ACM Symposium on Solid Modeling and Applications (2004)
G. Elber, N. Patrikalakis, P. Brunet (Editors)

Fast Continuous Collision Detection for Articulated Models

Stephane Redon[1], Young J. Kim[2], Ming C. Lin[1] and Dinesh Manocha[1]

[1]Department of Computer Science, UNC at Chapel Hill, U.S.A., *{redon,lin,dm}@cs.unc.edu*
[2]Computer Science and Engineering, Ewha Womans University, Korea, *kimy@ewha.ac.kr*

http://gamma.cs.unc.edu/Articulate

Abstract

We present a novel algorithm to perform continuous collision detection *for articulated models. Given two discrete configurations of the links of an articulated model, we use an "arbitrary in-between motion" to interpolate its motion between two successive time steps and check the resulting trajectory for collisions. Our approach uses a three-stage pipeline: (1) dynamic bounding-volume hierarchy (D-BVH) culling based on interval arithmetic; (2) culling refinement using the swept volume of line swept sphere (LSS) and graphics hardware accelerated queries; (3) exact contact computation using OBB-trees and continuous collision detection between triangular primitives. The overall algorithm computes the time of collision, contact locations and prevents any interpenetration between the articulated model with the environment. We have implemented the algorithm and tested its performance on a 2.4 GHz Pentium PC with 1 Gbyte of RAM and a NVIDIA GeForce FX 5800 graphics card. In practice, our algorithm is able to perform accurate and continuous collision detection between articulated models and complex environments at nearly interactive rates.*

1. Introduction

Collision detection (CD) is a fundamental geometric problem that arises in diverse geometric applications like CAD/CAM, dynamic simulation, robotics and automation, haptics, virtual environments, computer games, etc. Given its importance, it has been extensively studied in these areas.

Most of the work in CD has focused on *discrete* algorithms, which check for interferences at fixed time instants only. In such cases, it is possible to miss a collision between two successive instances. Such situations can arise in different applications, e.g. in dynamics simulation [BW01] when a fast moving object pokes through small or thin objects in the environment. A typical remedy to handle these problems is to reduce the size of the time interval and increase the collision checking rates. However, it can not guarantee a collision free path between the two sampled instances. Another application is probabilistic roadmap methods (PRM) for robot motion planning [SSL02]. In these algorithms, the position of a robot is specified by its configuration. A key step in the PRM computation is checking whether there exists any collision-free path between two nearby configurations. Finally, in virtual reality applications the position of a user is tracked using external sensors and only measured at discrete time instants [Bur96]. It is important to check for any collisions between the avatar model and the virtual environment between successive time instants.

To overcome the limitations of discrete collision detection algorithms, techniques have been proposed that model the motion between successive instances as a continuous path and check the path for collisions with the environment. These are classified as *continuous collision detection* (CCD) algorithms [Can86, RKC00, RKC02, KR03]. A major issue in the design of such algorithms is modeling the continuous motion between the two successive positions and orientations of the object. It is important that the motion formulation is generic enough to interpolate any two given instances of an object, as well as simple enough so that it can be frequently and efficiently evaluated by the underlying CCD algorithm. Furthermore, it is relatively more expensive to check for collisions along a continuous path as opposed to a discrete instance.

Most of the prior work in CCD has been limited to rigid objects [Can86, GLGT99, RKC00, RKC02, KR03]. In this paper, we mainly focus on articulated models, including robot arms, kinematic or molecular chains that are composed of multiple links and are frequently used in robotics, CAD/CAM, protein modeling or other simulated environments. Some of the major challenges in the design of a CCD algorithm for articulated models include:

1. The complexity of the problem of generating a continuous motion and evaluating it increases with the number of links in the articulated model. Furthermore, we need to ensure that the generated motion should not allow any interpenetration between different links.
2. Some of the commonly known techniques to accelerate collision detection use bounding-volume hierarchies (BVHs). These hier-

archies are typically precomputed and such techniques are not directly applicable to articulated models with multiple moving links.

3. A classic approach to check for collisions for a continuously moving object is to calculate the swept volume (SV) of the object along the trajectory and test collisions between the calculated SV and the rest of the objects in the environment. However, the computation of the SV of an articulated model is quite costly and no efficient or robust algorithms are known for exact computation.

Main results: We present a novel algorithm to perform continuous collision detection (CCD) for articulated models in a virtual prototyping environment. Our algorithm accurately computes the time of collision and the contact locations and prevent any interpenetration of the articulated model with the environment. Since the actual object's motion is not known, we use an "arbitrary in-between" motion to interpolate between successive configurations of the articulated model. This motion formulation is used to check for collisions with the environment, as well as computing the contact location of the links of the articulated model at the time of collision.

Our approach uses a three-stage pipeline. In the first step, we use interval arithmetic to dynamically compute a bounding-volume hierarchy that encloses the links of the articulated model as well as the volume swept by them. The hierarchy is used to cull away links that are not in close proximity to the environment. The second stage refines the culling by performing dynamic collision detection between the environment and the volumes swept by the line swept spheres (LSS) that enclose the links. We use graphics hardware to perform fast collision detection between the swept volume of the LSS and the environment. Finally, in the third stage, we compute the exact contact positions and the time of collision between the articulated model and the environment. It performs geometric culling using OBB-trees and performs continuous collision detection between triangular primitives. We have implemented the algorithm and tested its performance on a 2.4 GHz Pentium PC with 1 Gbyte of RAM and a NVIDIA GeForce FX 5800 graphics card. In practice, our algorithm is able to perform accurate and continuous collision detection between articulated models and an environment consisting of tens of thousands of triangles at nearly interactive rates, as shown in Fig. 1.

Organization: The organization of the rest of the paper is as follows. In Section 2, we briefly review the prior work on CCD, discrete CD methods used in dynamic simulation and path planning, and various acceleration techniques. We give an overview of our approach in Section 3. Section 4 presents the first two stages of our algorithm that localize the collision computation, and Section 5 describes the algorithm for exact collision detection, including computation of the time of collision. In Section 6, we describe its implementation and highlight its performance on complex benchmarks. We analyze its performance in Section 7 and highlight some of its limitations.

2. Prior Work

Most of the prior work on CD has focused on checking for collisions at discrete time instances (please refer to [LM03] for a recent survey). This includes specialized algorithms for convex polytopes that exploit coherence between successive time steps and algorithms for general polygonal or spline models. The latter can be further classified based on whether they involve any preprocessing or not. In this section, we give a brief survey of the earlier work on continuous collision detection, pseudo-continuous collision detection methods such as backtracking, and acceleration techniques for collision detection.

Continuous Collision Detection: A few algorithms have been proposed for continuous collision detection (CCD) between a rigid object and the simulated environment. These algorithms model the trajectory of the object between successive discrete time instances and check the resulting path for collisions. More specifically, there are four different approaches presented in the literature: algebraic equation solving approach [Can86, RKC00], swept volume (SV) approach [AMBJ02], kinetic data structures (KDS) approach [KSS00], and adaptive subdivision approach [RKC02, SSL02, KR03].

The algebraic equation solving approach attempts to solve the CCD problem by explicitly solving the underlying CCD equations. The SV-based approach is based on calculating the SV of moving objects explicitly and checking for collisions between the SV and the rest of the environment. The KDS approach is a kind of scheduling scheme that is based on the usage of certificates, which tell us when a collision might occur. The adaptive subdivision approach employs a conservative separation test which ensures complete separation between some time intervals, and it selectively subdivides the time interval that fails the test until the subdivided interval becomes smaller than tolerance along the time dimension.

Dynamics Simulation and Local Motion Planning: There are many applications that require continuous checking of collisions or contacts between moving objects. These include motion planning based on probabilistic roadmap methods (PRM) [KSLO96] and constraint-based dynamics simulation [BW01]. The PRM computes a plausible path by sampling a number of configurations in the free space and building a roadmap by connecting the free configurations. As the roadmap is constructed, the PRM algorithm needs to check locally whether there exists a collision-free, continuous path between two configurations in the free space [SSL02]. A sequence of local planning steps are used to compute a global path from the initial configuration to the goal configuration. However, earlier work in PRM has been limited to finding a collision-free, continuous path for a limited class of articulated models with rotational or prismatic joints. Moreover, when a collision is found, these algorithms are unable to compute whether it is the first time of contact along a given trajectory. The estimation of time of collision is particularly important for dynamics simulation since objects are not allowed to interpenetrate but must reach contacting states.

Acceleration methods using Bounding Volume Hierarchies: In order to accelerate the performance of CD algorithms, culling techniques based on bounding volume hierarchies (BVHs) have been proposed for general polygonal models. Essentially, these techniques precompute a BVH for each rigid model and traverse the hierarchies at runtime to localize the region of potential intersection. BVHs can be classified based on the underlying bounding volume or traversal schemes. These include OBB trees [GLM96], sphere trees [Hub95], k-dop trees [KHM*98], and convex hull-based trees which use surface-based convex decomposition [EL01]. Algorithms based on hierarchies that utilize the topology of kinematic chains have been proposed for articulated models [LSHL02].

Acceleration methods using Graphics Hardware: Interpolation-

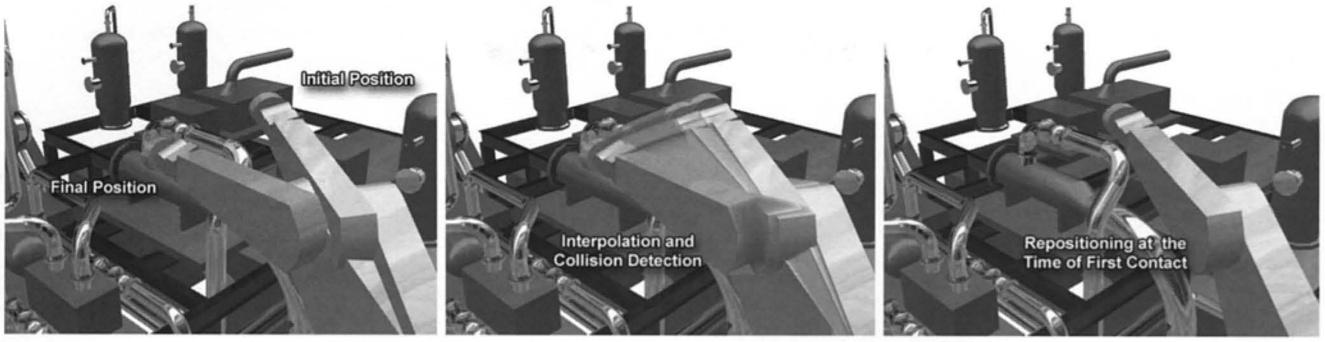
Initial Position
Final Position
Interpolation and Collision Detection
Repositioning at the Time of First Contact

Figure 1: *Benefits of our continuous collision detection algorithm over discrete methods. These images of our benchmarking system highlight the collision detection between a 6-dof robot in a CAD environment composed of pipes. The left image shows two discrete positions of the robot. The middle image illustrates the motion trajectory used by our algorithm. The right image indicates the position of the robot arm at the time of first contact with the CAD environment along that trajectory. These computations are performed at almost interactive rates.*

based graphics hardware is increasingly being used for geometric applications. This is mainly due to the recent advances in the performance of the graphics processors as well as the increased support for programmability. In particular, many algorithms based on graphics hardware have been proposed to perform interference or collision queries [RMS92, HZLM01, GRLM03]. Unlike BVH based algorithms, these techniques involve no preprocessing and are directly applicable to rigid as well as deformable models. However, the accuracy of these algorithms is governed by the image-space precision of the graphics hardware.

3. Overview

In this section, we give an overview of our approach to perform continuous collision detection between a moving articulated model and its surrounding environment. We first describe how we model the continuous motion for an articulated model using an *arbitrary in-between* motion [RKC00, RKC02], and highlight the complexity of explicitly checking for collisions. Next, we give an overview of our algorithm which proceeds in three stages.

3.1. Notation

We begin this section by explaining the notation used throughout the paper. In the following section, we describe the representation for an articulated chain that we use in the paper.

We use a bold-faced letter to distinguish a vector from a scalar value (e.g., a vector for a rotation axis $\mathbf{u}$). Let $\mathbf{u}_i^*$ denote the 3×3 matrix such as $\mathbf{u}_i^*\mathbf{x} = \mathbf{u}_i \times \mathbf{x}$ for every three-dimensional vector $\mathbf{x}$. If $\mathbf{u}_i = (u_i^x, u_i^y, u_i^z)^T$, then:

$$\mathbf{u}_i^* = \begin{pmatrix} 0 & -u_i^z & u_i^y \\ u_i^z & 0 & -u_i^x \\ -u_i^y & u_i^x & 0 \end{pmatrix} \quad (1)$$

3.2. Articulated Model

We represent an articulated model $\mathcal{A}$ made of p rigid links $\mathcal{A}_1, \ldots, \mathcal{A}_p$. We use a directed acyclic graph (DAG) to represent the articulated chain in the model. Each vertex in the graph represents a link $\mathcal{A}_i$ and an edge between $\mathcal{A}_i$ and $\mathcal{A}_j$ is connected if $\mathcal{A}_i$ and $\mathcal{A}_j$ are connected by a joint. We allow both translation and rotation for each joint. However, we assume that there is no kinematic loop in the graph describing the articulated chain, i.e., there is no cycle in the graph. Consequently, each link $\mathcal{A}_i$ has a unique parent link, except for the root link which has no parent. On the other hand, any link can have any number of children, as long as there is no loop induced. For the sake of simplicity of notation, we assume that the index of link i's parent is $i-1$. This can be easily modified when a parent has multiple children per link.

For a given link i, let $\mathcal{P}_i$ denote the reference frame associated with it. Let us further represent the orientation of $\mathcal{P}_i$ relatively to $\mathcal{P}_{i-1}$ as $\mathbf{P}_i^{i-1}$. Similarly, the motion of $\mathcal{P}_i$ relatively to $\mathcal{P}_{i-1}$ at time t is described by $\mathbf{M}_i^{i-1}(t)$. The time interval of t is normalized to $[0,1]$. Figure 2.(a) illustrates our notation for a link i moving within the reference frame of its parent.

3.3. Motion Formulation

As is the case in many applications, the actual motion of the moving articulated model is not known a priori and we are only given its positions and orientations at discrete time instances. For example, when the model is part of a constraint-based multi-body dynamics simulation system, the system's dynamics is solved using discretized techniques (e.g. Euler or Runge-Kutta methods). As a result, we do not have a closed-form expression of model's motion.

Given these constraints, we *arbitrarily* choose a motion formulation to interpolate between different model configurations. The goal is to use a formulation that is general enough to interpolate between any two successive configurations and preserves the rigidity of the links in the articulation. Moreover, it needs to be simple enough to allow us to perform the various steps of our collision detection algorithm.

We first begin by expressing the motion of each link in the reference frame of its unique parent. The motion of the root link is similarly expressed in the global frame.

Let's now describe the motion of $\mathcal{P}_i$ relatively to $\mathcal{P}_{i-1}$. We use the 3-dimensional vector $\mathbf{c}_i$ and the 3×3 matrix $\mathbf{R}_i$ to denote the position and orientation of $\mathcal{P}_i$ relatively to $\mathcal{P}_{i-1}$ at the beginning of the time interval $[0,1]$, respectively. We assume that the motion of

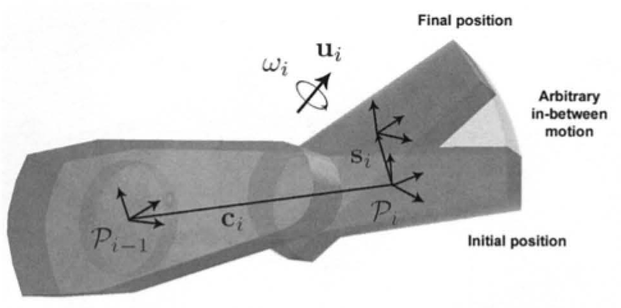

Figure 2: *Link i is moving in the reference frame of its parent. The initial and final positions of the link as well as the motion trajectory have been outlined.*

$\mathcal{P}_i$ relatively to $\mathcal{P}_{i-1}$ is composed of a rotation of angle ω_i around an axis $\mathbf{u}_i$, and of a translation $\mathbf{s}_i$. The parameters $\mathbf{c}_i$, $\mathbf{R}_i$ are determined by the relative configuration of $\mathcal{P}_i$ with respect to $\mathcal{P}_{i-1}$, and $\mathbf{u}_i$, $\mathbf{s}_i$ by the relative motion of $\mathcal{P}_i$ with respect to $\mathcal{P}_{i-1}$. Thus, for a given time step, $\mathbf{c}_i$, $\mathbf{R}_i$, $\mathbf{u}_i$ and $\mathbf{s}_i$ are constants and are expressed in terms of in $\mathcal{P}_{i-1}$. Moreover, we assume that $\mathcal{P}_i$ moves with constant translational and rotational velocities.

The position of $\mathcal{P}_i$ relatively to $\mathcal{P}_{i-1}$ for a given time t in $[0,1]$ is thus:

$$\mathbf{T}_i^{i-1}(t) = \mathbf{c}_i + t\mathbf{s}_i, \tag{2}$$

The orientation of $\mathcal{P}_i$ relatively to $\mathcal{P}_{i-1}$ is given as:

$$\mathbf{P}_i^{i-1}(t) = \cos(\omega_i t).\mathbf{A}_i + \sin(\omega_i t).\mathbf{B}_i + \mathbf{C}_i, \tag{3}$$

where $\mathbf{A}_i$, $\mathbf{B}_i$ and $\mathbf{C}_i$ are 3×3 constant matrices which are computed at the beginning of the time step:

$$\begin{aligned} \mathbf{A}_i &= \mathbf{R}_i - \mathbf{u}_i.\mathbf{u}_i^T.\mathbf{R}_i \\ \mathbf{B}_i &= \mathbf{u}_i^*.\mathbf{R}_i \\ \mathbf{C}_i &= \mathbf{u}_i.\mathbf{u}_i^T.\mathbf{R}_i \end{aligned} \tag{4}$$

Consequently, the motion of $\mathcal{P}_i$ relatively to $\mathcal{P}_{i-1}$ is described by the following 4×4 homogeneous matrix:

$$\mathbf{M}_i^{i-1}(t) = \begin{pmatrix} \mathbf{P}_i^{i-1}(t) & \mathbf{T}_i^{i-1}(t) \\ (0,0,0) & 1 \end{pmatrix}, \tag{5}$$

in the reference frame of the parent link $\mathcal{P}_{i-1}$. Finally, the matrix:

$$\mathbf{M}_i^0(t) = \mathbf{M}_1^0(t).\mathbf{M}_2^1(t)...\mathbf{M}_i^{i-1}(t) \tag{6}$$

describes the motion of link i in the world coordinate system.

Note that this formulation makes it extremely simple to compute all the motion parameters $\mathbf{s}_i$, $\mathbf{u}_i$ and ω_i for a given timestep. For a given link i, assume that $\mathbf{c}_i^0$ and $\mathbf{c}_i^1$ (resp. $\mathbf{R}_i^0$ and $\mathbf{R}_i^1$) are the initial and final positions (resp. orientations) of $\mathcal{P}_i$ relatively to $\mathcal{P}_{i-1}$. Then $\mathbf{s}_i = \mathbf{c}_i^1 - \mathbf{c}_i^0$, and $(\mathbf{u}_i, \omega_i)$ is the rotation extracted from the rotation matrix $\mathbf{R}_i^1(\mathbf{R}_i^0)^T$.

Using the continuous motion $\mathbf{M}_i^0(t)$ for each link i, our goal is to check for collisions between all $\mathcal{A}_i$'s following the motion in the articulated model and the other objects in the environment, and, if there is any collision, to report the first time of contact. Mathematically, we want to know whether the set in Eq. 7 is non-empty:

$$\{\, t \in [0,1] \mid \mathbf{M}_i^0(t)\mathcal{A}_i \cap \mathcal{O} \neq \emptyset, i = 1,\ldots,p\}. \tag{7}$$

Furthermore, we want to compute the smallest element t_c of this set. Here, p is the number of links in the articulated model and $\mathcal{O}$ represents all the objects in the environment.

3.4. Complexity of Continuous Collision Detection

An obvious approach to perform exact CCD is to compute the swept volume (SV) of a moving object and check the generated SV against the environment. However, an exact calculation of SV is very challenging even for a single rigid object because SV computation requires arrangement or envelope computation. The computational and combinatorial complexity of arrangement can be super-quadratic in the number of primitives and its robust implementation is also non-trivial. Some approximation algorithms have been proposed for SV computation [KVLM03, RK00]. However, these algorithms can take a few minutes for a single rigid object and can not be directly used for interactive collision detection.

The SV problem becomes even more complicated when we need to deal with sweeping articulated models because multiple-parameter sweeping needs to be considered [AMO99]. The multiple-parameter sweeping involves performing consecutive sweeping for each joint parameter in an articulated model. The major difficulty of multiple-parameter sweeping lies in the mathematical complexity of its formulation and representation of sweeping. In addition to checking for collisions, we also want to compute the time of collision (TOC). Therefore, computing the SV in a three-dimensional space is not enough and we need to add the time dimension to the underlying SV formulation [Cam90].

3.5. Our Approach

Due to the aforementioned challenges in performing exact CCD, we present an approximate and fast solution to the problem. The main idea of our approach is as follows. As a preprocess, we build a static BVH of the given articulated model using a line-swept sphere (LSS) as the bounding volume (BV). At runtime, we dynamically build a BVH of the articulated model using arbitrary in-between motion formulation by applying interval arithmetic (IA) to the SV of each leaf node (i.e., LSS) in the static BVH and recursively building the entire hierarchy in a bottom-up fashion. Using the dynamic BVH, we localize the contact geometry that is likely to collide with the environment. Once we localize the contact geometry, we compute the earliest time of collision for each triangle contained in localized geometric primitives.

The entire pipeline of our algorithm consists of three stages. We can group the first two stages as *contact localization* and the third stage as *exact contact computation*. Overall, the pipeline, also shown in Fig. 3, is:

1. Dynamic BVH Culling:
   a. Given two successive configurations of the articulated model, we compute an arbitrary in-between path from the initial to the final configuration.
   b. Using the continuous path for each link, we use interval arithmetic to compute an enclosing axis-aligned bounding box (AABB) and recursively construct an AABB hierarchy around the entire model. This hierarchy is used to cull away the links which are not in close proximity to the environment.
2. Dynamic SV Culling:

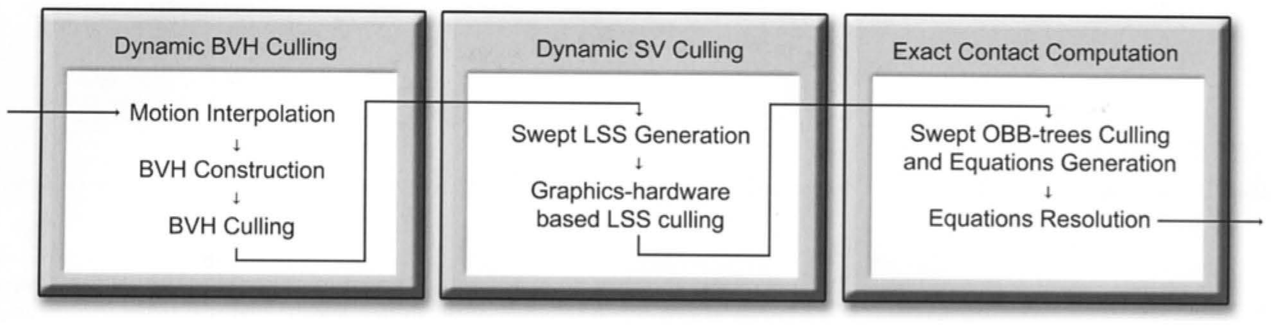

Figure 3: *The overall pipeline of our continuous collision detection algorithm. Different stages are performed on the CPU and the graphics processor.*

   a. For the remaining links, we approximate the volume swept by the precomputed bounding LSS by tessellating the offset of a ruled surface within some error deviation. The ruling line in the ruled surface corresponds to the skeleton of the LSS.
   b. We use graphics hardware to check whether the approximate SV collides with objects in the environment.

3. Exact Contact Computation:

   a. As a precomputation, for each link, we build an OBB-tree which encloses the actual geometry of the link. At runtime, for each potentially colliding link that is the output of previous stage in the pipeline, we further cull away portions of the actual link geometry by using a novel continuous OBB overlap test based on interval arithmetic. For the remaining portions of geometry (i.e., triangles), we generate a list of equations that provide the exact time of contact (TOC) of the geometry against the objects in the environment.
   b. The computation of the time of contact is performed using a combination of interval arithmetic and subdivision method. We present a new algorithm for articulated models that simultaneously solves the equation corresponding to elementary continuous collision detection tests. For a given time interval, all the relevant collision equations are solved and the interval is further subdivided if necessary.

4. Contact Localization

In this section, we describe the first and second stages in the pipeline.

4.1. Dynamic BVH Culling

Given the motion formulation, the first step in the collision detection algorithm is to compute a BVH around the swept volume of the articulated model. Each bounding volume (BV) in the BVH is an AABB. We compute an AABB for each link, that encloses its swept volume over the entire trajectory during a given time step. These leaf-boxes are then used to efficiently compute a complete hierarchy of AABB's used to quickly cull away links which are far from the environment.

The leaf-boxes are computed using *interval arithmetic* (IA) [Moo79]. We bound each component of the orientation matrices $\mathbf{P}_i^{i-1}(t)$ over the entire time interval $[0,1]$ using elementary interval operations. Similarly, we use elementary interval operations to bound the translation components $\mathbf{T}_i^{i-1}(t)$. Eventually, we obtain 4×4 homogeneous interval matrices $\tilde{\mathbf{M}}_i^{i-1}$ whose interval components bound the corresponding components of $\mathbf{M}_i^{i-1}$ over the time interval $[0,1]$. These interval matrices are concatenated by again performing elementary interval operations to compute the interval version $\tilde{\mathbf{M}}_i^0$ of the matrix $\mathbf{M}_i^0$.

By applying this interval matrix to both $\mathbf{L}_i^a$ and $\mathbf{L}_i^b$, we obtain two 3-dimensional interval vectors that bound the coordinates of the endpoints of the links over the time interval $[0,1]$. In effect, these 3-dimensional interval vector are AABBs which enclose the endpoints over the time interval $[0,1]$. By using the convexity argument, it can be seen that the AABB that encloses these two boxes bounds the entire link over the time interval. Next we enlarge the box by an offset equal to the radius of the LSS to ensure that the AABB bounds the LSS and its entire trajectory. Given the AABBs around the leaf-nodes, we compute a complete AABB hierarchy in a bottom-up manner around the entire model. After computing the BVH, we recursively check for overlaps with the environment and conservatively cull away the links that are far from the environment.

4.2. Dynamic Swept Volume Culling

For the remaining links that are not culled away by the previous stage in the pipeline, we check whether their approximate SV is colliding with the environment. If it is colliding, we proceed to the next stage in the pipeline where more precise collision checking is performed.

4.2.1. Swept Volume of Line Swept Sphere

The approximate SV of each link is computed by calculating the SV of the LSS which encloses the link and tessellating it, as shown in Fig. 4. The LSS enclosing a link is precomputed using the algorithm proposed in [LGLM00]. For example, in Fig. 5, we show a puma robot model and its approximation using LSS. It is well known that the SV of an LSS is equivalent to the offset surface of a ruled surface, where the ruling line in the ruled surface corresponds to the axial line of LSS and the offset radius in the offset surface corresponds to the radius of LSS.

The mathematical formulations of a ruled surface, $x(t,s)$, and its

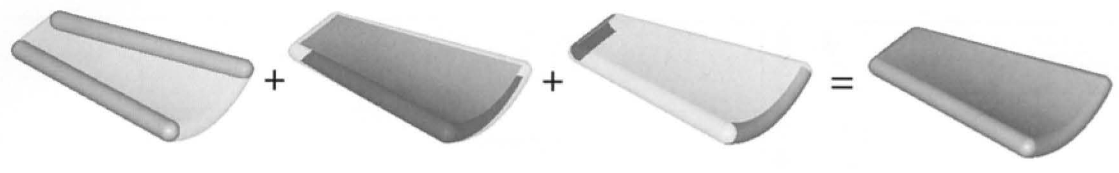

Figure 4: *The SV (rightmost image) of LSS consists of LSSs at initial and final configurations (leftmost image), ruled surface (center left) and pipe surface (center right).*

offset surface with offset radius d, $x_d(t,s)$, are given in Eq. 8 and 9, respectively:

$$x(t,s) = b(t) + s\delta(t) \tag{8}$$

$$x_d(t,s) = x(t,s) \pm d\, n(t,s) \tag{9}$$

Here, $b(t)$ is a directrix and $\delta(t)$ is the direction of a ruling line in the ruled surface, and $n(t,s)$ is the unit normal vector field defined on the surface of $x(t,s)$. Moreover, we assume that $x(t,s)$ is regular. In case $x(u,v)$ may contain non-regular points, conventional techniques to handle such cases bound $n(u,v)$ with a spherical polygon [PW01]. Also notice that, in Eq. 9, $x_d(u,v)$ is defined as a two-sided offset surface suited for our application.

Using the relationship between the offset of a ruled surface and the SV of the LSS, we compute the swept volume by independently computing the SV of the cap portion of LSS and computing the union with the remaining portion of LSS. The SV generated by the caps of LSS is a *pipe surface*. As a matter of fact, the pipe surface is a special case of a *canal surface*. A canal surface is generated by sweeping a sphere of varying radii along some continuous trajectory. A pipe surface is a special case of a *canal surface* where the radius is fixed. The parametric equation of a pipe surface is formulated as [KL03]:

$$K(t,\theta) = C(t) + R(\cos\theta b_1(t) + \sin\theta b_2(t)) \tag{10}$$

$$b_1(t) = \frac{C'(t) \times C''(t)}{\|C'(t) \times C''(t)\|}$$

$$b_2(t) = \frac{C'(t) \times b_1(t)}{\|C'(t) \times b_1(t)\|}$$

Here, $C(t)$ is the spline curve that a sphere of fixed radius sweeps along to generate a pipe surface $K(t,\theta)$. Once we have computed the offset of the ruled surface and the pipe surface, we compute the SV of the LSS by taking the union of them. This relationship is also illustrated in Fig. 4.

4.2.2. Tessellation of Swept Volume

Given the parametric representation of offset and pipe surfaces, there are two main challenges in performing collision detection using these surfaces. These include computing an accurate, explicit representation of the SV and checking it for interference with the environment. Since an exact, explicit representation of SV requires costly arrangement calculation and surface/surface intersections, we approximate the SV with piecewise triangular patches and do not perform the exact intersection or clipping computations. Moreover, we analyze the maximum deviation error from the exact surfaces. Since these patches are computed on the fly, we cannot use preprocessing techniques based on BVHs for fast collision checking. Rather, we use a graphics hardware accelerated interference checking algorithm that requires no preprocessing.

4.2.2.1. Uniform Tessellation The earlier algorithms for approximating an offset surface assume that the underlying progenitor surface is a free-form surface such as Bézier or NURBS surface. Under this assumption, there are three typical approaches to approximate an offset surface [ELK97]; control polygon-based, interpolation-based and circle approximation approach. In particular, the interpolation-based approach is based on directly sampling the positions and derivatives of the exact offset surface and attempts to optimize the approximated offset surfaces [Far86, Hos88, Kla83]. We adopt this technique in our application because of its simplicity which makes it better suited for interactive applications. In particular, we uniformly sample the offset of the ruled surface in the u and v parameter domain, as given in Eq. 9, and create strips of triangles by varying one of the parameters while fixing the other one. The tessellation of a pipe surface is performed using a similar approach. Given the formulation in Eq. 10, we uniformly sample the pipe surface along the t and θ parameters.

4.2.2.2. Tessellation Error The deviation error of an approximated offset surface is calculated by computing $\|x_d(t,s) - x(t,s)\| - d$ or squared distance $\|x_d(t,s) - x(t,s)\|^2 - d^2$ [ELK97]. The error is relatively easy to compute when the progenitor surface is represented as Bézier or NURBS surface. However, progenitor surface in our case is a non-rational surface and described using trigonometric functions. As a result, error calculation becomes non-trivial. In this case, there are two possibilities to calculate the error bound. Either we can use iterative numerical techniques like the Newton-Raphson method to derive the error bound, or if we can bound the derivatives of the progenitor surface, we can bound the deviation error as well. We use the second approach because we needed to calculate the derivatives as part of offset (Eq. 9) and pipe surface formulation (Eq. 10). The intervals (i.e., bounds) of the derivatives can be obtained by applying interval arithmetic similarly done as in Sec. 4.1.

Our method to derive an error bound is based on a well-known result in the approximation theory. The theorem by Filip et al. [FMM86] is stated as follows: Given a C^2 surface $f : [0,1] \times [0,1] \to \mathbb{R}^3$ and a tolerance ε, a piecewise linear surface $l : [0,1] \times [0,1] \to \mathbb{R}^3$ with n and m uniform subdivision along each $[0,1]$ satisfies $\sup \|f(t,s) - l(t,s)\| \leq \varepsilon$ when

$$\frac{1}{8}\left(\frac{1}{n^2}M_1 + \frac{2}{nm}M_2 + \frac{1}{m^2}M_3\right) = \varepsilon \tag{11}$$

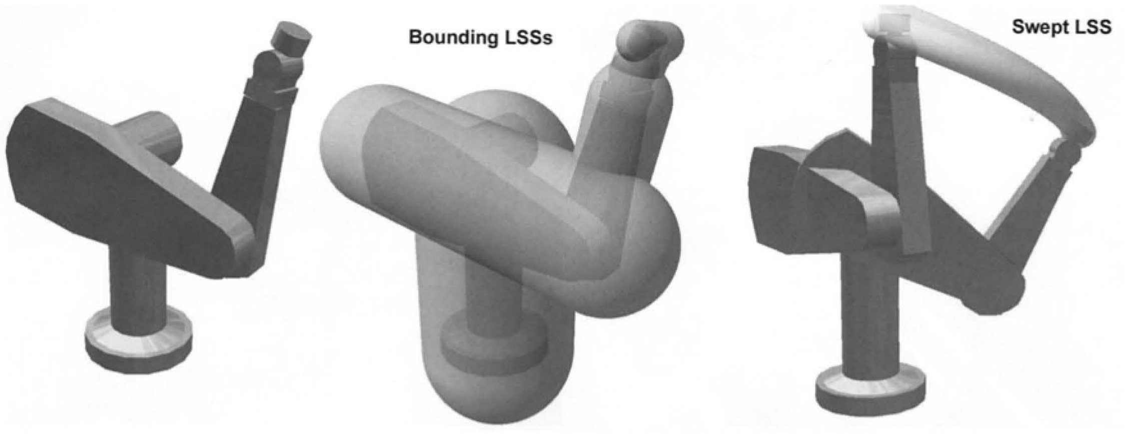

Figure 5: *The leftmost image shows an articulated 6-dof Puma robot model, the center image shows LSS' bounding the Puma robot, and the right image shows the SV of the LSS bounding the end-effector of the Puma robot.*

where

$$M_1 = \sup_{(t,s)\in[0,1]\times[0,1]} \left|\left|\frac{\partial^2 f(t,s)}{\partial u^2}\right|\right|$$

$$M_2 = \sup_{(t,s)\in[0,1]\times[0,1]} \left|\left|\frac{\partial^2 f(t,s)}{\partial u \partial v}\right|\right|$$

$$M_3 = \sup_{(t,s)\in[0,1]\times[0,1]} \left|\left|\frac{\partial^2 f(t,s)}{\partial v^2}\right|\right|$$

In our case, $f(t,s)$ corresponds to the offset surface $x_d(t,s)$ of a ruled surface $x(t,s)$ in Eq. 8 and 9. The relationship between the derivatives of $x_d(t,s)$ and $x(t,s)$ can be algebraically expressed [Far86]. Therefore, we first bound the derivatives of $x(t,s)$ using interval arithmetic, followed by bounding the derivatives of $x_d(t,s)$. As a result, given error tolerance ε, we can determine the required subdivision step sizes (i.e., n,m in Eq. 11) to tessellate the offset surface.

Similarly, we apply Eq. 11 to the parametric representation of a pipe surface (Eq. 10) and combine it with interval arithmetic, to compute the step sizes to tessellate the pipe surface.

4.2.3. Graphics Hardware-based Collision Detection

Once we have tessellated offset and pipe surfaces, we use the graphics processor to check for collisions. Since these tessellated surfaces are generated on the fly, we cannot use earlier CD techniques based on precomputed hierarchies to speed up collision queries. Instead, we choose the CULLIDE algorithm [GRLM03] that uses graphics hardware to perform interactive collision detection. The basic idea of CULLIDE is to pose the collision detection problem in terms of performing a sequence of visibility queries. If an object is classified as fully-visible with respect to the rest of the environment, it is a sufficient condition that the object does not overlap with the environment. For those objects that are classified as partially visible, the algorithm performs exact triangle-level intersection tests. CULLIDE performs the visibility queries using the graphics processors and the exact triangle-level intersection tests on the CPUs.

Precisely, we perform 2.5D overlap tests between the objects on the GPU by performing orthographic projections along the X, Y and Z directions. The graphics hardware is very well optimized to perform these transformations, scan converting the primitives and performing these pixel level comparisons by using the multiple pixel processing engines in parallel. In particular, we use the NVIDIA OpenGL extension GL_NV_occlusion_query[NVI03] to perform the visibility queries. This query is available on the commodity graphics processors.

5. Exact Contact Computation

The contact localization algorithm described in Section 4 is used to cull away some of the links that are not colliding with the environment. In this section, we present an algorithm for exact contact computation between the links and the objects in the environment. We also accurately compute the time of contact and the position of the links at those times. The exact contact computation algorithm proceeds in two parts. First we use hierarchies of oriented bounding boxes (OBBs) to perform inter-object culling. The second step involves performing continuous collision detection operations for triangular primitives. We present novel and improved algorithms for each step.

5.1. Geometry Culling based on OBB-trees

We use hierarchies of OBBs to perform the culling [GLM96]. Since the links in the articulated model and the objects in the environment are rigid, each OBB-tree is computed offline. We present an improved algorithm to perform continuous overlap tests between the OBBs over a given time interval.

Given two discrete positions of the OBBs, we check for overlap based on the separating axis test [GLM96]. Lets assume that the first OBB is described by three axes $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$, a center $\mathbf{T}_A$, and its half-sizes along its axes a_1, a_2 and a_3. Similarly, assume the second OBB is described by its axes $\mathbf{f}_1$, $\mathbf{f}_2$ and $\mathbf{f}_3$, its center $\mathbf{T}_B$, and its half-sizes along its axes b_1, b_2 and b_3. The separating axis theorem states that two static OBBs overlap if and only if all of fifteen separating axis tests fail. A separating test is simple: an axis $\mathbf{a}$ separates the OBBs if and only if:

$$|\mathbf{a}\cdot\mathbf{T}_A\mathbf{T}_B| > \sum_{i=1}^{3} a_i|\mathbf{a}\cdot\mathbf{e}_i| + \sum_{i=1}^{3} b_i|\mathbf{a}\cdot\mathbf{f}_i|. \quad (12)$$

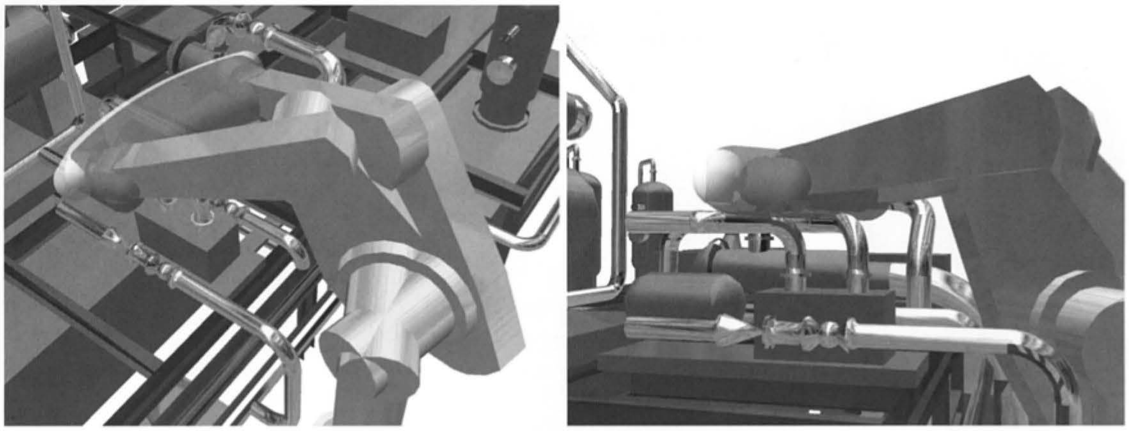

Figure 6: *Dynamic SV culling based on graphics hardware (i.e., CULLIDE) applied to the last two links of a Puma robot model. The volume swept by the LSS which bounds the last link does not collide with the environment, and thus the link is culled away. The dynamic SV culling allows us to perform an efficient culling even for very large motions.*

This test is performed for 15 axes at most [GLM96].

For continuous collision detection, it is necessary to perform continuous overlap tests between the bounding volumes. We use the continuous test proposed by Redon et al. in [RKC02], which extends the discrete OBB/OBB overlap test to the continuous domain using interval arithmetic. Since each member of inequality (12) is a function of time depending on the specific arbitrary in-between motion, interval arithmetic is used to bound both members very efficiently over a time interval $[t_n, t_{n+1}]$. As before, once the bounds, $\tilde{\mathbf{M}}_i^0(t)$, on the position matrices $\mathbf{M}_i^0(t)$ for the corresponding links have been obtained, the bounds $\tilde{\mathbf{v}}$ on the corresponding elements are computed by performing the interval matrix-vector multiplication $\tilde{\mathbf{v}} = \tilde{\mathbf{M}}_i^0(t)\mathbf{v}$. When the lower bound on the left member is larger than the upper bound on the right member, the axis $\mathbf{a}$ separates the boxes during the entire time interval $[t_n, t_{n+1}]$, and the pair of boxes is discarded.

However, this continuous overlap test can be quite conservative in practice. There are two main reasons:

- Two OBBs can be separated over $[t_n, t_{n+1}]$ even when there does not exist one axis which separates them on the whole time interval.
- Since the bounds are obtained by recursively performing interval arithmetic operations, the bounds are not tight. As a result, the continuous test may fail even when there exists an axis which separates the boxes during the whole time interval.

5.1.1. Improved Continuous Overlap Test

We present an improved overlap test that computes a *refinement level* for the entire link for the given time interval. It is based on the motion of the link in the world space and uses the same refinement level for all the OBBs associated with the link.

More precisely, for a given link i, the refinement level r_i computes the number $n_i = 2^{r_i}$ of equally-sized time sub-intervals the given time interval has been split into for evaluating the bounds $\tilde{\mathbf{M}}_i^0(t)$ for the link i. When performing a continuous overlap test between two OBBs, the separating axis tests are executed on the time sub-intervals processed in the order of increasing time values, and stop as soon as an overlap has been detected for a given time sub-interval. The maximum of the refinement levels of the two objects determines the resolution of the test. For example, given a time interval, $[0,1]$, and $r_0 = 1$ and $r_1 = 2$. Then at most four continuous overlap tests are performed with different pairs of bounds:

- one for $[0, 0.25]$ with $\tilde{\mathbf{M}}_0^0[0, 0.5]$ and $\tilde{\mathbf{M}}_1^0[0, 0.25]$,
- one for $[0.25, 0.5]$ with $\tilde{\mathbf{M}}_0^0[0, 0.5]$ and $\tilde{\mathbf{M}}_1^0[0.25, 0.5]$,
- one for $[0.5, 0.75]$ with $\tilde{\mathbf{M}}_0^0[0.5, 1]$ and $\tilde{\mathbf{M}}_1^0[0.5, 0.75]$,
- one for $[0.75, 1]$ with $\tilde{\mathbf{M}}_0^0[0.5, 1]$ and $\tilde{\mathbf{M}}_1^0[0.75, 1]$.

The boxes are known to be disjoint over the given time interval when they are disjoint over each of the time sub-intervals. Note that using distinct refinement levels for distinct links is not problematic and still provides a conservative test, as in this example where the bounds $\tilde{\mathbf{M}}_0^0[0, 0.5]$ (resp. $\tilde{\mathbf{M}}_0^0[0.5, 1.0]$) contain the exact bounds on $\mathbf{M}_0^0(t)$ over the two time sub-intervals $[0, 0.25]$ and $[0.25, 0.5]$ (resp. $[0.5, 0.75]$ and $[0.75, 1]$). The refinement levels and the bounds $\tilde{\mathbf{M}}_i^0(I)$ are computed only once for a given time interval, and not each time an overlap test between two boxes has to be performed.

5.2. Elementary Continuous Collision Detection

When two leaf-nodes in the OBB trees overlap, we need to compute the first time of contact (TOC) between the triangular primitives that are contained in the leaf nodes. In this section, we present a novel and improved algorithm for fast continuous collision detection between the triangular primitives. It is based on techniques for solving multiple equations simultaneously. We first present the mathematical formulation of the "elementary tests". Next, we highlight the performance limitations of prior approaches in solving the set of resulting equations. Finally, we present our novel algorithm for simultaneously solving the set of equations resulting from the elementary tests.

5.2.1. Elementary Tests

Given two triangles i and j, two types of contacts can occur between them: either a collision between the edges of i with edges of j or a collision between a vertex of i with the face of j (and vice-versa). As a result, we need to perform two types of *elementary*

Angle θ_i^{max}	Dynamic BVH Culling		Dynamic SV Culling		Exact Contact Computation		Total Time for CCD	
	COL	NO-COL	COL	NO-COL	COL	NO-COL	COL	NO-COL
1	0.0014	0.0012	11.9552	2.9801	11.8	3.3123	23.7566	6.2936
5	0.0017	0.0013	16.1848	4.3327	17.9427	3.6844	32.3713	8.0184
15	0.0018	0.0015	22.1523	3.9728	30.5652	5.4548	52.7193	9.4291
30	0.0018	0.0013	18.6973	4.4477	85.4134	18.7761	104.1125	23.2251

Table 1: *Performance of our algorithm for different trajectories of the Puma robot in the "Pipes" environment (timings in milliseconds). In the first column, a higher value of θ_i^{max} implies a larger motion. The other columns show the average time spent in different stages of the algorithm depending on whether a collision is detected (COL) or not (NO-COL), during the timestep.*

tests: edge/edge ($E-E$) or vertex/face ($V-F$) tests. The continuous collision detection equations for both tests can be derived easily [Can86, RKC00, KR03]. Given two moving edges expressed as functions of their moving end-vertices: $\mathbf{a}(t)\mathbf{b}(t)$ of i and $\mathbf{c}(t)\mathbf{d}(t)$ of j. The lines supporting them collide during a given time interval if there exists a real root of the equation ($E-E$):

$$\mathbf{a}(t)\mathbf{c}(t).(\mathbf{a}(t)\mathbf{b}(t)\wedge\mathbf{c}(t)\mathbf{d}(t))=0 \qquad (13)$$

in the given time interval. For each root, it is then checked whether the *edges*, and not only the supporting lines, are colliding. In the degenerate case where the edges are parallel, two contact points are reported. Similarly, a collision occurs between a vertex $\mathbf{a}(t)$ of i (resp j.) and the plane containing the triangle j (resp. i) $\mathbf{b}(t)\mathbf{c}(t)\mathbf{d}(t)$ if there is a real root of the equation ($V-F$):

$$\mathbf{a}(t)\mathbf{b}(t).(\mathbf{b}(t)\mathbf{c}(t)\wedge\mathbf{b}(t)\mathbf{d}(t)=0 \qquad (14)$$

in the given time interval. For each root, it is then checked whether at that time the vertex is *inside the triangle*.

We use an interval numerical method for root computation. Consequently, we need to obtain some bounds on the positions and derivatives of the elements (*i.e.* the vertices, edges and face normals) over the given time intervals. The bounds on the positions over a given time interval I are determined by first computing bounds $\tilde{\mathbf{M}}_i^0(I)$ on the position matrices $\mathbf{M}_i^0(t)$ over this time interval, and then performing interval matrix-vector multiplications to obtain the bounds on the elements positions. The bounds on the derivatives are computed in a similar manner.

Although conceptually simple, the computation of these bounds can become quite expensive for articulated models. These computations are performed by concatenating the interval matrices and therefore, the computational cost of the evaluation of the bounds $\tilde{\mathbf{M}}_i^0(I)$ grows linearly with the depth of link i (*i.e.* the number of links separating it from the root of the articulated model in the DAG). Consequently, the cost of solving one of the equations is

$$C=O(s.(d_i+d_j)), \qquad (15)$$

where d_i and d_j are the depths of links i and j, respectively, in the DAG, and s is the number of time intervals over which the bounds have been computed during the resolution of the equation. As a result, if n_e equations need to be solved for the pair of links i and j, and a is the average number of required time intervals user per equation, then the cost of solving the elementary equations independently is

$$C_{ind}=O(a.n_e.(d_i+d_j)). \qquad (16)$$

This can be relatively expensive for complex articulated models consisting of many links.

5.2.2. Simultaneous Solver for Articulated Models

We overcome the high complexity for articulated models by simultaneously solving all the elementary equations (13) and (14) that are generated during the traversal of OBB trees. If we solve the elementary equations independently for different $V-F$ and $E-E$ combinations, it requires the computation of bounds, $\tilde{\mathbf{M}}_i^0(I)$, over each sequence of time intervals for each equation. Instead, we maintain lists of *active* equations over the time intervals and compute the bounds $\tilde{\mathbf{M}}_i^0(I)$ only once per time interval. By using such a simultaneous solver, the cost of solving n_e equations becomes

$$C_{sim}=O(s_{max}.(d_i+d_j))+O(a.n_e), \qquad (17)$$

where s_{max} is the total number of time intervals processed simultaneously. In practice, s_{max} is usually much smaller than $a.n_e$. As a result, the simultaneous solver results in an improved performance. Next, we present an improved version of simple interval subdivision method and Newton interval method, which are part of the simultaneous solver.

5.2.3. Improved Interval Subdivision Method

Given a list of n_e elementary continuous collision detection equations, $f_1(t)=0,\ldots,f_{n_e}(t)=0$, which need to be solved on a given time interval $[l,r]$. This list is generated during the traversal of the OBB-trees and the elementary equations may involve any robot link or obstacle in the environment. We refer to the equations in this list as *active equations* on the time interval $[l,r]$, because they may have a root in this interval. We first compute the bounds $\tilde{\mathbf{M}}_i^0[l,r]$ for the links involved in the equations. Next, these bounds are used to bound the positions of all the elements involved in the equations, by performing matrix-vector interval multiplications. Finally, we bound each function $f_k()$ by performing elementary interval operations. If the bounds on a given function $f_k()$ have identical signs, then we know for sure that it does not have any root in $[l,r]$, and the equation can be discarded. Otherwise, the function $f_k()$ may have a root in $[l,r]$. A list of all such functions is computed and the same process is recursively applied to this list on two smaller time intervals: first on $[l,m]$ and then on $[m,r]$, where $m=\frac{l+r}{2}$, since we are trying to compute the first TOC. The recursion is stopped when the list of equations becomes empty or when the time interval is smaller than a user-defined threshold. In the latter case, the validity of the roots is checked for all active equations and all valid roots are reported.

5.2.4. Improved Newton Interval Method

We now extend the improved interval subdivision method by introducing a *Newton culling step* for articulated models. The traditional Newton interval method attempts to reduce the size of the current time interval by computing bounds on the derivative of the function whose roots are being computed. Given a function $f()$ with bounds on the current time interval $[l,r]$, assume that the bounds have opposite signs. We further assume that some bounds $[a,b]$ on its first derivative $f'()$ have been computed as well, with $a.b > 0$. Then the search interval can be safely reduced to the interval

$$\left([m,m] - \frac{[f(m),f(m)]}{[a,b]}\right) \cap [l,r] \qquad (18)$$

where m is any point in $[l,r]$ (usually the midpoint of the interval).

Since we are simultaneously solving for different equations on identical time intervals, we cannot reduce the interval for each active equation $f_k()$ independent of the others. Most of the time, we use the Newton interval method to potentially *cull away* an equation and speed up the improved interval dichotomy method, when the interval used by the Newton test does not intersect one or both of the time intervals $[l,m]$ and $[m,r]$. When one of these two time intervals have been reduced for all of the active equations, we can replace it by the union of the smaller intervals and recursively process the list of active equations on the smaller intervals.

6. Implementation and Results

We have implemented the CCD algorithm described and tested on a 2.4 GHz Pentium PC with 1 Gbyte of RAM and a NVIDIA GeForce FX 5800 graphics card. The dynamic SV culling algorithm used during contact localization uses the graphics hardware to perform overlaps between the SV of the LSS and the rest of the environment.

Pipes and Puma robot: We first have used a benchmark consisting of a Puma robot model (800 triangles and 7 links) and a CAD/CAM model of pipes (38,000 triangles), shown in Fig. 1. It is a relatively dense environment and we place the robot in close proximity to the pipes. In practice, our algorithm manages to compute the time of first contact, as well as the contacting location between an articulated model and a complex environment in tens of milliseconds. The average time required to perform a continuous overlap test (Section 5.1.1) between two moving OBBs is about one microsecond, when the bounds on the matrix elements have already been computed.

In order to evaluate the overall performance of our algorithm, we generated various random paths in the environment visible in Figure 1, in the following way. Starting from a collision free position, random motion parameters are generated for the current timestep. These parameters determine a unique interpolating motion (i.e. the arbitrary in-between motion), as defined in Section 3.3. Continuous collision detection is performed using this motion trajectory. If any link of the robot collides with the pipes, the algorithm computes the first TOC and the position of each link at that time.

In order to evaluate the influence of the amplitude of the motion on the performance of our algorithm, we used four different random trajectories. For each trajectory $i = 1,\dots,4$, the angular motion of each link for each timestep is randomly chosen between $0°$ and θ_i^{max}, with $\theta_1^{max} = 1°$, $\theta_2^{max} = 5°$, $\theta_3^{max} = 15°$ and $\theta_4^{max} = 30°$.

Table 1 shows the average time spent in each of the three stages of our algorithm as well as the total query time. Moreover, we show the average time when any of the links collides with the environment (COL) as well as when there is no collision (NO-COL). The results show that the first stage of dynamic BVH culling takes very little time as compared to the other two stages. Moreover, the cost of dynamic SV culling doesn't increase significantly with a larger motion. On the other hand, the cost of the third stage, computing the exact time of contact as well as the contact features, depends directly on the amplitude of the robot motions. There are two main reasons:

- The exact contact computation includes solving many elementary continuous collision detection equations. As we take a higher value of the angular motion and compute the motion trajectory, more features of the first objet penetrate deeply into the second object. This results in more pairwise collisions between the OBBs and triangles. As a result, many more elementary equations are generated during the traversal of the bounding-volume hierarchies and the CCD algorithm spends more time in the third stage.
- All the bounds are computed using interval arithmetic. They are not exact, but only conservative. In fact, they tend to be more and more conservative as the amplitude of the motion, or the depth of the links, increases. Consequently, more time is spent in computing the bounds on equations which have no solutions for the current time interval.

Auxiliary Machine Room and Puma robot: We have then measured the cost and benefit of each step of our algorithm by placing the same Puma robot in a partial model of an Auxiliary Machine Room (AMR). This new environment, visible in Figure 7, consists of 1,180 objects and 187,000 triangles. Again, we have used various values of the maximum rotation angle θ_i^{max}. This time however, we have for each trajectory determined the benefit of the two culling stages by measuring the average times needed by the other steps when they are deactivated. Table 2 gives the average time required by each stage, as well as the average total time, depending on the maximum rotation angle per link θ_i^{max} and the set of active stages. Again, the average times are given for two cases: when at least one of the links collides with the environment (COL), and when there is no collision (NO-COL). As expected, dynamic SV culling becomes useful when large motions occur, to counteract the increased conservativeness in OBB culling and the simultaneous resolution of the elementary collision detection equations resulting from the use of interval arithmetic in the exact contact computation stage. The results show that, even for large motions, the proposed algorithm is able to compute the first time of collision and the contact state at nearly interactive rates. Moreover, recent benchmarks show that SV culling becomes essential as the depth of the articulated model increases [RKLM03].

7. Analysis and Limitations

We have highlighted the performance of the algorithm in close proximity configurations in Section 6. We are able to perform continuous collision detection and compute the first possible time of contact in tens of milliseconds (when the articulated model is colliding with the environment). We are able to achieve this performance by using a three-stage algorithmic pipeline. The relative

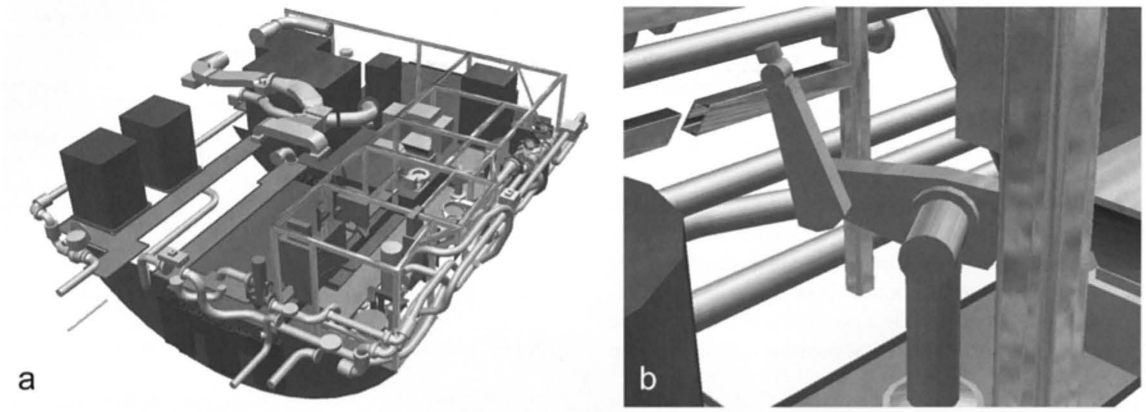

Figure 7: *The Puma robot in the AMR environment (1,180 objects, 187,000 triangles).*

		Dynamic BVH Culling		Dynamic SV Culling		Exact Contact Computation		Total Time for CCD	
Angle θ_i^{max}	Stages	COL	NO-COL	COL	NO-COL	COL	NO-COL	COL	NO-COL
1°	1+2+3	0.33	0.33	41.58	18.54	7.01	2.06	48.92	19.04
	1+3	0.34	0.33	-	-	7.33	1.79	7.67	1.24
	2+3	-	-	47.60	41.68	23.29	15.00	70.89	43.53
	3	-	-	-	-	82.24	75.05	82.24	75.05
30°	1+2+3	0.33	0.33	30.83	20.18	116.42	46.15	147.58	48.31
	1+3	0.33	0.33	-	-	121.93	43.43	122.26	115.40
	2+3	-	-	56.14	44.74	147.00	63.72	203.14	98.90
	3	-	-	-	-	190.79	98.41	190.79	98.41
60°	1+2+3	0.34	0.33	43.13	19.05	480.45	91.74	523.92	40.91
	1+3	0.35	0.33	-	-	577.70	73.06	578.05	36.79
	2+3	-	-	62.23	44.70	519.39	107.86	581.62	70.08
	3	-	-	-	-	649.48	107.29	649.48	107.29

Table 2: *Average execution times for the AMR environment and the Puma robot (in milliseconds). Dynamic SV culling is useful to counteract the increased conservativeness in the third stage of our algorithm when large motions occur. Recent benchmarks show that SV culling becomes essential as the depth of the articulated model increases [RKLM03].*

benefit of each stage depends on the model complexity of the articulated model, the simulated environment, and the relative configuration of each link of the articulated model with respect to the environment. Overall, the algorithm spends very little time in dynamic BVH culling, nearly constant time in dynamic SV culling, and the performance of exact contact computation varies considerably with the length of the motion trajectory.

7.1. Sources of errors and potential solutions

Our algorithm is not exact. The sources of errors can be attributed to the following processes in the algorithm:

- **Surface Tessellation Error:** In Sec. 4.2.2, we approximate the SV of LSS using planar surface patches. As a result, we tessellate the pipe and offset surface within some error deviation, ε. Thus, if the articulated object moves closer to some of the objects in the environment within ε or penetrates the objects by ε, these collisions can be missed. For applications where guaranteed conservativeness is absolutely required, dynamic swept-volume culling can be suppressed, to the expense of a higher overall cost. Recent benchmarks show the importance of dynamic swept-volume culling as the depth of the articulated model increases [RKLM03].
- **Image Space Precision Error:** We use a graphics-hardware based collision checking algorithm to check for an collision of the tessellated SV. As a result, the precision of the algorithm is limited by the underlying hardware precision such as frame and depth buffer resolution. However, recent results show that GPU-based interference checking can be made *conservative*, so that no collision is ever missed [GLM04].
- **Floating Point Error:** Essentially, the precision of the interval arithmetic and root-finding methods are limited by underlying floating-point precision: the interval dichotomy method presented in Sec. 5.2.3 requires a certain threshold to stop the refinement. This threshold also depends on the floating point precision. However, with a careful implementation, the interval arithmetic computations can be made conservative [SWF*93].

8. Conclusions and Future Work

In this paper, we have presented a novel algorithm for continuous collision detection between a moving articulated model and the simulated environment. The algorithm consists of three stages that perform dynamic BVH culling, dynamic SV culling and exact contact computation respectively. We use interval arithmetic to construct the dynamic BVH, and use a graphics hardware accelerated algorithm to perform the dynamic SV culling. We have applied the algorithm to an articulated robot model moving in a complex CAD environment composed of tens of thousands of polygons. Our initial results are quite promising and the algorithm is able to compute all the contacts, as well as the time of first possible collision within tens of millisecond.

There are many avenues for future work. We plan to perform a more thorough analysis of the potential of the new algorithm. Preliminary results discuss and show the importance of the various culling steps in complex benchmarks, especially when the depth of the articulated body increases [RKLM03].

We believe the algorithm described in the paper can be easily extended to handle self-collision detection as well as multiple moving articulated bodies, and we plan to investigate this topic. Also, we would like to apply our interference algorithm to other potential applications such as virtual reality-based training, dynamics simulation, etc. In particular, we would like to use it for local planning in PRM-based planners. Furthermore, we want to extend our algorithm to relax the no-loop constraints in the articulated chain such that the algorithm is applicable to all articulated models.

Acknowledgements

This project is supported in part by ARO Contract DAAD19-02-1-0390, NSF awards IIS-982167, ACI 987691 and ACR-0118743, ONR VIRTE Program Contract N00014-01-1-0496, a Ewha Womans University research grant of 2003, and Intel.

References

[AMBJ02] Abdel-Malekl K., Blackmore D., Joy K.: Swept volumes: Foundations, perspectives, and applications. *International Journal of Shape Modeling* (2002).

[AMO99] Abdel-Malek K., Othman S.: Multiple sweeping using the denavit-hartenberg representation method. *Computer-Aided Design 31* (1999), 567–583.

[Bur96] Burdea G.: *Force and Touch Feedback for Virtual Reality*. John Wiley and Sons, 1996.

[BW01] Baraff D., Witkin A.: *Physically-Based Modeling*. ACM SIGGRAPH Course Notes, 2001.

[Cam90] Cameron S.: Collision detection by four-dimensional intersection testing. *Proceedings of International Conference on Robotics and Automation* (1990), 291–302.

[Can86] Canny J. F.: Collision detection for moving polyhedra. *IEEE Trans. PAMI 8* (1986), 200–209.

[Duf92] Duff T.: Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), Catmull E. E., (Ed.), vol. 26, pp. 131–138.

[EL01] Ehmann S., Lin M. C.: Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001) 20*, 3 (2001), 500–510.

[ELK97] Elber G., Lee I.-K., Kim M.-S.: Comparing offset curve approximation methods. *IEEE Computer Graphics and Applications 17*, 3 (1997), 62–71.

[Far86] Farouki R.: The approximation of non-degenerate offset surfaces. *Computer Aided Geometric Design 3* (1986), 15–43.

[FMM86] Filip D., Magedson R., Markot R.: Surface algorithms using bounds on derivatives. *CAGD 3* (1986), 295–311.

[GLGT99] Gregory A., Lin M., Gottschalk S., Taylor R.: H-collide: A framework for fast and accurate collision detection for haptic interaction. In *Proceedings of Virtual Reality Conference 1999* (1999), pp. 38–45.

[GLM96] Gottschalk S., Lin M., Manocha D.: OBB-Tree: A hierarchical structure for rapid interference detection. *Proc. of ACM Siggraph'96* (1996), 171–180.

[GLM04] Govindaraju N., Lin M., Manocha D.: *Fast and reliable collision detection using graphics hardware*. Tech. rep., University of North Carolina, Department of Computer Science, 2004.

[GRLM03] Govindaraju N., Redon S., Lin M., Manocha D.: Cullide: Interactive collision detection between complex models in large environments using graphics hardware. *Proc. of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2003), 25–32.

[Hos88] Hoschek J.: Spline approximation offset curves. *Computer Aided Geometric Design 5*, 1 (1988).

[Hub95] Hubbard P. M.: Collision detection for interactive graphics applications. *IEEE Trans. Visualization and Computer Graphics 1*, 3 (Sept. 1995), 218–230.

[HZLM01] Hoff K., Zaferakis A., Lin M., Manocha D.: Fast and simple 2d geometric proximity queries using graphics hardware. *Proc. of ACM Symposium on Interactive 3D Graphics* (2001), 145–148.

[KHM*98] Klosowski J., Held M., Mitchell J. S. B., Zikan K., Sowizral H.: Efficient collision detection using bounding volume hierarchies of *k*-DOPs. *IEEE Trans. Visualizat. Comput. Graph. 4*, 1 (1998), 21–36.

[KL03] Kim K.-J., Lee I.-K.: The perspective silhouette of a canal surface. In *Eurographics (Graphics Forum)* (2003), vol. 22, pp. 15–22.

[Kla83] Klass R.: An offset spline approximation for plane cubic splines. *Computer-Aided Design 15*, 5 (1983), 297–299.

[KR03] Kim B., Rossignac J.: Collision prediction for polyhedra under screw motions. In *ACM Conference on Solid Modeling and Applications* (June 2003).

[KSLO96] Kavraki L., Svestka P., Latombe J. C., Overmars M.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.* (1996), 12(4):566–580.

[KSS00] Kirkpatrick D., Snoeyink J., Speckmann B.: Kinetic collision detection for simple polygons. In *ACM Symposium on Computational Geometry* (2000), pp. 322–330.

[KVLM03] Kim Y., Varadhan G., Lin M., Manocha D.: Efficient swept volume approximation of complex polyhedral models. *Proc. of ACM Symposium on Solid Modeling and Applications* (2003).

[LGLM00] Larsen E., Gottschalk S., Lin M., Manocha D.: Distance queries with rectangular swept sphere volumes. *Proc. of IEEE Int. Conference on Robotics and Automation* (2000).

[LM03] Lin M., Manocha D.: Collision and proximity queries. In *Handbook of Discrete and Computational Geometry* (2003).

[LSHL02] Lotan I., Schwarzer F., Halperin D., Latombe J.: Efficient maintenance and self-collision testing for kinematic chains. *Proc. of Symposium on Computational Geometry* (2002), 43–52.

[Moo79] Moore R. E.: *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979. ISBN 0-89871-161-4.

[NVI03] NVIDIA Corporation: `http://oss.sgi.com/projects/ogl-sample/registry/NV/occlusion_query.txt`, 2003.

[PW01] Pottmann H., Wallner J.: *Computational Line Geometry*. Springer, 2001.

[RK00] Rossignac J., Kim J.: Computing and visualizing pose-interpolating 3-D motions. *Computer-Aided Design* (2000).

[RKC00] Redon S., Kheddar A., Coquillart S.: An algebraic solution to the problem of collision detection for rigid polyhedral objects. *Proc. of IEEE Conference on Robotics and Automation* (2000).

[RKC02] Redon S., Kheddar A., Coquillart S.: Fast continuous collision detection between rigid bodies. *Proc. of Eurographics (Computer Graphics Forum)* (2002).

[RKLM03] Redon S., Kim Y. J., Lin M. C., Manocha D.: *Fast Continuous Collision Detection for Articulated Models*. Tech. Rep. TR03-038, University of North Carolina at Chapel Hill, 2003.

[RMS92] Rossignac J., Megahed A., Schneider B.: Interactive inspection of solids: cross-sections and interferences. In *Proceedings of ACM Siggraph* (1992), pp. 353–60.

[SSL02] Schwarzer F., Saha M., Latombe J.-C.: Exact collision checking of robot paths. In *Workshop on Algorithmic Foundations of Robotics (WAFR)* (Dec. 2002).

[SWF*93] Snyder J. M., Woodbury A. R., Fleischer K., Currin B., Barr A. H.: Interval method for multi-point collision between time-dependent curved surfaces. In *Computer Graphics (SIGGRAPH '93 Proceedings)* (Aug. 1993), Kajiya J. T., (Ed.), vol. 27, pp. 321–334.