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 assumptions 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 milliseconds 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 MSL¹ 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 milliseconds 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. {tanmin|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

¹http://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] solve an extension of [11] to articulated models and their algorithm suffers from a similar problem. Schwarz 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 \( A \) and \( B \) be two polygon-soup models in 3D, where \( A \) is movable under rigid transformation \( M(t) \). Without loss of generality we assume that \( B \) is fixed. Further, we assume that the initial and final configurations of \( A \) are given as \( q_0 \) and \( q_1 \) at time \( t = 0 \) and \( t = 1 \), respectively. We also define \( A(t) = M(t)A \). The problem of CCD can be formulated as checking whether Eq. 1 has a feasible solution:

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

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, \( \tau \).

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

\[
\Delta \tau \leq \frac{d(A(t), B)}{\mu}. \tag{2}
\]

Then, the first time of contact \( \tau \) can be obtained by repeatedly calculating the time-step \( \Delta \tau \) and summing it (i.e. \( \tau = \sum \Delta \tau \)) until \( d(A(\tau), B) \) 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 \( A \) are given as \( q_0, q_1 \). Using these configurations, we compute a continuous motion \( M(t) \) to interpolate \( q_0, q_1 \) with constant translational and rotational velocities. If we know the actual motion of \( A \) a priori (e.g. [15]), we approximate...
the motion with $M(t)$ 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, $\mu$ 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 a SSV. We start by analytically representing $r_i$ for the three kinds of SSV as follows (also see Fig. 2):

- For PSS, $r_1 = c_1 + r$, where $c_1$ is the endpoint of PSS and $r$ is some point on a sphere centered at the origin with a radius of $r$.
- For LSS, $r_1 \in (sc_1 + (1-s)c_2 + r)$ and $s \in [0,1]$, where $c_1$ and $c_2$ are both end points of the line in LSS. $r$ is defined similarly as for the PSS.
- For RSS, $r_1 \in (sc_1 + (1-s)c_2 + \mu (c_3-c_1)) + r$, $s \in [0,1]$ and $\mu \in [0,1]$ where $c_1$, $c_2$ and $c_3$ are the three endpoints of the rectangle in RSS. $r$ is defined similarly as above.

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

$$|c(t) \cdot r_i| = |c(t) \cdot r + k^\perp| \leq |c(t)||r + k||k^\perp|,$$  

where $k^\perp$ is the projection of $k$ 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 $k$ to $S$ should be inside the projection of the generator primitive of SSV to $S$, we get the following relationship:

$$
\mu = \max_i \int_0^1 |\dot{p}_i(t) \cdot n| dt \\
\leq |v \cdot n| + \int_0^1 \max_i |\omega \times r_i(t)| dt \\
\leq |v \cdot n| + \int_0^1 \max_i |c(t) \cdot r_i| dt,
$$

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

**Lemma 1** Let $\alpha$ be a SSV. Then, the directed motion bound $\mu$ of $\alpha$ is:

$$\mu \leq |v \cdot n| + \|\omega \times n\| (r + \max_i (|c_i^\perp|),$$

where $c^\perp_i = \|c_i \times \omega\| / \|\omega\|$, and $r$ is the radius of a sphere used for constructing SSV, and $c_i$ are the endpoints of the generator primitives of SSV. In other words, for PSS, LSS and RSS, $i = 1,1\ldots2,1\ldots4$, respectively.

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

- For PSS, $r_1 = c_1 + r$, where $c_1$ is the endpoint of PSS and $r$ is some point on a sphere centered at the origin with a radius of $r$.
- For LSS, $r_1 \in (sc_1 + (1-s)c_2 + r)$ and $s \in [0,1]$, where $c_1$ and $c_2$ are both end points of the line in LSS. $r$ is defined similarly as for the PSS.
- For RSS, $r_1 \in (sc_1 + (1-s)c_2 + \mu (c_3-c_1)) + r$, $s \in [0,1]$ and $\mu \in [0,1]$ where $c_1$, $c_2$ and $c_3$ are the three endpoints of the rectangle in RSS. $r$ is defined similarly as above.

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

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 $\alpha$ denote a BV or triangle primitive of $\alpha$. Further, let $\dot{p}$ be a point on $\alpha$, $n$ be the closest direction between $\alpha, \beta$, $r_i$ be a vector from the origin of the local body frame attached to $\alpha$, to $\dot{p}$. The maximum trajectory length (or motion bound) of $\alpha$, projected onto $n$ is given as [17]:

$$\mu = \max_i \int_0^1 |\dot{p}_i(t) \cdot n| dt \\
\leq |v \cdot n| + \int_0^1 \max_i |\omega \times r_i(t)| dt \\
\leq |v \cdot n| + \int_0^1 \max_i |c(t) \cdot r_i| dt,$$

The generator primitive of PSS, LSS and RSS is a point, a line, and a rectangle, respectively.
\[ \| k \| \leq \max(\| c_1 \|, \| c_2 \|), \quad \| c_i \| = \frac{\| c_i \times \omega \|}{\| \omega \|} \] (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 \( \alpha \) be a triangle primitive. Then the directed motion bound \( \mu \) of \( \alpha \) is given as:

\[ \mu \leq |v \cdot n| + \| \omega \times n \| \left( \max \left(\| c_i \| \right) \right) \]

where
\[ \| c_i \| = \frac{\| c_i \times \omega \|}{\| \omega \|} \]

and \( c_i \) are the vertices of \( \alpha \), and \( i = 1 \ldots 3 \).

**Proof:** Similar to Lemma 1

![Swept Sphere Volume in 2D](image)

**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:

\[ T = 2T_{BVH} + N_T \times N_{BV} \times T_{CA}, \] (6)

where \( T \) is the total cost function for CCD, \( T_{BVH} \) is the cost of BVH construction of a polygonal model, \( N_T \) is the total number of CA iterations to find \( \tau \), \( N_{BV} \) 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 \( N_T \) and \( N_{BV} \).

**B. Balancing \( N_T \) 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 closest 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 \( \Delta t \) based on Eq. 2 would also be less tight since \( d(\Omega (t), \Theta) \) 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 \( d(\Omega (t), \Theta) \) results in a useful value of \( \Delta t \). However, decreasing \( N_{BV} \) for some \( i \)'s may result in more CA iterations \( N_T \) and in that case reducing both \( N_T \) and \( N_{BV} \) might be hard. We need to balance \( N_T \) 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 \( C^2 \): Controlled Conservative Advancement**

**Input:** BVH nodes \( n_{N_0}, n_{N_2} \), current closest distance \( d_{cur} \), current advancement step \( \Delta t_{cur} \), CA controlling variable \( w \)

**Output:** Updated \( d_{cur} \) and \( \Delta t_{cur} \)

{Initially, \( C^2(A(BVH_{N_0}.root, BVH_{N_2}.root \Rightarrow 1, w) \) is called and \( w < 1 \).}

1. if # of CA iterations > \( R_{max} \) or \( d_{cur} < D_{threshold} \) then
2. \( w = 1; \)
3. end if
4. if \( n_{N_0} \) and \( n_{N_2} \) are leaf nodes then
5. \( d = \text{Distance}(n_{N_0}, n_{N_2}); \) {Using the PQP library}
6. if \( d < d_{cur} \) \( d_{cur} = d; \)
7. \( \Delta t = \text{CalculateCAStep}(d, n_{N_0}, n_{N_2}); \) {Using Eq. 2}
8. if \( \Delta t < \Delta t_{cur} \) \( \Delta t_{cur} = \Delta t; \)
9. return \( d_{cur}, \Delta t_{cur} \)
10. end if
11. if \( n_{N_0} \) is not a leaf node then
12. \( A = n_{N_0}.leftchild; \) \( B = n_{N_0}.rightchild; \) \( C = D = n_{N_2}; \)
13. \( d_1 = \text{Distance}(A, C); \) \( d_2 = \text{Distance}(B, D); \)
14. else
15. \( A = B = n_{N_0}; \) \( C = n_{N_0}.leftchild; \) \( D = n_{N_0}.rightchild; \)
16. \( d_1 = \text{Distance}(A, C); \) \( d_2 = \text{Distance}(B, D); \)
17. end if
18. if \( d_2 < d_1 \) then
19. if \( d_2 < wd_{cur} \) then
20. return \( C^2(A, B, D, d_{cur}, \Delta t_{cur}, w); \)
21. else
22. \( \Delta t = \text{CalculateCAStep}(d_2, B, D); \)
23. if \( \Delta t < \Delta t_{cur} \) \( \Delta t_{cur} = \Delta t; \)
24. end if
25. if \( d_1 < wd_{cur} \) then
26. return \( C^2(A, C, d_{cur}, \Delta t_{cur}, w); \)
27. else
28. \( \Delta t = \text{CalculateCAStep}(d_1, A, C); \)
29. if \( \Delta t < \Delta t_{cur} \) \( \Delta t_{cur} = \Delta t; \)
30. end if
31. else
32. if \( d_1 < wd_{cur} \) then
33. return \( C^2(A, C, d_{cur}, \Delta t_{cur}, w); \)
34. else
35. \( \Delta t = \text{CalculateCAStep}(d_1, A, C); \)
36. if \( \Delta t < \Delta t_{cur} \) \( \Delta t_{cur} = \Delta t; \)
37. end if
38. if \( d_2 < wd_{cur} \) then
39. return \( C^2(A, B, D, d_{cur}, \Delta t_{cur}, w); \)
40. else
41. \( \Delta t = \text{CalculateCAStep}(d_2, B, D); \)
42. if \( \Delta t < \Delta t_{cur} \) \( \Delta t_{cur} = \Delta t; \)
43. end if
44. end if
45. return \( C^2(A, B, D, d_{cur}, \Delta t_{cur}, w); \)
46. else
47. \( \Delta t = \text{CalculateCAStep}(d_1, A, C); \)
48. if \( \Delta t < \Delta t_{cur} \) \( \Delta t_{cur} = \Delta t; \)
49. end if
50. return \( C^2(A, B, D, d_{cur}, \Delta t_{cur}, w); \)
51. end if
approximate distance is smaller than some threshold value or whether \( N_f \) 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.

\(^3\)Under normal circumstances, this value should be initialized as the infinity or some cached distance from last time instance.

**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 \( \Delta t_i \) is greater than some upper bound value \( \Lambda_i \), we prune away the node and its children nodes. Initially, we set \( \Lambda_i = 1 \) but for \( k \)th 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 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](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...
<table>
<thead>
<tr>
<th>Benchmark</th>
<th># of Triangles</th>
<th>CCD without C²A</th>
<th>CCD with C²A</th>
<th>Collision-Free</th>
<th>Total</th>
<th>FAST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Club VS. Club</td>
<td>104.8K (each)</td>
<td>14.97</td>
<td>3.6</td>
<td>3.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gear VS. Gear</td>
<td>25.6K (each)</td>
<td>55.49</td>
<td>2.96</td>
<td>0.0048</td>
<td>1.98</td>
<td></td>
</tr>
<tr>
<td>Hammer VS. CAD piece</td>
<td>1.7K, 2.6K</td>
<td>7.68</td>
<td>2.92</td>
<td>0.0052</td>
<td>1.89</td>
<td></td>
</tr>
<tr>
<td>Bunny VS. Bunny</td>
<td>69.7K (each)</td>
<td>8.64</td>
<td>4.11</td>
<td>0.048</td>
<td>2.77</td>
<td>4.01</td>
</tr>
<tr>
<td>Bunny Dynamics</td>
<td>26.4K (each)</td>
<td>0.41</td>
<td>5.54</td>
<td>0.11</td>
<td>0.22</td>
<td>0.31</td>
</tr>
<tr>
<td>Torusknot VS. Torusknot</td>
<td>34.6K (each)</td>
<td>6.8</td>
<td>2.81</td>
<td>0.41</td>
<td>2.01</td>
<td>1.96</td>
</tr>
</tbody>
</table>

**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.

[Image 93x483 to 261x519]

**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).

[Image 93x552 to 519x610]

**Fig. 6.** **Profiling Torusknot vs Torusknot.** Top: the red, blue, yellow, and green torusknot represents the initial \( q_0 \) and final \( q_1 \) configurations of \( \mathcal{B} \), the configuration of \( \mathcal{B} \), and the configuration of \( \mathcal{S} \) at \( \tau \). From the second row to bottom: the number of CA iterations \( N_\text{CA} \), the number of front nodes \( N_\text{BV} \), 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**


