancement of LSS.**

CA (conservative advancement) is a simple technique that computes a lower bound of τ between two convex objects by repeatedly advancing $\mathfrak{A}$ by Δt_i toward $\mathfrak{B}$ while avoiding collisions [1], [2]. In this case, Δt_i is calculated based on a lower bound on the closest distance $d(\mathfrak{A}(t), \mathfrak{B})$ between $\mathfrak{A}(t)$ and $\mathfrak{B}$, and an upper bound μ on the motion of $\mathfrak{A}(t)$ projected onto $d(\mathfrak{A}(t), \mathfrak{B})$ per unit second (also shown in Fig. 1); i.e.

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

Then, the first time of contact τ can be obtained by repeatedly calculating the time-step Δt_i and summing it (i.e. $\tau = \sum \Delta t_i$) until $d(\mathfrak{A}(\tau), \mathfrak{B})$ becomes less than some user-specified threshold. Notice that the CA works only for convex objects. We extend this formulation to arbitrary polygon-soup models using swept sphere volume (SSV) hierarchies.

B. Our Approach

We assume that only the initial and final configurations of $\mathfrak{A}$ 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 $\mathfrak{A}$ a priori(*e.g.* [15]), we approximate

the motion with $\mathbf{M}(t)$ in a piecewise manner. A similar interpolating method was also used in [6] and [17]. [6] pointed out when the simulation time-step is small, the difference between the actual objects' motions and the interpolated paths is negligible.

As a preprocess, we compute a bounding volume hierarchy (BVH) for polygon soups, in particular the SSV hierarchy. SSV is a BV consisting of one of the three types, namely point swept sphere (PSS), line swept sphere (LSS) and rectangle swept sphere (RSS). These bounding volumes can contain a set 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 until the leaf-level SSV node contains only a single primitive. More details on the construction and its use for distance and proximity computations are available in [7].

At run-time, we apply CA to the nodes in the SSV hierarchy in a *selective* manner. As shown in Eq. 2, CA requires two components: closest distance and motion bound. Thus, we need to compute the closest distance between SSVs and its motion bound. The distance is obtained as a byproduct of the SSV algorithm; however, we still need to compute a tight upper bound on the motion, μ for SSV, which will be explained in Sec. IV.

A critical computation on the algorithm is deciding which nodes in the hierarchy are the likely candidates for CA. One choice is to use nodes that the BVH traversal stops at during the closest distance query, called the *front nodes*. In practice, these nodes have a higher probability to realize τ than the others. However, the depth of the front nodes in the hierarchy can be high and thus the number of nodes in the front can be rather large. This affects the performance of our CCD algorithm since we need to apply CA to these nodes repeatedly. In Sec. V, we propose a novel scheme, called controlled CA (C^2A), to control the depth of the front nodes during the CA iterations, thereby improving the performance of 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 towards the final CA iterations, we perform exact distance computation. The choice of the level of distance approximation determines the depth of front nodes or the number of nodes used in our computation.

IV. Motion Bound Calculation

In this section, we present our algorithm to compute the motion bound, μ. This bound is computed for the swept sphere volumes (SSVs) and triangle primitives when these primitives undergo rigid transformation $\mathbf{M}(t)$ with constant translational velocity $\mathbf{v}$ and rotational velocity ω.

A bounding volume, SSV consists of PSS, LSS and RSS, which are defined as the Minkowski sums between a point, line and rectangle with a sphere in 3D, respectively (see a 2D illustration of SSV in Fig.2). Let α denote a BV or triangle primitive of $\mathfrak{A}$. Further, let $\mathbf{p}_i$ be a point on α, $\mathbf{n}$ be the closest direction between α, β, $\mathbf{r}_i$ be a vector from the origin of the local body frame attached to $\mathfrak{A}$, to $\mathbf{p}_i$. The maximum trajectory length (or motion bound) of α, projected onto $\mathbf{n}$ is given as [17]:

$$\begin{aligned}\mu &= \max_i \int_0^1 |\dot{\mathbf{p}}_i(t)\cdot\mathbf{n}|dt \\ &\le |\mathbf{v}\cdot\mathbf{n}| + \int_0^1 \max_i |\omega\times\mathbf{n}\cdot\mathbf{r}_i(t)|dt \\ &\le |\mathbf{v}\cdot\mathbf{n}| + \int_0^1 \max_i |\mathbf{c}(t)\cdot\mathbf{r_i}|dt,\end{aligned} \tag{3}$$

where $\mathbf{c}(t) = \mathbf{R}^{-1}(t)\mathbf{n}\times\omega$ and $\mathbf{R}$ is the rotational component of $\mathbf{M}(t)$. In Eq. 3, the first term is constant. Therefore, our goal is maximize the second term. Notice that $\mathbf{c}(t)$ is coplanar and forms a plane S whose normal is ω. The maximal value of the second term in Eq. 3 can be obtained by the maximal projection of $\mathbf{r}_i$ onto S as shown below.

Lemma 1 *Let α be a SSV. Then, the directed motion bound μ of α is:*

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

where

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

and r is the radius of a sphere used for constructing SSV, and $\mathbf{c}_i$ are the endpoints of the generator primitives[2] of SSV. In other words, for PSS, LSS and RSS, $i = 1, 1\ldots2, 1\ldots4$, respectively.

Proof: We start by analytically representing $\mathbf{r_i}$ for the three kinds of SSV as follows (also see Fig. 2):

- For PSS, $\mathbf{r_i} = \mathbf{c_1} + \mathbf{r}$, where $\mathbf{c}_1$ is the endpoint of PSS and $\mathbf{r}$ is some point on a sphere centered at the origin with a radius of r .
- For 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 both end points of the line in LSS. $\mathbf{r}$ is defined similarly as for the PSS.
- For 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 three endpoints of the rectangle in RSS. $\mathbf{r}$ is defined similarly as above.

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

$$|\mathbf{c}(t)\cdot\mathbf{r_i}| = \left|\mathbf{c}(t)\cdot\mathbf{r_i}^{\perp}\right| \le \|\mathbf{c}(t)\|\cdot\left\|\mathbf{r_i}^{\perp}\right\| \le \|\mathbf{c}(t)\|\,r + \|\mathbf{c}(t)\|\left\|\mathbf{k}^{\perp}\right\|, \tag{4}$$

where $\mathbf{k}^{\perp}$ is the projection of $\mathbf{k}$ to the plane S. It is known that the projection of a point to a plane is a point, the projection of a line to a plane is a line or a point, and the projection of a rectangle to a plane is a quadrangle or a line. Since the projection of $\mathbf{k}$ to S should be inside the projection of the generator primitive of SSV to S, we get the following relationship:

[2]The generator primitive of PSS, LSS and RSS is a point, a line, and a rectangle, respectively.

$$\left\|\mathbf{k}^{\perp}\right\| \leq \max\left(\left\|\mathbf{c_i}^{\perp}\right\|\right) \ , \quad \left\|\mathbf{c_i}^{\perp}\right\| = \frac{\|\mathbf{c_i} \times \omega\|}{\|\omega\|} \tag{5}$$

By using Eq.'s 3, 4, 5, we obtain the result of the lemma. ■

We can compute a motion bound for a triangle primitive by projecting the triangle's vertices to bound the range of projection.

Lemma 2 *Let α be a triangle primitive. Then the directed motion bound μ of α is given as:*

$$\mu \leq |\mathbf{v}\cdot\mathbf{n}| + \|\omega \times \mathbf{n}\| \left(\max\left(\left\|\mathbf{c_i}^{\perp}\right\|\right)\right)$$

where

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

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

Proof: Similar to Lemma 1 ■

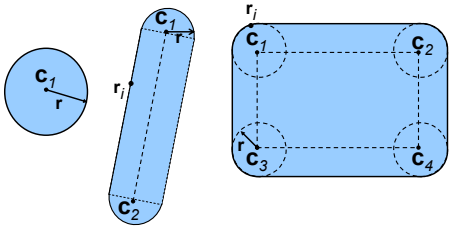

Fig. 2. **Swept Sphere Volume in 2D.** *From left to right: PSS, LSS and RSS.*

V. Controlled Advancement

We first analyze the cost function for performing CCD and then present our controlled advancement scheme to minimize the cost function.

A. Cost Analysis

We present a simple formula to measure the performance of our CCD algorithm as follows:

$$T = 2T_{BVH} + N_\tau \times N_{BV} \times T_{CA}, \tag{6}$$

where T is the total cost function for CCD, T_{BVH} is the cost of BVH construction of a polygonal model, N_τ is the total number of CA iterations to find τ, N_{BV} is the number of bounding volume pairs that CA is applied to, and T_{CA} is the cost to evaluate the CA equation in Eq. 2. In our case, T_{BVH} and T_{CA} are constants. As a result, the main goal is to design efficient methods to reduce N_τ and N_{BV}.

B. Balancing N_τ and N_{BV}

As explained in Sec. III-B, we apply CA operations to the front nodes of the BVH that are computed during the closet distance query. Therefore, N_{BV} corresponds to the number of the front nodes. In order to decrease N_{BV}, we reduce or *control* the depth of the front nodes by terminating the BVH traversal early during the closest distance query; see Fig. 3. The result of early termination yields only approximate distance bound, typically smaller than the actual closest distance. Therefore, the advancement time-step Δt_i based on Eq. 2 would also be less tight since $d(\mathfrak{A}(t), \mathfrak{B})$ is smaller. In our experiments, we have observed that when $i < j$, $\Delta t_i \gg \Delta t_j$, in particular when i is equal to one or two. Therefore, when i is small (i.e. during the first few iterations of CA), having a smaller value of $d(\mathfrak{A}(t), \mathfrak{B})$ results in a useful value of Δt_i. However, decreasing N_{BV} for some i's may result in more CA iterations N_τ and in that case reducing both N_τ and N_{BV} might be hard. We need to balance N_τ and N_{BV} to reduce the cost function, Eq. 6. Thus, to prevent an excessive number of iterations, for each iteration, we check whether the

Algorithm 1 C^2A: Controlled Conservative Advancement
Input: BVH nodes $n_{\mathfrak{A}}, n_{\mathfrak{B}}$, current closest distance d_{cur}, current advancement step Δt_{cur}, CA controlling variable w
Output: Updated d_{cur} and Δt_{cur}
{Initially, $C^2A(BVH_{\mathfrak{A}}.root,\ BVH_{\mathfrak{B}}.root\ \infty,\ 1,\ w)$ is called and $w < 1$. }

```
if # of CA iterations > R_max or d_cur < D_threshold then
    w = 1;
end if
if n_A and n_B are leaf nodes then
    d = Distance(n_A, n_B); {Using the PQP library}
    if(d < d_cur) d_cur = d;
    Δt = CalculateCAStep(d, n_A, n_B); {Using Eq. 2}
    if(Δt < Δt_cur) Δt_cur = dt;
    return d_cur, Δt_cur
end if
if n_A is not a leaf node then
    A = n_A.leftchild; B = n_A.rightchild; C = D = n_B;
    d_1 = Distance(A, C); d_2 = Distance(B, D);
else
    A = B = n_A; C = n_B.leftchild; D = n_B.rightchild;
    d_1 = Distance(A, C); d_2 = Distance(B, D);
end if
if d_2 < d_1 then
    if d_2 < w d_cur then
        return C^2A(B, D, d_cur, Δt_cur, w);
    else
        Δt = CalculateCAStep(d_2, B, D);
        if(Δt < Δt_cur) Δt_cur = Δt;
    end if
    if d_1 < w d_cur then
        return C^2A(A, C, d_cur, Δt_cur, w);
    else
        Δt = CalculateCAStep(d_1, A, C);
        if(Δt < Δt_cur) Δt_cur = Δt;
    end if
else
    if d_1 < w d_cur then
        return C^2A(A, C, d_cur, Δt_cur, w);
    else
        Δt = CalculateCAStep(d_1, A, C);
        if(Δt < Δt_cur) Δt_cur = Δt;
    end if
    if d_2 < w d_cur then
        return C^2A(B, D, d_cur, Δt_cur, w);
    else
        Δt = CalculateCAStep(d_2, B, D);
        if(Δt < Δt_cur) Δt_cur = Δt;
    end if
end if
```

approximate distance is smaller than some threshold value or whether N_τ becomes too big. If it is the case, we don't use early termination and traverse all the way to the leaf nodes to compute the closest distance.

There are different ways to enforce the early termination during the closest distance query. Our choice is that during the recursive call for distance query, we provide a *faked*, small distance value as an initial distance value[3] to the recursive function. This will cause the recursive distance query to terminate early when the recursion cannot improve the faked, current distance value. Finally, when the entire recursive traversal is over, we collect the front nodes where the recursion stops and use them for the CA operations.

The pseudocode of our algorithm is given in Algorithm 1. In our implementation, we use $w = 0.3 \sim 0.5$ for the first few iterations to control the advancement; we reset $w = 1$ toward the end of CA iteration.

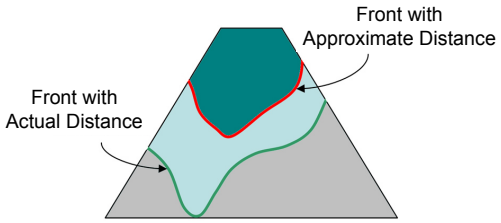

Fig. 3. **Controlling the Depth of Front Nodes.** *This image shows the front nodes with different depth values. The different front nodes are obtained by terminating the recursion early during the closest point query with approximate and exact distance values.*

C. Early Rejection

We use a simple, early rejection scheme to further improve the CCD performance. 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 bound value Λ_i, we prune away the node and its children nodes. Initially, we set $\Lambda_1 = 1$ but for kth iteration, set $\Lambda_k = 1 - \sum_{i=1}^{k-1} \Delta t_i$. This works as the condition $\sum \Delta t_i > 1$ imply no collision for the node n during $[0,1]$ and thus we can prune it away.

VI. Results and Discussions

In this section, we describe our implementation and highlight its results on various benchmarks. We implemented our CCD algorithm using C++ on a PC running Windows XP, equipped with an Intel Core Q9450 2.66GHz CPU and 2.75GB main memory. We use a public-domain library PQP for SSV hierarchy construction and distance queries, and modify it to suit for our purpose. We benchmark our CCD algorithm using models of varying complexities, ranging from 1K to 105K triangles, as shown in Fig. 5. The performance over all the benchmarks is shown in Table I.

We adopt the benchmarking setup used in [17] to measure the performance of our algorithm; 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 (e.g. see Fig. 6). We repeat this test for more than 200 trials and record the performance. We use three benchmarks corresponding to polygon-soup models; Club VS Club, Gear VS Gear and Hammer VS CAD piece. In case of club VS club, the interpolated motion between $\mathbf{q}_0$ and $\mathbf{q}_1$ yields collisions for all the trials (i.e. $\tau < 1$); in gear VS gear, two thirds of trials generate collisions but for the rest there are no colliding configurations. In the hammer VS CAD model, the configurations are similar to the gear VS gear scenario. As shown in Table I, the controlled advancement improves the performance of our CCD algorithm by a factor of $1 \sim 28$. In Fig. 4, we show the number of front nodes (N_{BV}) for each trial. By using controlled advancement, N_{BV} reduces by 82.3% on average.

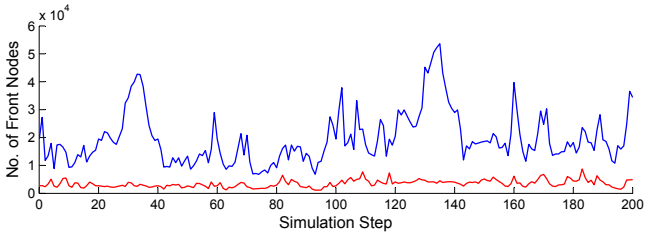

Fig. 4. **Number of Front Nodes.** *The red line shows the number of front nodes using controlled conservative advancement whereas the blue one is without it for the Club VS Club benchmark*

We also compare the performance of our algorithm with that of FAST [17] using the same benchmarks: Bunny VS Bunny, Bunny Dynamics, and Torusknot VS Torusknot (as shown in 6). For these benchmarks, we do not take advantage of the connectivity information in the models, though FAST exploits it. However, the performance of our algorithm is better than that of FAST due to controlled advancement. Finally, our software implementation is available for download at `http://graphics.ewha.ac.kr/C2A`.

VII. Conclusions

In this paper, we have presented an interactive CCD algorithm for polygon-soup models. The algorithm is based on tight motion bound calculation for swept sphere volumes (SSV) and conservative advancement to adaptively control the performance. One of the limitations of our algorithm is that distance calculation based on SSV is still a bottleneck. For future work, we would like to extend our approach to articulated models, N-body scenarios and deformable models [19], [22], [23]. Moreover, we are interested in using the algorithm for different robotic applications such as motion planning and dynamics.

Acknowledgements

This research work was supported in part by the KRF Grant funded by the Korean Government (KRF-2007-331-D00400) and the IT R&D program of MKE/IITA (2008-F-033-01, Development of Real-time Physics Simulation Engine for e-Entertainment). Dinesh Manocha was supported in part by NSF, ARO, Intel and

[3]Under normal circumstances, this value should be initialized as the infinity or some cached distance from last time instance.

Benchmark	# of Triangles	CCD without C^2A	CCD with C^2A			FAST
			Collision	Collision-free	Total	
Club VS. Club	104.8K (each)	14.97	3.6	–	3.6	–
Gear VS. Gear	25.6K (each)	55.49	2.96	0.0048	1.98	–
Hammer VS. CAD piece	1.7K, 2.6K	7.68	2.82	0.0052	1.89	–
Bunny VS. Bunny	69.7K (each)	8.64	4.11	0.048	2.77	4.01
Bunny Dynamics	26.4K (each)	0.41	5.54	0.11	0.22	0.31
Torusknot VS. Torusknot	34.6K (each)	6.8	2.81	0.41	2.01	1.96

TABLE I

Benchmark Statistics. *Each row represents, from left to right, benchmarking type, the number of triangles in each model, the performance of CCD algorithm without controlled advancement (in milli-sec), the performance of CCD algorithm with controlled advancement (in milli-sec) with only collision-free configurations, colliding configurations, average query time over collision-free and colliding configurations, and performance of FAST.*

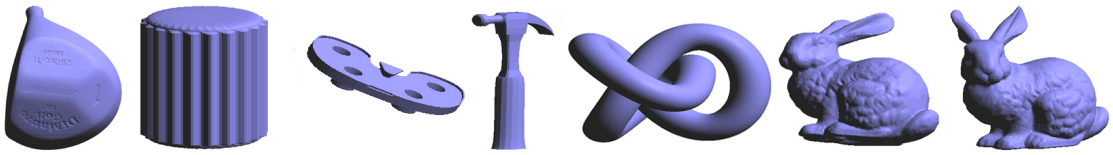

Fig. 5. **Benchmarking Models.** *From left to right (with a triangle count): Gear (25.6K), Club (104.8K), CAD Piece (2.6K), Hammer (1.7K), Torusknot (34.6K), Bunny1 (26K), Bunny2 (70K).*

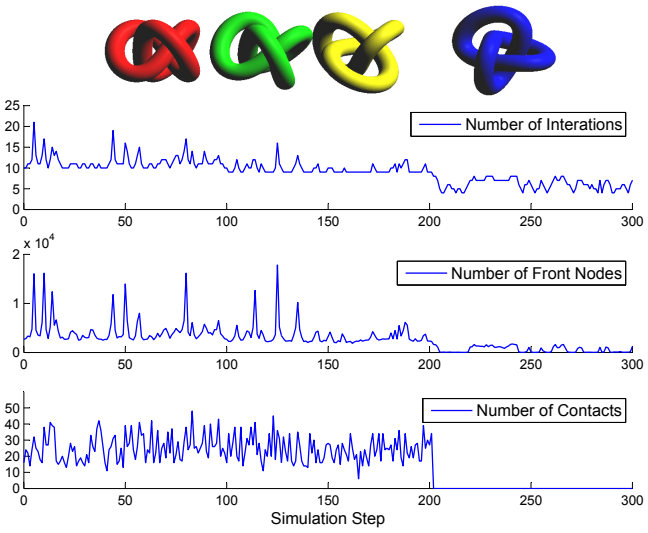

Fig. 6. **Profiling Torusknot vs Torusknot.** *Top: the red, blue, yellow, and green torusknot represents the initial* $\mathbf{q}_0$ *and final* $\mathbf{q}_1$ *configurations of* $\mathfrak{A}$*, the configuration of* $\mathfrak{B}$*, and the configuration of* $\mathfrak{A}$ *at* τ*. From the second 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.*

RDECOM. We would also like to thank Xinyu Zhang, Liangjun Zhang and Xin Huang for their help.

REFERENCES

[1] M. C. Lin, "Efficient collision detection for animation and robotics," Ph.D. dissertation, University of California, Berkeley, CA, Dec. 1993.

[2] B. V. Mirtich, "Impulse-based dynamic simulation of rigid body systems," Ph.D. dissertation, University of California, Berkeley, 1996.

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

[4] S. Redon and M. Lin, "Practical local planning in the contact space," *Proc. of IEEE ICRA*, 2005.

[5] L. Zhang and D. Manocha, "Constrained motion interpolation with distance constraints," in *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2008.

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

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

[8] J. F. Canny, "Collision detection for moving polyhedra," *IEEE Trans. PAMI*, vol. 8, pp. 200–209, 1986.

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

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

[11] S. Redon, A. Kheddar, and S. Coquillart, "An algebraic solution to the problem of collision detection for rigid polyhedral objects," *Proc. of IEEE Conference on Robotics and Automation*, 2000.

[12] K. Abdel-Malek, D. Blackmore, and K. Joy, "Swept volumes: Foundations, perspectives, and applications," *International Journal of Shape Modeling*, 2002.

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

[14] 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.

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

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

[17] X. Zhang, M. Lee, and Y. J. Kim, "Interactive continuous collision detection for non-convex polyhedra," *The Visual Computer*, pp. 749–760, 2006.

[18] S. Gottschalk, M. Lin, and D. Manocha, "OBB-Tree: A hierarchical structure for rapid interference detection," *Proc. of ACM Siggraph'96*, pp. 171–180, 1996.

[19] X. Zhang, S. Redon, M. Lee, and Y. J. Kim, "Continuous collision detection for articulated models using Taylor models and temporal culling," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, vol. 26, no. 3, p. 15, 2007.

[20] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha, "Fast continuous collision detection for articulated models," in *Proceedings of ACM Symposium on Solid Modeling and Applications*, 2004.

[21] S. Redon, Y. J. Kim, M. C. Lin, D. Manocha, and J. Templeman, "Interactive and continuous collision detection for avatars in virtual environments," in *Proceedings of IEEE VR Conference*, 2004.

[22] N. Govindaraju, D. Knott, N. Jain, I. Kabal, R. Tamstorf, R. Gayle, M. Lin, and D. Manocha, "Collision detection between deformable models using chromatic decomposition," *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)*, vol. 24, no. 3, pp. 991–999, 2005.

[23] S. Curtis, R. Tamstorf, and D. Manocha, "Fast collision detection for deformable models using representative-triangles," *Proc. of ACM Symposium on Interactive 3D Graphics and Games*, 2008.