Continuous Collision Detection for Articulated Models using Taylor Models and Temporal Culling

Xinyu Zhang[1] Stephane Redon[2] Minkyoung Lee[1] Young J. Kim[1]
[1] Ewha Womans University, Korea [2] INRIA Rhone-Alpes, France
http://graphics.ewha.ac.kr/CATCH

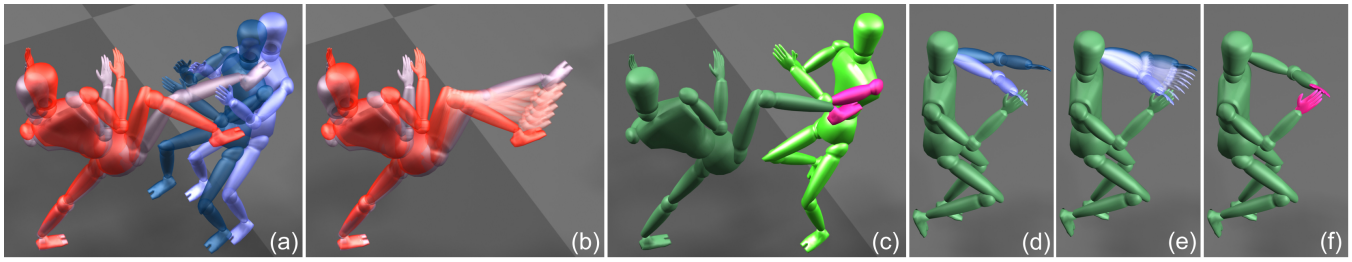


Figure 1: Continuous Collision Detection between Articulated Models. **(a)** *Initial (dark red and blue) and final (light red and blue) configurations of two moving mannequin models consisting of 15 links and 20K triangles each.* **(b)** *Motion interpolation from the initial and final configurations.* **(c)** *Finding the first time of contact between the two mannequins (contact features highlighted in pink).* **(d)(e)(f)** *Self-collision between the left and right arms of a mannequin. Our algorithm can perform such continuous collision detection in a fraction of a milli-second.*

Abstract

We present a fast continuous collision detection (CCD) algorithm for articulated models using Taylor models and temporal culling. Our algorithm is a generalization of conservative advancement (CA) from convex models [Mirtich 1996] to articulated models with non-convex links. Given the initial and final configurations of a moving articulated model, our algorithm creates a continuous motion with constant translational and rotational velocities for each link, and checks for interferences between the articulated model under continuous motion and other models in the environment and for self-collisions. If collisions occur, our algorithm reports the first time of contact (TOC) as well as *collision witness features*. We have implemented our CCD algorithm and applied it to several challenging scenarios including locomotion generation, articulated-body dynamics and character motion planning. Our algorithm can perform CCDs including self-collision detection for articulated models consisting of many links and tens of thousands of triangles in 1.22 ms on average running on a 3.6 GHz Pentium 4 PC. This is an improvement on the performance of prior algorithms of more than an order of magnitude.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

Keywords: Continuous Collision Detection; Articulated Models; Convex Decomposition; Conservative Advancement; Dynamics Simulation

1 Introduction

Collision detection (CD) is the problem of checking for possible interferences between geometric models moving in space. In computer graphics, non-penetration is an important constraint in making objects look physically believable and CD plays a crucial role in enforcing this constraint. Besides computer graphics, CD also has many applications in other fields such as robotics, geometric modeling, human computer interfaces, and virtual reality, that involve non-penetration constraints. As a result, CD has been extensively studied in these fields over the past two decades and is typically considered to be a mature technology, with robust implementations available for certain types of objects (e.g., rigid objects). However, most of the CD techniques developed so far are by nature *discrete*, in the sense that these techniques only detect interferences between *static models* with fixed spatial configurations. As objects move over some time interval, a simple heuristic can detect collisions by checking for intersections at some fixed times during that interval, but it cannot actually guarantee the non-penetration of moving objects. In interactive applications such as computer games, severe side-effects may be caused by a rapidly moving object momentarily passing through a wall or a door.

Recently, *continuous collision detection* (CCD) has received a lot of attention from different research communities because it offers robust handling of non-penetration constraints for rigid models. These include time of contact (TOC) computations for impulse-based dynamics [Mirtich 1996], constraint-based dynamics [Redon et al. 2002], god-object computation in 6-DOF haptic rendering [Ortega et al. 2006], local planning in sampling-based motion planning [Schwarzer et al. 2002; Redon and Lin 2005], avatar interaction in virtual reality environments [Redon et al. 2004a] and game physics engines [Ageia 2006; Havok 2006; Coumans 2006]. Despite the increasing demand for CCD, however, its use has been limited since it typically requires higher computational costs than discrete collision detection, particularly for articulated models.

Main Results: In this paper, we introduce a novel, efficient CCD algorithm for articulated models which takes into account the *continuous* motion of objects, and reports all possible interferences (see Fig. 1). Furthermore, if collisions do occur, we can determine their first time of contact (TOC) and report the colliding features (vertex/face or edge/edge contacts), also known as *the contact manifold* or *collision witness features*. Our CCD algorithm is a generalization of conservative advancement (CA) [Mirtich 1996; Mirtich 2000] to articulated models and consists of one preprocessing step and two run-time steps: (1) preprocessed link-level bounding volume hierar-

chy (BVH) construction, (2) link-level *spatial culling* using Taylor models and a dynamic BVH and (3) exact contact determination using CA and *temporal culling*. In particular, our CCD algorithm has the following novel aspects:

- Link-level spatial culling is based on dynamic BVH construction and culling. An efficient BVH using axis-aligned bounding boxes (AABBs) is constructed using *Taylor models*, a generalization of interval arithmetic (Sec. 5).
- Given a continuous motion of an articulated body with constant angular and translational velocities for each link, we provide an efficient, recursive method to calculate a tight upper bound on the motion. This bound serves as the basis on which to employ CA for exact contact determination (Sec. 6).
- In order to alleviate the problem of quadratic complexity of performing CAs for all possible link combinations, we propose a novel sorting scheme based on times of contact. In practice, this scheme drastically reduces the complexity to a few pairs of links (Sec. 7).
- Our algorithm handles CCD between different articulated models as well as between elements of the same model (self-CCD). Moreover, our algorithm fully utilizes well-studied, fast discrete CD techniques such as [Ehmann and Lin 2001] to achieve highly interactive performance; e.g., 1.22 *ms* for an articulated model consisting of $20K$ triangles and 15 links against an environment consisting of $101K$ triangles. In our benchmarks, our algorithm outperforms existing CCD algorithms by more than an order of magnitude (Sec. 8).

2 Previous Work

Most prior work on collision detection has been focused on discrete algorithms. We briefly review work relevant to our algorithm including other known CCD methods and interval arithmetic.

2.1 Discrete Collision Detection

Discrete CD algorithms can be broadly categorized into specialized algorithms for convex polytopes and general algorithms for polygonal or spline models based on spatial partitioning or bounding volume hierarchies (BVH). We refer readers to [Lin and Manocha 2003] for an extensive survey of the field.

Euclidean distance calculation is an integral part of our CCD algorithm and has been extensively studied in the literature. Like discrete CD algorithms, distance calculation algorithms can be classified into those for convex objects and those for non-convex objects, again see [Lin and Manocha 2003]. We will pay particular attention to the *Voronoi marching* algorithm for convex polytopes, which exploits motion coherence [Lin 1993], and its extension to general polyhedral models based on a convex-hull tree [Ehmann and Lin 2001], since robust implementations of these techniques are publicly available and show good run-time behavior in practice.

2.2 Continuous Collision Detection

Six different approaches to CCD have been presented in the literature: algebraic equation-solving [Canny 1986; Choi et al. 2006; Kim and Rossignac 2003; Redon et al. 2000], swept volumes [Abdel-Malek et al. 2002; Hubbard 1993], adaptive bisection [Redon et al. 2002; Schwarzer et al. 2002], kinetic data structures (KDS) [Agarwal et al. 2001; Kim et al. 1998; Kirkpatrick et al. 2000], the configuration space approach [van den Bergen 2004], and conservative advancement [Mirtich 1996; Mirtich 2000; Coumans 2006; Zhang et al. 2006]. However, most of these approaches are relatively slow (i.e., not real-time) or work only for restricted types of objects such as 2D polygonal models, convex models, or simple algebraic models. Consequently, the applicability of these techniques has been rather limited compared to discrete CD algorithms. The algorithms based on adaptive bisection show a good run-time performance and can handle *polygon soups*; however, since they do not take advantage of the topological information in polyhedral models or of motion coherence in CCD computations, they are slower than [Zhang et al. 2006] for polyhedral models. However, it would not be trivial to extend [Zhang et al. 2006] to articulated models because the many links of an articulated model may complicate the motion bound calculation and the number of collision checks would increase quadratically with the number of links. More details about these issues are explained in Sec.'s 6 and 7 of this paper.

Recently, there have been attempts to devise efficient CCD algorithms for articulated models [Redon et al. 2004a; Redon et al. 2004b] because of the increasing popularity of articulated models in computer graphics and virtual reality. However, [Redon et al. 2004a] handles only simple, capsule-shaped avatar models for real-time applications, while [Redon et al. 2004b] works for general, articulated models but its performance is relatively slow. Moreover, these techniques do not consider self-collisions between links belonging to the same model. In contrast, our algorithm can handle general, articulated models and self-collisions, and its performance is more than an order of magnitude faster than [Redon et al. 2004b].

2.3 Interval Arithmetic and Variants

Our CCD algorithm uses a generalization of interval arithmetic to compute a link-level AABB hierarchy for a moving articulated model (see Sec. 5). Interval arithmetic has its origins in the pioneering work of Moore [Moore 1979]. Despite its ability to provide guaranteed error bounds on floating point operations, interval arithmetic's well-known drawback is its increased conservativeness as the complexity of the underlying operations grows (see Sec. 5.1). Several extensions and generalizations have been discussed to overcome this problem within the scope of collision query [Bühler et al. 2004]. Affine arithmetic [Comba and Stolfi 1993] is an extension that preserves correlations between error terms, and reduces bounds size when error terms counterbalance one another. Ellipsoid arithmetic [Neumaier 1993] was introduced to overcome the so-called "wrapping effect", that occurs when applying interval arithmetic to study dynamic systems. Taylor models [Berz and Hofstätter 1998] provide higher-order representations of the bounded functions. Our algorithm relies on Taylor models to compute dynamic bounding volumes which significantly improve the efficiency in culling when applying CCD to complicated articulated models.

2.4 Organization

The rest of this paper is organized as follows. In Sec. 3, we provide the preliminary concepts necessary to describe our CCD algorithm, and then we introduce our approach in Sec. 4. In Sec. 5, we explain how to construct a dynamic BVH using Taylor models and how to perform spatial culling. In Sec. 6, we derive an upper bound on motion for CA and, in Sec. 7, the temporal culling method is presented. We demonstrate our implementation in different benchmarking scenarios in Sec. 8, and conclude this paper in Sec. 9.

3 Preliminaries

In this section, we provide some preliminary concepts to help us describe our CCD algorithm including notation, our representation of an articulated model and the concept of conservative advancement.

3.1 Notations and Representations

Given an articulated model $\mathscr{A}$ made up of m rigid links $\mathscr{A}_1, \ldots, \mathscr{A}_m$ with no closed loop, we use a directed acyclic graph to represent the link structure of $\mathscr{A}$, where each vertex in the graph denotes a link and each edge corresponds to the joint connecting two links. We assume that many links may share the same parent link, but each individual link has only one parent (although the root link has no

parent). For the sake of simplicity, we will assume that the index of the parent of link i is $i-1$. This notation can be easily modified when a parent has multiple children.

For a given link i, let $\{i\}$ denote its associated reference frame, and let $\{0\}$ represent the world reference frame. Let us further represent the orientation of $\{i\}$ relative to $\{j\}$ as ${}^j_i\mathbf{R}$ on SO(3) when $j<i$. Similarly, the relative transformation of $\{i\}$ with respect to $\{j\}$ at time t can be described by ${}^j_i\widehat{\mathbf{M}}(t)$ in SE(3). We will use $\widehat{\mathbf{M}}$ to distinguish a time-dependent, continuous transformation in SE(3) from an instance $\mathbf{M}$ of SE(3); we will also use ${}^j_i\widehat{\mathbf{M}}$ to represent ${}^j_i\widehat{\mathbf{M}}(t)$ if it is clear in the context. A similar notation is applied to $\widehat{\mathbf{R}}$ and $\widehat{\mathbf{T}}$. We use ${}^j\mathbf{L}_i$ to denote the vector from the origin of $\{j\}$ to that of $\{i\}$. Any point on link i can be represented by ${}^j_i\mathbf{r}$ with respect to $\{j\}$.

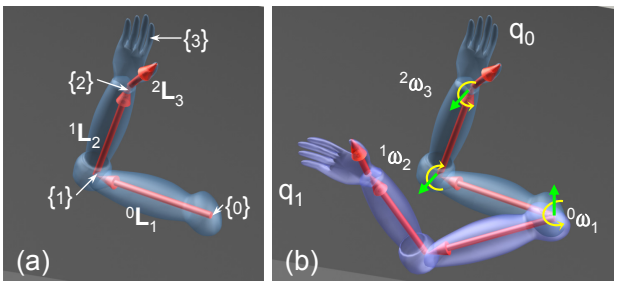


Figure 2: Three-link Articulated Arm. ***(a)*** *Reference frames and* ${}^j\mathbf{L}_i$. ***(b)*** *Link i moving in the reference frame of its parent (link* $i-1$*) with a constant rotational velocity.*

3.2 Motion Interpolation

As is the case with other CCD algorithms, one input to our algorithm must be a continuous motion of the models under consideration. However, in many graphics applications, such a continuous motion is unfortunately not known in advance. For example, in interactive graphics applications with the user in the loop, it is hard to predict an intended continuous motion [Redon et al. 2002]. Therefore, our algorithm assumes that only the initial and final configurations $\mathbf{q}_0$ and $\mathbf{q}_1$ of an articulated model are given, and synthesizes a continuous motion to interpolate $\mathbf{q}_0$ and $\mathbf{q}_1$ over the time interval $[0,1]$. This motion is generated as a linear motion with constant translational and rotational velocities for each link i, forming the whole motion ${}^0_i\widehat{\mathbf{M}}(t)$ in configuration space. This simple choice accelerates the entire CCD computation pipeline. The interpolating motion ${}^0_i\widehat{\mathbf{M}}(t)$ can be derived similarly to [Redon et al. 2004b]:

$$ {}^0_i\widehat{\mathbf{M}}(t) = {}^0_1\widehat{\mathbf{M}}(t)\cdot{}^1_2\widehat{\mathbf{M}}(t)\ldots{}^{i-1}_i\widehat{\mathbf{M}}(t) \tag{1} $$

$$ {}^{i-1}_i\widehat{\mathbf{M}}(t) = \begin{pmatrix} {}^{i-1}_i\widehat{\mathbf{R}}(t) & {}^{i-1}_i\widehat{\mathbf{T}}(t) \\ (0,0,0) & 1 \end{pmatrix} \tag{2} $$

where ${}^{i-1}_i\widehat{\mathbf{T}}(t), {}^{i-1}_i\widehat{\mathbf{R}}(t)$ are the position and orientation of $\{i\}$ with respect to $\{i-1\}$ at a given time t in $[0,1]$, respectively. For more details, see [Zhang and Kim 2007].

3.3 CCD Problem Formulation

Assume each link is having a continuous motion ${}^0_i\widehat{\mathbf{M}}(t)$. The goal of our algorithm is to detect the collision of any moving link with any other objects in the environment (inter-object CCD) as well as collisions between different links in the same model (intra-objects or self-CCD). If any collision or self-collision actually occurs, the algorithm returns the time of initial collision (TOC). More formally, given two given links $\mathscr{A}_i$, $\mathscr{B}_j$ in articulated bodies $\mathscr{A},\mathscr{B}$ with respective associated continuous motions ${}^0_i\widehat{\mathbf{M}}_{\mathscr{A}}(t), {}^0_j\widehat{\mathbf{M}}_{\mathscr{B}}(t)$, we want to know whether the following set is non-empty:

$$ \{\, t\in[0,1] \mid {}^0_i\widehat{\mathbf{M}}_{\mathscr{A}}(t)\mathscr{A}_i \cap {}^0_j\widehat{\mathbf{M}}_{\mathscr{B}}(t)\mathscr{B}_j \neq \emptyset \}. \tag{3} $$

If this set is not empty, we want to determine the minimum value of t (called the TOC, τ) that makes this set non-empty and we also want to construct the contact manifold of links $\mathscr{A}_i, \mathscr{B}_j$ at τ[1]. Notice that if $\mathscr{A}=\mathscr{B}$ and $|i-j|>1$, the event is the self-collision between non-adjacent links[2].

3.4 Conservative Advancement

The framework underlying our CCD algorithm is conservative advancement (CA) [Mirtich 1996; Coumans 2006; Zhang et al. 2006]. CA is a simple technique to compute a lower bound on τ by repeatedly advancing a movable convex object $\mathscr{A}$ by Δt toward another fixed convex object $\mathscr{B}$ while avoiding collision. Here, Δt is calculated from the closest distance $d(\mathscr{A}(t),\mathscr{B})$ between $\mathscr{A}(t)$ and $\mathscr{B}$, and the upper bound μ on the motion along $d(\mathscr{A}(t),\mathscr{B})$, traced at a rate of $\mathscr{A}(t)$ over the time interval $[0,1]$, as shown in Eq. 4:

$$ \Delta t \;\leq\; \frac{d(\mathscr{A}(t),\mathscr{B})}{\mu}. \tag{4} $$

$\mathscr{A}$ can then safely advance from time t to time $t+\Delta t$ without collision (see Fig. 3). The TOC, τ, is obtained by accumulating such time-steps $\sum\Delta t$ until $d(\mathscr{A}(t),\mathscr{B})$ becomes less than some user-specified distance threshold. Further, a tighter bound on μ can be obtained by projecting the motion of $\mathscr{A}$ on to the direction $\mathbf{n}$ corresponding to the shortest distance between $\mathscr{A}$ and $\mathscr{B}$.

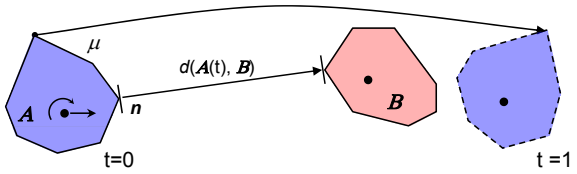


Figure 3: Conservative Advancement.

4 Algorithm Overview

We will now give an overview of our approach to CCD between articulated models. Our algorithm consists of one preprocessing stage and two run-time stages as follows.

Preprocess: for each link $\mathscr{A}_i$ in an articulated model $\mathscr{A}$, we precompute its oriented bounding box (OBB $\Diamond_{\mathscr{A}_i}$) [Gottschalk et al. 1996; Barequet and Har-Peled 2001] and a BVH of convex hulls [Ehmann and Lin 2001] based on a *surface* convex decomposition. At run-time, $\Diamond_{\mathscr{A}_i}$ is used to construct a BVH of AABBs for the entire model $\mathscr{A}$ to perform link-level spatial culling, and the BVH of convex hulls is used for distance calculation between the links for CA.

Link-level spatial culling: At run-time, for each link $\mathscr{A}_i$, an AABB $\square_{\mathscr{A}_i}$ is first created to bound the extent of the motion of the associated link-level OBB ($\Diamond_{\mathscr{A}_i}$) under the interpolating motion. To compute a tight set of AABBs, we use a generalization of interval arithmetic called Taylor models [Berz and Hofstätter 1998]. Then, a dynamic BVH of AABBs is recursively built using the boxes $\square_{\mathscr{A}_i}$

[1] $\tau>1$ means that there is no collision in the time interval $[0,1]$.

[2] We do not consider a case of self-collision between adjacent links, since that is often prohibitive in practice due to the kinematic constraints between joints.

as leaf-level nodes. Based on this hierarchy, we can successively cull those links that are far from the environment (see Sec. 5). For a self-collision test, however, we do not use the entire hierarchy; instead we check all pairwise combinations between its leaves (i.e., the boxes $\square_{\mathscr{A}_i}$). From this, *potentially colliding links* are identified and are fed into the next stage.

Determining times of collision with CA and temporal culling: For each remaining potentially colliding link, we calculate the time of collision and the contact features using the CA technique. CA is performed by comparing the closest distance between links, computed using a convex-hull tree, against an upper bound on the motion (see Section 6). During each iteration of the CA algorithm, we use a novel temporal culling method that quickly reduces the number of potential collisions so as to avoid unnecessary iterations (see Sec. 7).

5 Dynamic BVH Culling

Given the interpolating motion between successive frames at runtime, we first compute a dynamic hierarchy of bounding volumes for each articulated model $\mathscr{A}$ over the time interval $[0,1]$. We use this hierarchy to cull away links $\mathscr{A}_i$ that are far from the environment and to identify potential collisions between links.

5.1 Taylor Models

We build the dynamic BVH using *Taylor models*, a generalization of interval arithmetic which reduces the overestimation errors that inevitably occur in interval arithmetic.

Consider a function $f : [0,\pi/2] \to \mathbb{R}$, $f(t) = \cos(t) + \sqrt{3}\sin(t)$, which is similar to those involved in the description of the interpolating motion ${}^{i-1}_{i}\widehat{\mathbf{M}}(t)$ in Sec. 3.2. Suppose that we want to bound this function over the time interval $[0,\pi/2]$. Using standard interval arithmetic, we can perform the following sequence of operations:

$$\begin{array}{lll} t \in [0,\pi/2] & \Rightarrow & \cos(t) \in [0,1], \\ t \in [0,\pi/2] & \Rightarrow & \sin(t) \in [0,1], \\ & \Rightarrow & \sqrt{3}\sin(t) \in [\sqrt{3},\sqrt{3}] \times [0,1] = [0,\sqrt{3}], \\ & & \\ & \text{thus} & f(t) \in [0,1] + [0,\sqrt{3}] = [0,1+\sqrt{3}]. \end{array}$$

But it can be easily shown that the tightest bounding interval of f over $[0,\pi/2]$ is actually $[1,2]$. Interval arithmetic has allowed us to rapidly obtain conservative bounds on f, but the crude approximation of the elementary sub-functions (here, the cosine and sine functions) has led to an overestimation of the bounds. In the case of the interpolating motion of an articulated model, which is the product of local motions (*cf* . Eq. 1), standard interval arithmetic produces bounds that exponentially overestimate the link motion (*cf* . Fig. 5). This results in decreasing efficiency as the depth complexity of the articulation increases [Redon et al. 2004b].

Definitions One major reason for overestimation in the above example is that interval arithmetic does not keep track of the correlations between the sub-functions. Taylor models have been introduced in order to model functions with higher-order descriptions (*e.g.,* [Bühler et al. 2004]). Let f be a C^∞ time-dependent function. An nth-order Taylor model T_f of f over the interval $[t_0,t_1]$ is composed of the nth-order Taylor polynomial P of f at some point $m \in [t_0,t_1]$, and an interval remainder $R = [r_0,r_1]$, such that:

$$f(t) \in \sum_{i=0}^{n} \frac{f^{(i)}(m)}{i!}(t-m)^i + [r_0,r_1], \tag{5}$$

for all $t \in [t_0,t_1]$. In brief, a Taylor model is a *conservative polynomial enclosure of a function*. Arithmetic operations on Taylor models of an identical order can easily be defined [Berz and Hofstätter 1998]. For example, the addition of an nth-order Taylor model (P_f,R_f) of a function f and an nth-order Taylor model (P_g,R_g) of a function g is simply (P_f+P_g,R_f+R_g), which is an nth-order Taylor model of the function $f+g$. Similarly, Taylor models of vectors and matrices can be defined (each component being a Taylor model), and linear algebra operations can be performed. Taylor models provide a generalization of interval arithmetic that reduces to standard interval arithmetic for $n = 0$.

In our implementation, we determine third-order[3] Taylor models of the cosine and sine functions in ${}^{i-1}_{i}\widehat{\mathbf{R}}(t)$ in Eq. 2 over a time interval I by computing their Taylor polynomials at the *midpoint* of I, and then determining the *exact* resulting interval remainder (using simple analysis). The Taylor models of the affine functions ${}^{i-1}_{i}\widehat{\mathbf{T}}(t)$ in Eq. (2) are the affine functions themselves (provided $n \geq 1$), and their interval remainder is $[0,0]$.

5.2 Constructing Dynamic Bounding Volume Hierarchies of Articulated Models using Taylor Models

We use Taylor models to efficiently construct a dynamic BVH for each articulated model. First, we represent each component in the orientation matrix and translation vector in Eq. 2 over the time interval $[0,1]$ using third-order Taylor models, which results in 4×4 homogeneous *Taylor matrices*. Then, we concatenate these matrices (as in Eq. 1) to obtain a Taylor matrix representing the whole link motion.

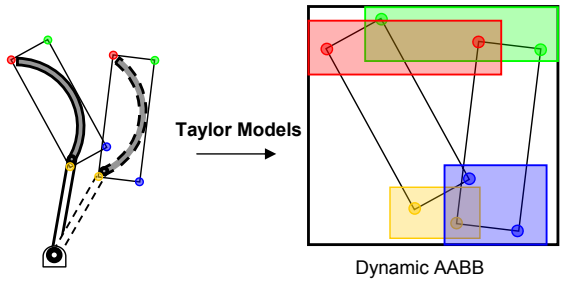


Figure 4: *(***Left***) The motion of the OBB of a link and (***Right***) dynamic AABB obtained by applying Taylor models to all the vertices in the OBB.*

If we apply the latter Taylor matrix to a given point in the link reference frame, we obtain a *Taylor vector* which models the motion of this point over the time interval $[0,1]$. Finally, bounding each Taylor component over this time interval produces an AABB which bounds the trajectory of the point over the time interval[4]. Thus, we apply this strategy to each of the eight corner vertices of the OBB of a link $\mathscr{A}_i$, and obtain eight AABBs $\square_{\mathscr{A}_i}$ bounding the trajectories of these corners during $[0,1]$. By a simple convexity argument, the AABB which bounds these eight AABBs bounds the whole link $\mathscr{A}_i$ during $[0,1]$ (*cf* . Fig. 4). Fig. 5 shows how Taylor models can greatly improve the quality of bounds for articulated-body motions compared to standard interval arithmetic [Redon et al. 2002; Redon et al. 2004b]. The figure shows much better culling of the AABBs when the number of links is high and models are in close proximity with their environment (e.g., sliding contacts in rigid-body dynamics [Baraff and Witkin 2001] or god objects in 6-DOF haptic rendering [Ortega et al. 2006]).

Once we have computed the dynamic AABBs $\square_{\mathscr{A}_i}$ for all links in $\mathscr{A}$, we compute a hierarchy of these boxes (i.e. a dynamic BVH of AABBs) in a bottom-up manner around the entire articulated

[3]Third-order Taylor models are chosen in our implementation to maintain at least two non-zero, non-linear terms in the Taylor series of sine and cosine functions, and to efficiently compute the bounds on their remainders as well.

[4]The bounds on each Taylor model are computed exactly using simple analysis, since we use third-order Taylor models.

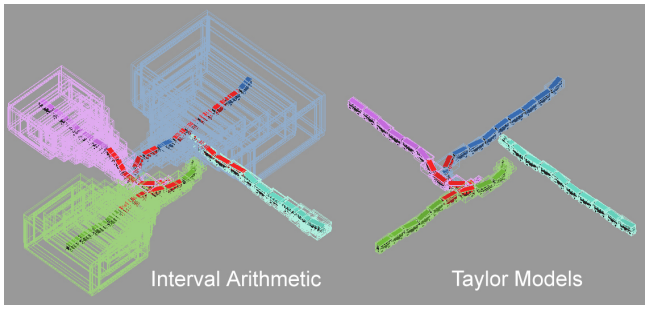


Figure 5: Using Taylor models to bound link motions. *Taylor models allow us to reduce the overestimation problem inherent in interval arithmetic, thus to reduce the PCLs (highlighting in red).* **Left:** *Although the initial and final positions of the train model consisting of 10 car links each (cf. Fig. 10) are relatively close to each other, interval arithmetic alone fails to provide accurate bounds on the link motions. The displayed AABBs exponentially overestimate the link motions.* **Right:** *By allowing better representations of local link motions, Taylor models produce more accurate bounds on the link motions.*

model $\mathscr{A}$. We then check recursively for interference with the environments and cull subsets of the links that do not collide with the environment or other links. The links that survive the culling process are called *potentially colliding links* (PCLs). These links are fed to the next step, in which the first times of contact between them are found.

6 Motion Bound Calculation

The machinery of CA requires two basic components (Eq. 4): a distance calculation ($d(\mathscr{A}(t),\mathscr{B})$) and the determination of an upper bound (μ) on the motion. We need to perform these two computations efficiently if CA is to be viable. Typical distance algorithms for rigid models such as [Ehmann and Lin 2001] are applicable to a rigid link $\mathscr{A}_i$ in an articulated model $\mathscr{A}$. We concentrate on deriving a tight upper bound μ on $\mathscr{A}_i$ under a linear motion ${}^0_i\widehat{\mathbf{M}}(t)$ in configuration space. To simplify the following discussion, we will assume that the link geometry is convex; otherwise, techniques such as [Zhang et al. 2006], based on convex decomposition, and BVH of convex hulls will need to be applied to non-convex links.

Let ${}^0_i\mathbf{p}$ be a point on a link $\mathscr{A}_i$ in the world reference frame $\{0\}$. As $\mathscr{A}_i$ undergoes ${}^0_i\widehat{\mathbf{M}}(t)$, ${}^0_i\mathbf{p}$ will trace out a trajectory ${}^0_i\mathbf{p}(t)$ in 3D space. In order to calculate μ, we must first derive the velocity ${}^0\mathbf{v}_i = {}^0_i\dot{\mathbf{p}}(t)$ of ${}^0_i\mathbf{p}$ in the world reference frame. Then, an upper bound on the motion μ of the entire link $\mathscr{A}_i$ is:

$$\mu = \max_{{}^0_i\mathbf{p}\in\mathscr{A}_i} \int_0^1 \left|{}^0_i\dot{\mathbf{p}}(t)\cdot\mathbf{n}\right| dt = \max_{{}^0_i\mathbf{p}\in\mathscr{A}_i} \int_0^1 \left|{}^0\mathbf{v}_i\cdot\mathbf{n}\right| dt, \qquad (6)$$

where $\mathbf{n}$ is the direction vector that points at the closest other object in the environment [Zhang et al. 2006].

We will now present a novel, recursive way to calculate μ. We will use ${}^0\mathbf{v}_{i-j}$ to denote the velocity of the origin of link $i-j$ when $j>0$; however, when $j=0$, ${}^0\mathbf{v}_i$ represents the velocity of the point ${}^0_i\mathbf{p}$ on link $\mathscr{A}_i$ that we want to find. The velocity of point $\mathbf{p}$ on link i is that of link $i-1$ plus a new velocity term contributed by the motion of link i. Therefore, ${}^0\mathbf{v}_i$ is equal to the velocity of the origin of link $i-1$, ${}^0\mathbf{v}_{i-1}$, plus the translational and rotational velocities of link i. This can be written as follows [Craig 1989]:

$${}^0\mathbf{v}_i = {}^0\mathbf{v}_{i-1} + \left({}^0_{i-1}\widehat{\mathbf{R}}\ {}^{i-1}\mathbf{v}_i + {}^0\omega_i \times {}^0_{i-1}\widehat{\mathbf{R}}\ {}^{i-1}\mathbf{L}_i\right), \qquad (7)$$

where ${}^0_{i-1}\widehat{\mathbf{R}}$ is the orientation matrix of $\{i-1\}$ relative to $\{0\}$. The term ${}^{i-1}\mathbf{v}_i$ is the translational velocity of link i with respect to its parent link's reference frame $\{i-1\}$; ${}^0\omega_i$ is the rotational velocity of link i with respect to the reference frame $\{0\}$; and ${}^{i-1}\mathbf{L}_i$ is a vector from a point on link i to its center of rotation, which is usually the origin of frame $\{i-1\}$. Since velocities can be added when they are represented with respect to the same reference frame [Spong et al. 2005], we have:

$${}^0\omega_i = \sum_{k=1}^{i} {}^0_{k-1}\widehat{\mathbf{R}}\ {}^{k-1}\omega_k,$$

where ${}^{k-1}\omega_k$ is the rotational velocity of link k with respect to its parent link's reference frame $\{k-1\}$. Therefore, we can rewrite Eq. 7 as:

$${}^0\mathbf{v}_i = {}^0\mathbf{v}_{i-1} + \left({}^0_{i-1}\widehat{\mathbf{R}}\ {}^{i-1}\mathbf{v}_i + \left(\sum_{k=1}^{i} {}^0_{k-1}\widehat{\mathbf{R}}\ {}^{k-1}\omega_k\right) \times {}^0_{i-1}\widehat{\mathbf{R}}\ {}^{i-1}\mathbf{L}_i\right).$$

We can apply this recursive formulation successively from link to link. Consequently, we can obtain the velocity of point $\mathbf{p}$ with respect to the world reference frame as follows:

$${}^0\mathbf{v}_i = \sum_{j=1}^{i} \left({}^0_{j-1}\widehat{\mathbf{R}}\ {}^{j-1}\mathbf{v}_j + \left(\sum_{k=1}^{j} {}^0_{k-1}\widehat{\mathbf{R}}\ {}^{k-1}\omega_k\right) \times {}^0_{j-1}\widehat{\mathbf{R}}\ {}^{j-1}\mathbf{L}_j\right),$$

where ${}^0_0\widehat{\mathbf{R}} = \mathbf{I}$ (the identity matrix). If $j=i$, ${}^{j-1}\mathbf{L}_j$ is a vector from the point $\mathbf{p}$ on link j to its center of rotation (i.e. the origin of frame $\{j-1\}$), which is ${}^{j-1}_j\mathbf{r}_\mathbf{p}(t)$. Note that ${}^{j-1}_j\mathbf{r}_\mathbf{p}(t) = {}^{i-1}_i\widehat{\mathbf{R}}\ {}^{j-1}_j\mathbf{r}_\mathbf{p}(0)$. If $j<i$, then ${}^{j-1}\mathbf{L}_j$ is the vector from the origin of link j to the origin of frame $\{j-1\}$. As the origin of the link j undergoes a constant translational motion ${}^{j-1}\mathbf{v}_j$ relative to $\{j-1\}$, at any time t over the time interval $[0,1]$ ${}^{j-1}\mathbf{L}_j$ is ${}^{j-1}\mathbf{L}_j(0) + {}^{j-1}\mathbf{v}_j t$, where ${}^{j-1}\mathbf{L}_j(0)$ is the same vector at $t=0$. In summary, ${}^{j-1}\mathbf{L}_j$ can be represented in its parent's frame as follows:

$${}^{j-1}\mathbf{L}_j(t) = \begin{cases} {}^{j-1}\mathbf{L}_j(0) + {}^{j-1}\mathbf{v}_j t & j<i \\ {}^{i-1}_i\widehat{\mathbf{R}}\ {}^{j-1}_j\mathbf{r}_\mathbf{p}(0) & j=i. \end{cases} \qquad (8)$$

By projecting the velocity ${}^0\mathbf{v}_i$ onto the closest direction vector $\mathbf{n}$ and integrating it over the time interval $[0,1]$, we obtain Eq. 10, where

$$\left|{}^{j-1}\mathbf{L}_j(t)\right|_\mu = \begin{cases} \max\left(\left|{}^{j-1}\mathbf{L}_j(0)\right|, \left|{}^{j-1}\mathbf{L}_j(1)\right|\right) & j<i \\ \max\limits_{{}^0_i\mathbf{p}\in\mathscr{A}_i}\left(\left|{}^{j-1}_j\mathbf{r}_\mathbf{p}\right|\right) & j=i \end{cases} \qquad (9)$$

Notice that the motion of root $\mathscr{A}_0$ affects the motion bound of $\mathscr{A}_i$ linearly with respect to the depth complexity i of $\mathscr{A}_i$ when other factors remain constant such as $\mathbf{n}$ and ${}^{j-1}\mathbf{L}_j$. For more derivation details, see [Zhang and Kim 2007].

7 Temporal Culling for CA

Culling using a dynamic hierarchy of bounding volumes can efficiently eliminate links that are not likely to collide with the environment or with other links. Now we need to apply the CA algorithm to each pair of potentially colliding links (PCLs) that have survived the spatial culling and find the minimum TOC. However, the efficiency of the spatial culling process begins to degrade when many links are in close proximity but do not actually collide. This results in many PCLs going through to the CA step, which is the most expensive run-time procedure in our algorithm. We will therefore present a novel collision-time sorting and temporal culling method to reduce the number of CA iterations.

$$\max_{{}^0_i\mathbf{p}} \int_0^1 \left|{}^0\mathbf{v}_i \cdot \mathbf{n}\right| dt = \max_{p} \int_0^1 \left| \left(\sum_{j=1}^{i-1} \left({}^0_{j-1}\widehat{\mathbf{R}}\ {}^{j-1}\mathbf{v}_j + \left(\sum_{k=1}^{j} {}^0_{k-1}\widehat{\mathbf{R}}\ {}^{k-1}\omega_k \right) \times {}^0_{j-1}\widehat{\mathbf{R}}\ {}^{j-1}\mathbf{L}_j \right) \right) \cdot \mathbf{n} \right| dt$$

$$\leq \left|{}^0\mathbf{v}_1 \cdot \mathbf{n}\right| + \left|\mathbf{n} \times {}^0\omega_1\right| \left|{}^0\mathbf{L}_1(t)\right|_\mu + \sum_{j=2}^{i-1} \left(\left|{}^{j-1}\mathbf{v}_j\right| + \left(\left|\mathbf{n} \times {}^0\omega_1\right| + \sum_{k=2}^{j} \left|{}^{k-1}\omega_k\right| \right) \left|{}^{j-1}\mathbf{L}_j(t)\right|_\mu \right) \quad (10)$$

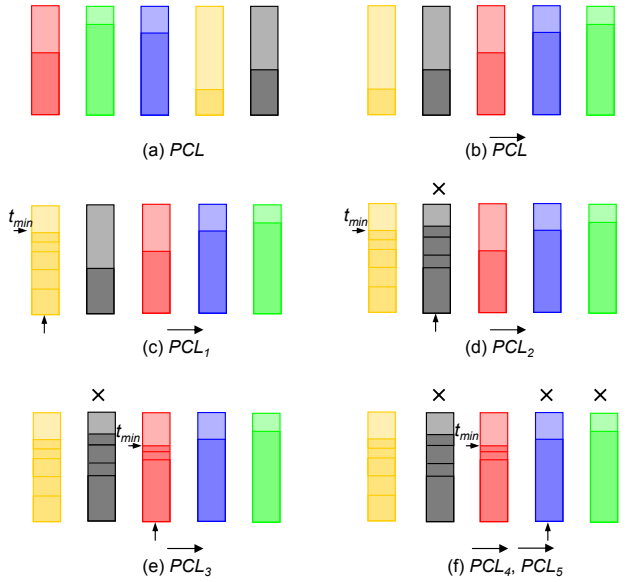


Figure 6: Temporal Culling. ***(a)*** *Bars in different colors represent different link pairs PCL_i. The shaded height in each bar represents a lower bound $\breve{\tau}_i$ on τ_i, obtained by either Δt_i^1 or $\breve{\tau}_{\mathrm{CH}(i)}$ (Sec. 7.2).* ***(b)*** *PCL sorted into $\overrightarrow{PCL}$ in ascending order of $\breve{\tau}_i$.* ***(c)*** *Four more CAs are performed on $\overrightarrow{PCL}_1$ until we find τ_1 and set $t_{\min} = \min(\tau_1, 1.0)$.* ***(d)*** *Iterative CAs on $\overrightarrow{PCL}_2$ are performed, but after three more iterations, since $\sum_{j=1}^{4} \Delta t_2^j > t_{\min}$, we abandon $\overrightarrow{PCL}_2$ and move on $\overrightarrow{PCL}_3$.* ***(e)*** *Two more CAs are performed on $\overrightarrow{PCL}_3$ until we find τ_3. Since $\tau_3 < t_{\min}$, we set $t_{\min} = \min(\tau_3, t_{\min})$.* ***(f)*** *$\overrightarrow{PCL}_4$ and $\overrightarrow{PCL}_5$ are abandoned immediately, since $\breve{\tau}_4 > t_{\min}$ and $\breve{\tau}_5 > t_{\min}$, Finally, we set $\tau = t_{\min}$.*

7.1 Collision-time Sorting

A straightforward way of finding the time of contact τ of all potentially colliding links is to apply CA to each pair of links, so that $\tau = \min_i(\tau_i)$, where τ_i is the collision time of the ith PCL (PCL_i) and $\tau_i = \sum_{j=1}^{N_i} \Delta t_i^j$ where Δt_i^j is the advancement time step determined as a result of the jth CA iteration for PCL_i, and N_i is the total number of CA iterations for PCL_i. However, this scheme does not consider temporal correlations between link pairs in PCLs when it performs CA. If we can conservatively determine a pair of links (say PCL_i) which has a smaller τ_i value than some other pairs (say PCL_j), then we do not need to perform CA for those other PCL_j's. In order to calculate the final TOC τ, we need to perform CA iterations for all PCLs which takes computation time T:

$$T = \sum_{j=1}^{N_1} T_1^j + \sum_{j=1}^{N_2} T_2^j + \ldots + \sum_{j=1}^{N_n} T_n^j, \quad (11)$$

where T_i^j is the time spent on the jth CA iteration for PCL_i. To reduce the computational cost T we can either decrease N_i or T_i^j.

We will now show how to reduce N_i using temporal correlations, and we will explain in the next section how to reduce both N_i and T_i^j using model simplification.

Our main approach to reducing N_i is, instead of *individually* completing all the CA iterations for each pair of links PCL_i, *simultaneously* performing CA iterations for all pairs to determine pairs PCL_j that have no chance of realizing τ. As a result, many of the terms N_j in Eq. 11 are reduced to a single term. We use the following procedure:

1. **Initialization:** for each PCL_i, we perform a single iteration of CA and find the advancement step Δt_i^1. Note that Δt_i^1 is a lower bound on the TOC for PCL_i, i.e., $\Delta t_i^1 \leq \tau_i$. We use $\breve{\tau}_i$ to denote the lower bound on the TOC for PCL_i, so $\breve{\tau}_i = \Delta t_i^1$.

2. **Sorting:** we sort the list of PCLs into ascending order of $\breve{\tau}_i$; let us denote this sorted list as $\overrightarrow{PCL}$ and the ith element in $\overrightarrow{PCL}$ as $\overrightarrow{PCL}_i$. If a link pair PCL_i has a small $\breve{\tau}_i$, it is more likely to have the smallest value of τ_i, which is the final collision time τ. After sorting, the link pairs with higher chances of early collision (i.e., smaller τ_i) will be at the front of $\overrightarrow{PCL}$.

3. **Temporal Culling:** we initialize the current estimate $t_{\min}$ of τ as $t_{\min} = 1.0$ and repeat the following steps by retrieving a link pair $\overrightarrow{PCL}_i$ from $\overrightarrow{PCL}$ until $\overrightarrow{PCL}$ becomes empty:

   Case 1: if $\breve{\tau}_i > t_{\min}$, then $\tau_i > t_{\min}$ because $\tau_i \geq \breve{\tau}_i$. Therefore, the final collision time τ cannot be realized by $\overrightarrow{PCL}_i$, and we can safely remove $\overrightarrow{PCL}_i$ from $\overrightarrow{PCL}$. In fact, since $\overrightarrow{PCL}$ is sorted in ascending order of $\breve{\tau}_i$, τ_j is greater than $t_{\min}$ for those link pairs of $\overrightarrow{PCL}_j$ for which $j > i$; therefore, we can remove these pairs.

   Case 2: if $\breve{\tau}_i < t_{\min}$, we continue to perform CA iterations for $\overrightarrow{PCL}_i$ until we find τ_i. After that, if $\tau_i < t_{\min}$, we set $t_{min} = \tau_i$. However, during the CA iterations for $\overrightarrow{PCL}_i$, if the current collision time after the kth iteration, $\sum_{j=1}^{k} \Delta t_i^j$, is greater than $t_{\min}$, we can immediately stop the CA iterations for $\overrightarrow{PCL}_i$, as τ_i is later than the final TOC τ. We then remove $\overrightarrow{PCL}_i$ from $\overrightarrow{PCL}$.

4. **Termination**: we report $t_{\min}$ as the final TOC τ.

Fig. 6 illustrates an example of temporal culling. Compared to Eq. 11, the total computational time for CA iterations using temporal culling is reduced to:

$$T = \sum_{j=1}^{M_1} T_1^j + \sum_{j=1}^{M_2} T_2^j + \ldots + \sum_{j=1}^{M_{m-1}} T_{m-1}^j + \sum_{k=m}^{n} T_k^1 + T_{sort}, \quad (12)$$

where M_i is the total number of the CA iterations for $\overrightarrow{PCL}_i$, m is the first $\overrightarrow{PCL}_m$ with $\breve{\tau}_m > t_{\min}$ (e.g., $m = 4$ in Fig. 6-(f)), and T_{sort} is the time spent on sorting the PCLs. In practice, since $M_i \leq N_i$ and $m \ll n$, and T_{sort} is negligible compared to the CA iteration time, the temporal culling algorithm performs very efficiently. In our benchmarking examples in Sec. 8, m is typically 1 or 2, whereas n can be 300 in some situations.

Algorithm 1 CCD using temporal culling
Input: A list of PCLs.
Output: The first time of contact τ among PCLs.

1: **for** each link pair PCL_i in PCLs **do**
2: calculate its lower bound of TOC, $\check{\tau}_i$.
3: **end for**
4: sort PCL in the ascending order of $\check{\tau}_i$ into $\overrightarrow{PCL}$.
5: $t_{\min} = 1.0$.
6: **for** each $\overrightarrow{PCL_i}$ in $\overrightarrow{PCL}$ **do**
7: **if** $\check{\tau}_i > t_{\min}$ **then**
8: **return** $\tau = t_{\min}$
9: **else**
10: advance the CA time of $\overrightarrow{PCL_i}$ to $\tau_i = \check{\tau}_i$
{Start of CA iterations}
11: **repeat**
12: calculate the closest distance and direction, $d\left(\overrightarrow{PCL_i}\right), \mathbf{n}$.
13: calculate the motion bounds μ (Eq. 10).
14: $\Delta t_i^j = \frac{d\left(\overrightarrow{PCL_i}\right)}{\mu}$ (Eq. 4).
15: **if** $\tau_i + \Delta t_i^j > t_{\min}$ **then**
16: skip $\overrightarrow{PCL_i}$ and jump to 6
17: **else**
18: advance the CA time of $\overrightarrow{PCL_i}$ to $\tau_i = \tau_i + \Delta t_i^j$
19: **end if**
20: **until** $d\left(\overrightarrow{PCL_i}\right) <$ a user-provided threshold ε
{End of CA iterations}
21: $t_{\min} = \min(t_{\min}, \tau_i)$.
22: **end if**
23: **end for**
24: **return** $\tau = t_{\min}$

7.2 Model Simplification

To reduce both N_i and T_i^j in Eq. 11 we employ a different, lower bound on the collision time $\check{\tau}_i$ than the time Δt_i^j for a pair of links PCL_i, which is the value sorted in the temporal culling algorithm in the previous section. More specifically, in step (1) of the temporal culling algorithm, for each PCL_i, we calculate the convex hull of each link pair in PCL_i as a preprocess, and then perform the CA algorithm iteratively to find a collision time based on the convex hulls, $\check{\tau}_{\mathrm{CH}(i)}$. We initialize $\check{\tau}_i$ as $\check{\tau}_{\mathrm{CH}(i)}$. Since calculating the distance between convex objects typically takes much less time than between non-convex objects[5], we can greatly reduce T_i^j while still using the temporal culling algorithm to reduce N_i. If the convex hulls collide at $t = 0$ even though their bounding objects do not intersect but are merely very close to each other, we simply set $\check{\tau}_{\mathrm{CH}(i)} = 0$ and eventually calculate τ_i. Our temporal culling algorithm is summarized as pseudo-code in Alg. 1.

8 Results and Analysis

We will now explain the implementation-dependent details of our algorithm, and show results for applications including locomotion generation, dynamics simulation and motion planning.

8.1 Implementation and Benchmarking

We have implemented our CCD algorithm using Visual C++ on Windows XP. We use a public-domain proximity query library, SWIFT++ [Ehmann and Lin 2001] for distance calculation and for the construction of a hierarchy using convex hulls.

The performance of our algorithm in different applications is shown in Figs. 7-10. The program ran on an Intel P4 3.6GHz PC with 2Gb of main memory. Most of the articulated models used in the benchmarks are highly non-convex, and the links in the models are also non-convex. The user-controlled threshold ε of distance in the definition of τ is 0.001 throughout all the experiments. The performance statistics of our algorithm for different benchmarks are summarized in Table 1.

Benchmarks	T_a	T_f	T_c	T_w
Walk	1.22	0.67	3.63	7.64
Exercise	0.38	0.28	1.69	4.54
Excavator (Fig. 8(b))	780	150	820	3.40K
Excavator (Fig. 8(c))	100	75	120	980
TowerCrane (Fig. 9(b))	5.66	1.32	11.1	134
TowerCrane (Fig. 9(c))	15.1	2.90	27.3	211
Four Trains	535	454	2.60K	4.30K
Falling Trains	274	259	294	1.30K

Table 1: Timing Statistics. *T_a, T_f, T_c represent the average timings (in ms) of all frames, collision-free frames (i.e., $\tau > 1.0$), and collision frames (i.e., $\tau \leq 1.0$), respectively. T_w represents the worst-case timing.*

(1) **Walking Mannequin on a Chessboard** (Fig. 7): a mannequin model walks on a chessboard where 16 chessmen are placed. The mannequin consists of 15 links and 20K triangles, and the chessmen consists of 101K triangles. The locomotion of the mannequin has been generated by creating key poses of the mannequin and running the Footstep™ software in 3DSMax™. We generated the movements of the mannequin without considering collisions, and so the mannequin often collides with chessmen as well as with itself (leg crossing). The performance statistics for this benchmark is also given in Fig. 7(c).

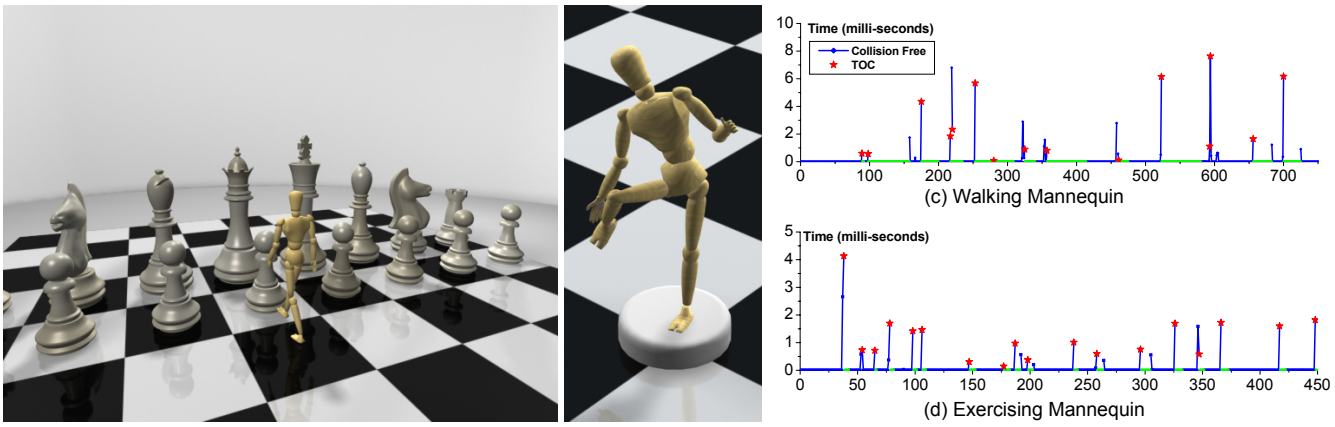
(2) **Exercising Mannequin** (Fig. 7): we created a key-framed animation with self-collisions between links in a mannequin model. For example, as shown in Fig. 7(b), the right hand of the mannequin collides with its own right foot.

(3) **Construction Site in the Toy World** (Figs. 8 and 9): we plug our CCD algorithm into a sampling-based motion planner [MPK-Team 2006; Schwarzer et al. 2002] to find locally collision-free paths between sampled configurations. The planning scenarios include finding collision-free motions for a moving excavator (Fig. 8) and a tower crane (Fig. 9) consisting of 19K and 1.2K triangles, respectively. The environment consists of 0.4M triangles. For a typical sampling-based motion planner such as [MPK-Team 2006], since it is sufficient to test whether a given, local path is collision-free or not, our algorithm does not calculate the accurate TOC for this benchmark; i.e., as soon as the CCD algorithm finds that either a lower bound on τ is greater than 1.0 (collision-free) or an upper bound on τ is less than 1.0 (collision), it immediately returns the status of the path. These planning scenarios are quite challenging because of the object complexities and large displacement between configuration samples. In particular, the excavator benchmark creates very large motions between configuration samples because of the *lazy* mechanism in [MPK-Team 2006].

(4) **Collision Course** (Fig. 10): we simulate the rigid body dynamics of articulated models, apply our CCD algorithm to each pair of consecutive frames in the simulation. In the first scenario, four train models consisting of 10 links and 23K triangles each are collided and tangled with one another. In the second scenario, a train model consisting of 17 links and 42K triangles drops from the sky and comes to rest onto a mountain model consisting of 29K triangles.

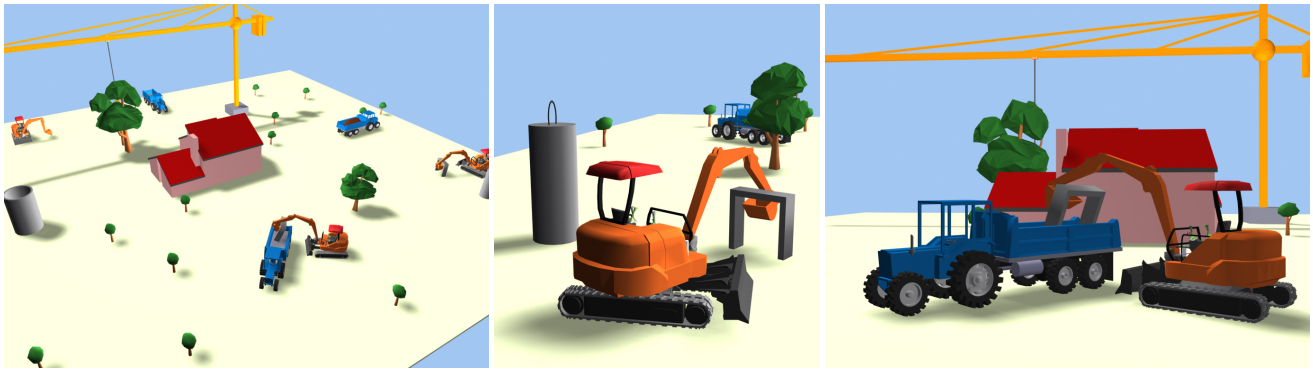
We show the efficiency of spatio-temporal culling for different benchmarks in Table 2. We can notice that the culling rates for both

[5] Our implementation relies on [Ehmann and Lin 2001] for distance calculations. For convex objects, it takes an expected constant time regardless of the model complexity



(a) Walking Mannequin (b) Exercise

Figure 7: Mannequin Benchmarks (a), (b) *A mannequin model consists of 15 links and 20K triangles (4.8K convex pieces), and a chessboard environment consists of 101K triangles (29K convex pieces).* **(c)** *The average timing of the walking mannequin benchmark is 1.22 ms.* **(d)** *The average timing of the exercising mannequin is 0.38ms.*



(a) (b) (c)

Figure 8: Construction Site Benchmark 1 (a) *A construction site scene.* **(b)** *After moving from the initial site to the second, an excavator picks up a weight and loads it into a truck as shown in* **(c)**. *The whole construction site consists of 0.394M triangles and 0.17M convex pieces besides an excavator which consists of 18.94K triangles and 13K convex pieces.*

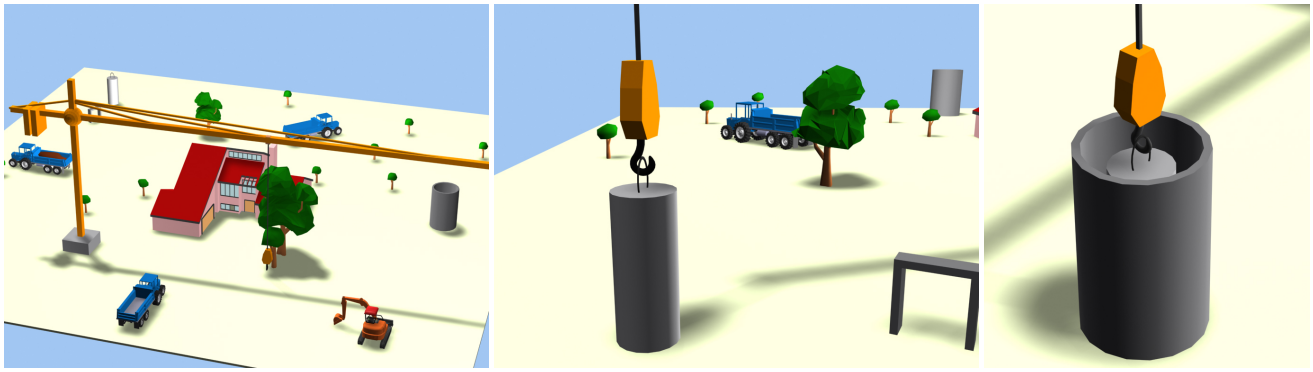
spatial and temporal culling are quite high. In particular, our culling technique drastically reduces the number of PCLs in the benchmark of four colliding trains thanks to the tight dynamic BVH using Taylor models.

Benchmark	Spatial Culling			Temporal Culling	
	Before	After	Rate	After	Rate
Walk	229	8.4	96.3%	0.11	98.7%
Exercise	81	3.0	96.3%	0.086	97.1%
Four Trains	179574	34.5	99.9%	3.3	90.4%
Falling Train	137	18.2	86.7%	11.3	37.9%

Table 2: Culling Efficiency. *Each row from top to bottom represents the culling results of the walking mannequin, exercising mannequin, colliding four trains, and falling train benchmarks, respectively. Each column from left to right represents benchmarking types, the number of PCLs before spatial culling (i.e., dynamic BVH culling), the number of PCLs after spatial culling, the culling ratio of spatial culling, the number of PCLs after temporal culling, and the culling ratio of temporal culling (1-(fifth column divided by third column)), respectively.*

8.2 Comparisons with Other Algorithms

[Schwarzer et al. 2002] performs CCD in a similar way to our algorithm using motion bounds and distance calculations. However, our motion bounds are tighter than theirs since our algorithm uses motion projection for CA. Moreover, our algorithm uses CA to advance the iteration time step adaptively to find τ whereas [Schwarzer et al. 2002] relies on a heuristic bisection search. Finally, our motion bound calculation is quite general so that it can handle any type of joints, whereas the technique of [Schwarzer et al. 2002] is only designed for prismatic and revolute joints. Thus, ours is more suitable for computer graphics applications. [Redon et al. 2004b] presents another CCD algorithm for articulated models. However, their dynamic BVH is less tight than ours since it relies on standard interval arithmetic. Moreover, since their algorithm is based on an explicit formulation of swept volume, its runtime cost can be high when an articulated model has many links. Finally, the distance calculation technique [Ehmann and Lin 2001] used in our CCD algorithm can utilize the connectivity information in polyhedra and motion coherence in iterative CA operations whereas theirs cannot. However, both [Schwarzer et al. 2002; Redon et al. 2004b] can handle polygon soup models, but our current implementation based on [Ehmann and Lin 2001] cannot handle such models, although in principle it could by using a distance calculation algorithm such as [Larsen et al. 1999]. In Fig. 11, we show the performance comparison of our algorithm against that of [Redon et al. 2004b] and observe 15$\sim$17 times performance improvement.



(a) (b) (c)

Figure 9: Construction Site Benchmark 2 (a) *Another construction site scene.* **(b)** *After moving from the first site to the second, a tower crane picks up a weight and drops it into a pipe as shown in* **(c)**. *The whole construction site consists of 0.394M triangles besides a moving tower crane which consists of 1,288 triangles and 272 convex pieces.*

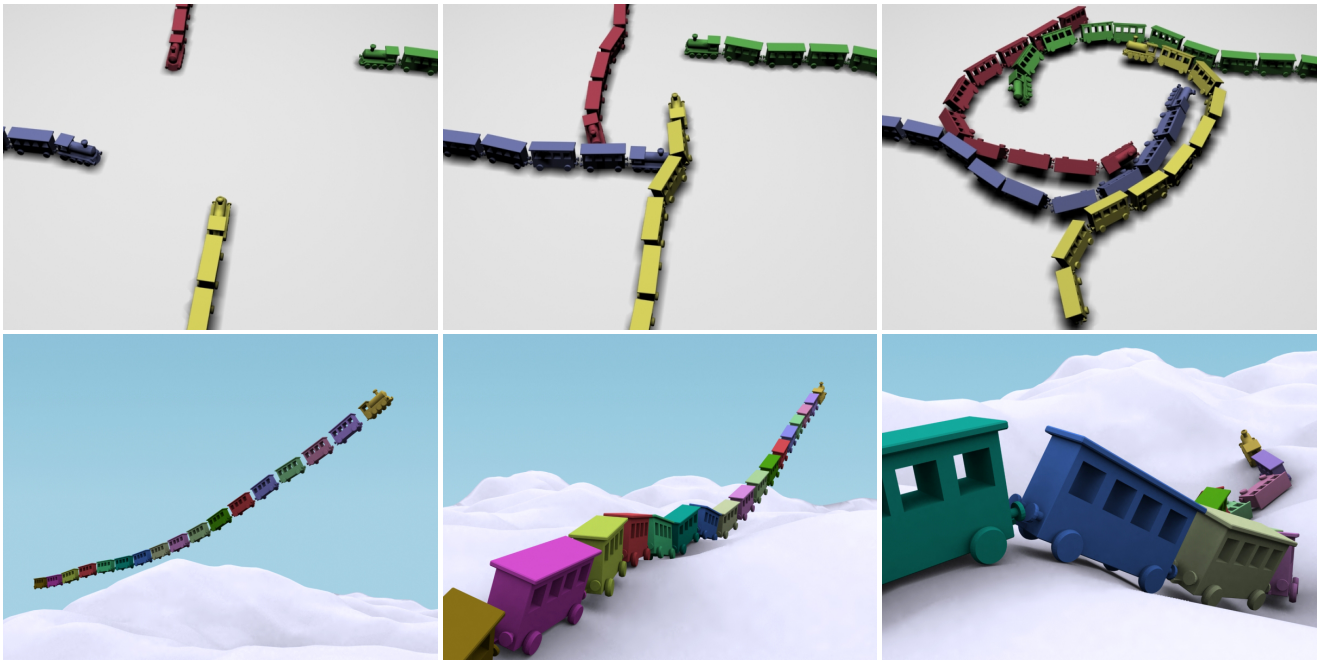


Figure 10: Collision Course Benchmarks. Top *Four trains consisting of 10 links and 23K triangles each are collided with one another. The trains have 17,444 convex pieces in total.* **Bottom** *A train consisting of 17 links and 42K triangles drops on a mountain model consisting of 29K triangles. The train and mountain have 27,931 and 21,437 convex pieces, respectively.*

9 Conclusions and Future Work

We have presented a novel continuous collision detection algorithm for articulated models. In essence, it consists of three steps: pre-processed bounding-volume hierarchy construction, dynamic BVH culling based on Taylor models, and conservative advancement using temporal culling. Our algorithm is very fast, and we have applied it to different applications such as articulated-body dynamics and motion planning for virtual characters.

Limitations Our current implementation requires 2-manifold, polyhedral models, even though the main idea is also applicable to *polygonal soup* models. Moreover, the bottleneck in our algorithm is the distance calculation based on convex decomposition.

Future work There are many avenues for future work. Since our algorithm is fast, a rapid prototyping application based on user interaction, like [Redon et al. 2002], is readily implementable. Our algorithm is also suitable for 6-DOF haptics, which also requires fast update rates. We would like to investigate the possibility of extending our algorithm to deformable models such as cloth or hair. Another possible future work may be to extend our algorithm to handle articulated models with kinematics loops.

Acknowledgements

This work was supported in part by the grant 2004-205-D00168 of KRF, the STAR program of MOST and the ITRC program. Stephane Redon was partially funded by a PAI STAR grant. We would like to thank the anonymous reviewers for their valuable comments.

References

ABDEL-MALEK, K., BLACKMORE, D., AND JOY, K. 2002. Swept volumes: foundations, perspectives, and applications. *International Journal of Shape Modeling*.

AGARWAL, P. K., BASCH, J., GUIBAS, L. J., HERSHBERGER, J., AND ZHANG, L. 2001. Deformable free space tiling for kinetic collision detection. In *Workshop on Algorithmic Foundations of Robotics*, 83–96.

AGEIA. 2006. *PhysX*. http://www.ageia.com.

BARAFF, D., AND WITKIN, A. 2001. *Physically-based modeling*. ACM SIGGRAPH Course Notes.

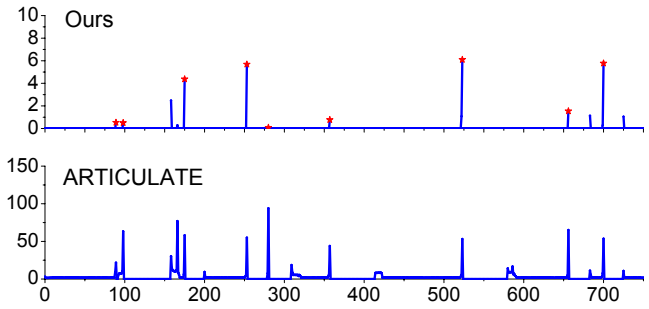


Figure 11: Performance Comparison against ARTICULATE: *Using the walking mannequin benchmark (Fig. 7(a)), we compare the performance of our algorithm* **(Top)** *against that of ARTICULATE [Redon et al. 2004b]* **(Bottom)**. *For this comparison, we disable self-collision since ARTICULATE do not implement such a case. The average CCD timing for ours is 0.26 ms whereas ARTICULATE takes 3.92 ms. The average TOC computation for ours is 3.1 ms and that of ARTICULATE is 56.44 ms. We observe 15∼17 times performance improvement of ours over ARTICULATE.*

BAREQUET, G., AND HAR-PELED, S. 2001. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *J. Algorithms 38*, 91–109.

BERZ, B., AND HOFSTÄTTER, G. 1998. Computation and application of Taylor polynomials with interval remainder bounds. *Reliable Computing 4*, 83.

BÜHLER, K., DYLLONG, E., AND LUTHER, W. 2004. Reliable distance and intersection computation using finite precision geometry. *Lecture Notes in Computer Science 2991*, 160–190.

CANNY, J. F. 1986. Collision detection for moving polyhedra. *IEEE Trans. PAMI 8*, 200–209.

CHOI, Y.-K., WANG, W., LIU, Y., AND KIM, M.-S. 2006. Continuous collision detection for elliptic disks. *IEEE Transactions on Robotics 22*, 2.

COMBA, J. L. D., AND STOLFI, J. 1993. Affine arithmetic and its applications to computer graphics. In *Proceedings of the VI Sibgrapi. Recife, Brazil*.

COUMANS, E. 2006. *Bullet Physics library*. http://www.continuousphysics.com.

CRAIG, J. J. 1989. *Introduction to Robotics: Mechanics and Control*. Prentice Hall.

EHMANN, S., AND LIN, M. C. 2001. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001) 20*, 3, 500–510.

GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 1996. OBBTree: A hierarchical structure for rapid interference detection. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, 171–180.

HAVOK. 2006. *Havok Physics*. http://www.havok.com.

HUBBARD, P. M. 1993. Space-time bounds for collision detection. Technical report cs-93-04, Dept. of Computer Science, Brown University, Feb.

KIM, B., AND ROSSIGNAC, J. 2003. Collision prediction for polyhedra under screw motions. In *ACM Conference on Solid Modeling and Applications*.

KIM, D., GUIBAS, L., AND SHIN, S. 1998. Fast collision detection among multiple moving spheres. *IEEE Trans. Vis. Comput. Graph. 4*, 3, 230–242.

KIRKPATRICK, D., SNOEYINK, J., AND SPECKMANN, B. 2000. Kinetic collision detection for simple polygons. In *Proc. of ACM Symposium on Computational Geometry*, 322–330.

LARSEN, E., GOTTSCHALK, S., LIN, M., AND MANOCHA, D. 1999. Fast proximity queries with swept sphere volumes. Tech. Rep. TR99-018, Department of Computer Science, University of North Carolina.

LIN, M., AND MANOCHA, D. 2003. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*.

LIN, M. C. 1993. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California, Berkeley, CA.

MIRTICH, B. V. 1996. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley.

MIRTICH, B. 2000. Timewarp rigid body simulation. *SIGGRAPH Conference Proceedings*, 193–200.

MOORE, R. E. 1979. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia. ISBN 0-89871-161-4.

MPK-TEAM. 2006. *Motion Planning Kit*. http://ai.stanford.edu/∼mitul/mpk/.

NEUMAIER, A. 1993. The wrapping effect, ellipsoid arithmetic, stability and confidence regions. *Computing Supplementum 9*, 175–190.

ORTEGA, M., REDON, S., AND COQUILLART, S. 2006. A six degree-of-freedom god-object method for haptic display of rigid bodies. In *IEEE International Conference on Virtual Reality*.

REDON, S., AND LIN, M. C. 2005. Practical local planning in the contact space. In *Proceedings of IEEE International Conference on Robotics and Automation*, 4200–4205.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2000. An algebraic solution to the problem of collision detection for rigid polyhedral objects. *Proc. of IEEE Conference on Robotics and Automation*.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2002. Fast continuous collision detection between rigid bodies. *Proc. of Eurographics (Computer Graphics Forum)*.

REDON, S., KIM, Y. J., LIN, M. C., AND MANOCHA, D. 2004. Interactive and continuous collision detection for avatars in virtual environments. In *Proceedings of IEEE Virtual Reality*.

REDON, S., KIM, Y. J., LIN, M. C., AND MANOCHA, D. 2004. Fast continuous collision detection for articulated models. In *Proceedings of ACM Symposium on Solid Modeling and Applications*.

SCHWARZER, F., SAHA, M., AND LATOMBE, J.-C. 2002. Exact collision checking of robot paths. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*.

SPONG, M. W., HUTCHINSON, S., AND VIDYASAGAR, M. 2005. *Robot Modeling and Control*. Wiley.

VAN DEN BERGEN, G. 2004. Ray casting against general convex objects with application to continuous collision detection. *Journal of Graphics Tools*.

ZHANG, X. Y., AND KIM, Y. J. 2007. Motion interpolation and bounds calculations in CATCH. Tech. Rep. CSE–TR-2007-01, Ewha Womans University, Seoul, Korea.

ZHANG, X. Y., LEE, M. K., AND KIM, Y. J. 2006. Interactive continuous collision detection for rigid non-convex polyhedra. *The Visual Computer (Proceedings of Pacific Graphics 2006) 22*, 9-11.