Fast and Accurate Collision Detection for Virtual Environments

Ming C. Lin
Department of Computer Science
University of North Carolina at Chapel Hill
lin@cs.unc.edu

Abstract

A realistic simulation system, which couples geometry and physics, can provide a useful toolkit for virtual environments. Interactions among moving objects in the virtual worlds are modeled by dynamic constraints and contact analysis. In order to portray the geometric interaction in a dynamical system or to simulate physical behaviors of entities in the virtual environments, it is essential to perform collision detection at interactive rate with minimal computation possible. In this paper, we present efficient algorithms for contact determination and interference detection between geometric models undergoing rigid motion. The set of models includes polyhedra and surfaces described by B-splines. The algorithms make use of temporal and spatial coherence between successive instances to reduce the number of pairwise tests and hierarchical data structures for checking overlaps between a pair of objects. Their running time is a function of the motion between successive instances. The main characteristics of these algorithms are their simplicity and efficiency. They have been implemented. A subset of these implementations, including I-Collide, RAPID and V-Collide are available as part of the collision detection packages at the UNC-CH website.

1 Introduction

Over the last few years there has been a great deal of interest in collision detection for interactive 3D graphics. A physics simulation engine takes in the geometry of the objects in the environment, their moving parts and physical properties to mimic their physical behaviors. Moving entities and avatars need to behave like the real ones as in the physical world, in order to enhance the degree of realism. However, real time dynamic simulation has been unattainable till recently, due to the absence of efficient, practical collision detection algorithms and fast, accurate dynamics computations.

Other than virtual environments, a real-time collision detection system is also an integral part of robotics and automation, engineering simulation and analysis, molecular modeling and drug design, and electronic prototyping. We will first give an overview of the state of the art. Then, we will present several collision detection algorithms that utilize a wide range of geometric techniques, including hierarchical data structure, *coherence* and *locality*. Concepts from computational geometry and geometric modeling are incorporated to design geometric algorithms for solving the collision detection problems among all geometric models in real time for most of the cases. These algorithms are generally applicable, but especially suited to dynamic domains where objects are moving at small, discrete steps. We'll also discuss the advantages and shortcomings of these methods. Next, we'll present techniques in reducing the number of pairwise intersection tests for a computer simulated environment consisting of multiple static or moving entities. We will also provide several online resources where a number of public domain packages for collision detection can be easily obtained.

The rest of the paper is organized as follows. In Section 2, we discuss problem domain specification based on different classification. In Section 3, we review some of the previous work in collision detection. Section 4 gives an overview on our collision detection systems. We describe our novel approaches to the problem of collision detection in Section 5 and briefly discuss collision response in Section 6 respectively. Next, we demonstrate our experimental results on several applications in Section 7. Section 8 provides a list of public domain libraries. We conclude with several future research directions.

2 Problem Domain Specification

A wide range of techniques, including hierarchical representation, geometric reasoning, algebraic formulations, spatial partitioning, analytical methods, and optimization methods, have been proposed. Algorithm design depends on the model representation, the de-

sired query types, and the simulation environment.

2.1 Model Representation

There are many types of model representations used in CAD/CAM and 3D graphics. Polygonal objects are the most commonly used models in computer graphics. They have a simple representation. They are versatile. Hardware-accelerated rendering of polygon is widely available. The most general class of polygonal model is the polygon soup, which is a collection of polygons that are not geometrically connected and has no topology information available. If the polygons form a closed manifold, then the model has a well-defined inside and outside – it is a proper solid. Some geometric algorithms rely on this structure. If the object defined by the closed manifold is convex, then this additional structure can be exploited in the design of collision detection algorithms.

Constructive Solid Geometry or CSG forms objects from primitives such as blocks, spheres, cylinders, cones, and tori, by combining them with set theoretic operations such as union, intersection, and set difference [18]. One strength of the CSG representation is that it enables an intuitive design process of building shapes by means of cutting (intersection and set difference) and joining (union) simple shapes to form more complex ones. The difficulty with CSG is that certain operations, such as rounding an edge or filleting a join, are difficult to describe with CSG operations. Furthermore, an accurate boundary or surface representation, useful for rendering or interference computations, can be hard to compute from CSG representations [18].

Implicit surfaces are defined using implicit functions, with mappings from 3D space to the real numbers, $f : R^3 \rightarrow R$, and the implicit surfaces are the loci of points where $f(x, y, z) = 0$. Such a function defines unambiguously what is inside the model, $f(x, y, z) < 0$ and what is outside, $f(x, y, z) > 0$. Consequently, implicit surfaces are generically closed manifolds.

Parametric surfaces are mappings from some subset of the 2D plane to 3D space, $f : R^2 \rightarrow R^3$. Unlike implicit surfaces, parametric surfaces are not generally closed manifolds. Hence, unlike CSG and implicit surfaces, they do not represent a complete solid model, but rather a description of surface boundary.

2.2 Different Types of Queries

In the simplest case, we want to know whether two models touch. Sometimes, we must find which parts (if any) touch, i.e. find their intersection. Sometimes we want to know their separation: if two objects are disjoint, what is the minimum Euclidean distance between them? If they penetrate, what is the minimum translational distance required to separate them [8]? Finally, if we know the objects' placements and motions, when will be their next collision? This is ETA computation, borrowing from the phrase, *estimated time of arrival.*

Different applications need different queries. Distance information is useful for computing interaction force and penalty functions in robot motion planning [27] and dynamic simulation [28, 37]. Intersection computation is important for physically-based modeling and animation systems which must know all contacts in order to compute the collision response. The ETA solution permits us to control the time step in a simulation [28, 31].

2.3 Simulation environments

Special characteristics of each simulation are often considered in designing and choosing the most appropriate algorithm for collision detection. Here we examine a few common cases.

- **Pair Processing vs. Nbody Processing**

  If the problem involves only a pair of models, we call it *pair processing.* If we have many different parts, we call it *Nbody processing*, in reference to the classic problem in celestial mechanics (many bodies moving under mutual gravitational influence).

- **Motions: Static vs. Dynamic**

  Queries are often executed repeatedly on the same models in the same environment, as the objects rotate and translate (or possibly subject to non-rigid transformations) at successive time steps. In these dynamic environments, the geometric relationship may only differ slightly from that of the previous step, if the motion between steps is relatively small. Algorithms that can capitalize on this property are said to exploit *coherence.*

  In order to exploit coherence, some algorithms require bounds [28, 31] on the motion of the objects (e.g. objects' velocities or accelerations). Other algorithms, such as the ones based on interval arithmetic, need a closed-form expression of the motion as a function of time. Some algorithms demand no information on the motion, but need only the placements of the objects at successive time steps.

  Sometimes the problem involves objects that are not in motion. For example, given a model of an entire power-plant, design engineers may be interested in performing static interference checks among components of the entire plant for tolerance verification and access clearance.

• **Rigid Bodies vs. Deformable Models**

When the component of time is introduced, there is also the possibility that the models deform over time. Assuming that the deformations between time steps are small, some algorithms may be able to exploit coherence in this case as well.

3 Brief Survey On State Of Art

There is a rich body of literature in both analyzing the geometric contacts of moving objects and the motion dynamics of rigid bodies. Collision detection has been a fundamental problem in robotics, computational geometry, animation and simulation, physical-based modeling. One of major bottlenecks in building real-time dynamic simulators is finding a practical, efficient and simple collision detection algorithm [17]. We will limit the scope of the discussion in this paper to mostly rigid polygonal models, though some of the techniques are applicable to other domains and model representation as well.

3.1 Collision Detection for Polygonal Models

Most of the earlier work in collision detection has focused on algorithms for convex polytopes. A number of algorithms with good asymptotic performance have been proposed in the computational geometry literature. Using hierarchical representations, an $O(log^2 n)$ algorithm is given in [DK90] for polytope-polytope overlap problem, where n is the number of vertices. This elegant approach has not been robustly implemented in 3D, however.

Good theoretical and practical approaches based on linear complexity of the linear programming problem are known [35, 44]. Minkowski difference and convex optimization techniques are used in [14] to compute the distance between convex polytopes by finding the closest points.

In applications involving rigid motion, geometric coherence has been exploited to design algorithms for convex polyhedra based on local features [2, 29, 28]. These algorithms exploit the spatial and temporal coherence between successive queries and work well in practice.

A number of hierarchies have been used for collision detection between general polygonal models. Typical examples of bounding volumes include axis-aligned boxes (cubes are a special case) and spheres, and they are chosen for their fast overlap tests. Other structures include cone trees, k-d trees and octrees [43], sphere trees [19, 41], trees based on S-bounds [6], etc. Binary space partitions (BSP) [39] and extensions to multi-space partitions [4], and spatial partitionings based on space-time bounds or four-dimensional testing [5, 9, 13, 19] have been used. All of these hierarchical methods do very well in performing "rejection tests" whenever two objects are far apart. However, when the two objects are in close proximity and can have multiple contacts, these algorithms either use subdivision techniques or check very large number of bounding volume pairs for potential contacts. In such cases, their performance slows down considerably.

More recent work seems to have focused on tighter-fitting bounding volumes. Gottschalk et al. [15] have presented a fast algorithm and a system, called RAPID, for interference detection based on oriented bounding boxes, which approximate geometry better than do axis-aligned bounding boxes. Barequet et al. [3] have also used oriented bounding boxes for computing hierarchical representations of surfaces for performing collision detection. Klosowski et al. [21, 22] have used discrete orientation polytopes (k-dops), which also are superior approximations to bounded geometry. Krishnas et. al. [25, 23] have proposed a higher order bounding volume, designed to match curvature of the underlying 3D geometry, especially suited for Bézier patches and NURBS.

3.2 Algorithms for Non-Polygonal Objects

In geometric and solid modeling, the problem of computing the intersection of surfaces represented as splines or algebraic surfaces has received a great deal of attention [18]. Given two surfaces, the problem corresponds to computing all components of the intersection curve, robustly and accurately. It includes work on curves and surface intersections [18, 33, 34, 24]. All these algorithms have focussed on accurate computation of the intersection set for static models. However, for collision detection we are actually dealing with a restricted version of this problem. That is, given two surfaces we want to know whether they intersect. Furthermore, we are interested in dynamic environments composed of moving objects.

In general, given two spline surfaces, there is no good and quick solution to the problem of whether they intersect or have a common geometric contact. The simplest solution is based on using subdivision and checking the control polytopes or convex bounding boxes for collision. For a more complete state-of-art report, please refer to a recent survey [30].

4 Multi-Level Approach

Our proposal takes a multi-level approach to the problem of collision detection. This technique has been incorporated into the design of I-COLLIDE and V-COLLIDE library [10, 20].

4.1 System Overview

In this section, we briefly describe the system architecture of our collision detection systems. A quick conservative approximation finds potentially-colliding pairs of objects among the entire database (using the n-body sweep-and-prune algorithm to be described next), after which a pairwise test (using those to be described in Section 5) determines whether two objects marked as overlapping actually collided.

Our collision detection algorithms utilize hierarchical data structures and whenever it is appropriate we also exploit the properties of *locality* and *coherence*. Several concepts from computational geometry and geometric modeling are incorporated to solve the collision detection problems among all geometric models in real time, for most of the cases. The algorithms are generally applicable, but especially well suited to dynamic domains where objects are moving at small, discrete steps.

The performance of our low-level collision detection algorithms for convex polyhedral models is independent of the model complexity and is a function of the object motion. The run time for checking interference between non-convex objects and curved models (such as NURBS, algebraic surfaces) is output sensitive. At the top-level, we use a *scheduling scheme* or *sweep and prune* technique to reduce the quadratic number of pairwise interference tests in large, interactive virtual environments, or scheduling scheme for dynamic simulation. Next, we will describe them in details.

4.2 Scheduling Scheme For Dynamic Simulation

In a dynamic simulation, where the motion is subject to dynamic constraints or external forces and cannot typically be expressed as a closed form function of time, various algorithms have been proposed [1, 2, 9, 5, 19, 28]. Different methods have also been proposed to overcome the bottleneck of $O(N^2)$ pairwise tests for N moving objects in an environment. The simplest of these are based on spatial subdivision. Another approach operates directly on four-dimensional volumes swept out by object motion over time [5]. None of these approaches adequately address the issue of real-time, exact collision detection for simulated environments, which requires performance at interactive rates for thousands of moving objects.

In order to avoid unnecessary computations and to speed up the run time, we use a scheduling scheme to reduce the frequency of collision detection. The algorithm maintains a priority queue (implemented as a heap) of all pairs of objects that might collide. They are sorted by estimated time to collision; with the one most likely to collide appearing at the top of the heap. The approximation is a lower bound on the time to collision and is calculated adaptively, so no collisions are missed. Non-convex objects, which are represented as hierarchy trees are treated as single objects from the point of view of the queue. More detail of the algorithm is given in [28, 31]. This scheduling scheme has been used successfully in the *Impulse* dynamic simulation system developed at Berkeley [37].

4.3 N-Body Algorithm Using Sweep and Prune

However, in an interactive virtual environment, we often cannot establish an upper bound on acceleration a priori. We propose to use a *sweep and prune* technique to cull out objects which are far apart, based on their geometry and position, in an interactive environment where the motion of the objects is unpredictable or unconstrained. This method is simple and efficient. It is output sensitive and its running time is linearly dependent on the number of objects in the environment instead of quadratic dependence. We use dynamic bounding boxes, linear-time sweep and prune, and geometric coherence to quickly reject the object pairs that are unlikely to collide within the next frame. This mechanism has reduced the number of pairwise tests dramatically, especially in a large simulated environment.

Sorting the bounding boxes surrounding the objects is the key to our *sweep and prune* approach [10]. It is not intuitively obvious how to sort bounding boxes in 3-space to determine overlaps. We use a *dimension reduction* approach. If two bounding boxes overlap in 3-D, then their orthogonal projections on the $x-$, $y-$, and $z-$ axes must overlap. The sweep and prune algorithm begins by projecting each 3-D bounding box surrounding an object onto the $x-$, $y-$, and $z-$ axes. Since the bounding boxes are axially-aligned, projecting them onto the coordinate axes results in intervals. We are only interested in overlaps among these intervals, because a pair of bounding boxes can overlap if and only if their intervals overlap in all three dimensions.

We construct three lists, one for each dimension. Each list contains the values of the endpoints of the intervals in each corresponding dimension. By sorting these lists, we can determine which intervals overlap. In the general case, such a sort would take $O(NlogN)$ time, where N is the number of objects. We can reduce this time bound by keeping the sorted lists from the previous frame, updating only the interval endpoints. In environments where the objects make relatively small movements between frames, the lists will be nearly sorted, so we can re-sort using insertion sort in expected linear time.

5 Exact Collision Detection

In most computer simulation systems, interactions among objects are generated by modeling contact constraints and impact dynamics. Since prompt recognition of possible impacts is a key to successful response to collisions in a timely fashion, a simple and efficient algorithm for collision detection is necessary to fast and realistic simulation of moving objects. In contrast to previous work, where accuracy is sacrificed for speed in a virtual environment, here we describe algorithms that run at interactive rate and can compute the geometric contacts to the accuracy of input models.

5.1 Collision Detection Between Convex Polyhedra

The heart of our collision detection algorithm for convex polyhedra with topological information is a simple and fast incremental method [29, 28] to compute the distance between them. It utilizes convexity to establish *local applicability criteria* by using Voronoi regions to verify the closest feature pairs. As the objects travel through space, the algorithm takes advantage of coherence to keep track of the closest feature pairs (a combination of vertex, edge, or face) between them to update their spatial relation.

Each convex polytope is pre-processed into a modified boundary representation. The polytope data structure has fields for its features (faces, edges, and vertices) and corresponding Voronoi regions. Each feature is described with its geometric parameters and its neighboring features, i.e. the topological information of incidences and adjacencies. This preprocessing procedure is also used to guarantee expected constant time performance when checking for contacts in a dynamic environment. Next, well describe an important geometric concept that forms the basis of our algorithm for computing distance between two convex objects via tracking the closest feature pair.

Definition: A *Voronoi region* associated with a feature is a set of points closer to that feature than any other [PS85].

The Voronoi regions form a partition of the space outside the polytope, and form the generalized Voronoi diagram of the polytope. Note that the generalized Voronoi diagram of a convex polytope has linear number of features and consists of polyhedral regions. A cell is the data structure for a Voronoi region of a single feature. It has a set of constraint planes which bound the Voronoi region with pointers to the neighboring cells (which share a constraint plane with it) in its data structure. If a point lies on a constraint plane, then it is equi-distant from the two features which share this constraint plane in their Voronoi regions. For more details on the construction and properties, please refer to [29, 28].

Tracking Closest Features: Our method for finding closest feature pairs is based on Voronoi regions. It hinges upon a fact that *each nearest point on the pair of closest features between two convex polytopes must lie within the Voronoi region of the other closest feature.*

We start with a candidate pair of features, one from each polytope, and check whether the closest points lie on these features. Since the polytopes and their faces are convex, this is a local test involving only the neighboring features of the current candidate features. If either feature fails the test, we step to a neighboring feature of one or both candidates, and try again. As the Euclidean distance between feature pairs must always decrease when a switch is made, cycling is impossible for non-penetrating objects [29, 28]. An implementation with linear programming code is available as part of the I-COLLIDE at the UNC-CH website.

To handle penetration, *internal* pseudo Voronoi regions can be constructed as well to detect penetration [32]. This can also help reducing the amount of "walking" on the polyhedral surfaces by "tunneling" through the internal of the objects, where the motion of the objects is abrupt and large. In addition, this modification makes the original algorithm much more robust and efficient.

5.2 Interference Tests Between General Polygonal Models

• Sub-Part Hierarchy

As for non-convex objects, we can assume that each non-convex object is given as a union of convex polyhedra or is composed of several non-convex subparts, each of these can be further represented as a union of convex polyhedra or a union of concave subparts. With such an assumption, we propose to use a sub-part hierar-

chy tree to represent each non-convex object. At each time step, we examine the possible interference using Lin-Canny algorithm [29, 28] described above between two convex parts. If the parents of the sub-part hierarchy collide, the algorithm is applied recursively to their children. The algorithm will only signal a collision if there is actually an impact between the sub-parts of two objects; otherwise, there is no collision between the two objects. This approach guarantees that we find the earliest collision between concave objects while reducing computation costs.

However, in general, such a hierarchy is only possible to construct if we have the object hierarchy or we construct the object hierarchy by hand ourselves. This may be a reasonable assumption for simple virtual environments, but unrealistic for many virtual environments of CAD models or massive spatial structures. In many CAD models, there may be no topological connectivity or object hierarchy that can be easily and automatically extracted. Therefore, we suggest the use of the hierarchical bounding volumes to address the problem of interactive collision detection.

• Hierarchy of Oriented Bounding Boxes (OBBTrees)

A bounding volume (BV) is often used to bound or contain sets of geometric primitives, such as triangles, polygons, NURBS, etc. In a bounding volume hierarchy (BVH), BVs are stored at the internal nodes of a tree structure. The root BV contains all the primitives of a model, and children BVs each contain separate partitions of the primitives enclosed by the parent. Leaf node BVs typically contain one primitive. In some variations, one may place several primitives at a leaf node, or use several volumes to contain a single primitive [41].

Two models are compared using a tandem recursive traversal of their BVHs. Each recursive call is made with BVs A and B, say one from each hierarchy, and must determine if A and B overlap. If A and B are not overlapping, the recursion stops at that level. But if A and B are overlapping, the enclosed primitives may overlap and the algorithm is applied recursively to their children. If A and B are both leaf nodes, the primitives within them are compared directly.

We propose a data structure called OBBTrees [15] and an algorithm for efficient and exact interference detection amongst complex models undergoing rigid motion. The algorithm is applicable to all general polygonal models. It pre-computes a hierarchical representation of models using tight-fitting oriented bounding box trees (OBBTrees) using top-down tree construction. At runtime, the algorithm traverses two such trees and tests for overlaps between oriented bounding boxes based on a separating axis theorem, which takes less than 200 operations in practice. This test is about one order of magnitude faster compared to earlier algorithms for checking overlap between oriented bounding boxes.

The tree construction has two components: first is the placement of a tight fitting OBB around a collection of polygons, and second is the grouping of nested OBB's into a tree hierarchy. We want to approximate the collection of polygons with an OBB of similar dimensions and orientation. We triangulate all polygons composed of more than three edges. The OBB computation algorithm makes use of first and second order statistics summarizing the vertex coordinates. They are the mean and the covariance matrix respectively. The eigenvectors of a symmetric matrix, such as the covariance matrix, are mutually orthogonal. After normalizing them, they are used as a basis. We find the extremal vertices along each axis of this basis, and size the bounding box, oriented with the basis vectors, to bound those extremal vertices. Two of the three eigenvectors of the covariance matrix are the axes of maximum and of minimum variance, so they will tend to align the box with the geometry of a tube or a flat surface patch. The exact formula and construction details are available in [15].

Given an algorithm to compute tight-fitting OBBs around a group of polygons, we need to represent them hierarchically. Most methods for building hierarchies fall into two categories: bottom-up and top-down. Bottom-up methods begin with a bounding volume for each polygon and merge volumes into larger volumes until the tree is complete. Top-down methods begin with a group of all polygons, and recursively subdivide until all leaf nodes are indivisible. In our current implementation, we have used a simple top-down approach.

Our subdivision rule is to split the longest axis of a box with a plane orthogonal to one of its axes, partitioning the polygons according to which side of the plane their center point lies on. The subdivision coordinate along that axis was chosen to be that of the mean point of the vertices. If the longest axis cannot not be subdivided, the second longest axis is chosen. Otherwise, the shortest one is used. If the group of polygons cannot be partitioned along any axis by this criterion, then the group is considered indivisible.

Given OBBTrees of two objects, the interference algorithm typically spends most of its time testing pairs of OBBs for overlap. A simple algorithm for testing the overlap status for two OBB's performs 144 edge-face tests. In practice, it is an expensive test. OBBs are convex polytopes and therefore, algorithms based on linear programming and closest features computa-

tion can be applied to check for overlap. However, they are relatively expensive compared to the overlap tests between a pair of spheres or between a pair of axis-aligned bounding boxes.

One trivial test for disjointness is to project the boxes onto some axis (not necessarily a coordinate axis) in space. This is an *axial projection*. Under this projection, each box forms an interval on the axis. If the intervals don't overlap, then the axis is called a *separating axis* for the boxes, and the boxes must then be disjoint. If the intervals do overlap, then the boxes may or may not be disjoint– further tests may be required. We make use of the separating axis theorem presented in [15] to check for overlaps. According to it, two convex polytopes in 3-D are disjoint if and only if there exists a separating axis orthogonal to a face of either polytope or orthogonal to an edge from each polytope. Each box has 3 unique face orientations, and 3 unique edge directions. This leads to 15 potential separating axes to test (3 faces from one box, 3 faces from the other box, and 9 pairwise combinations of edges). If the are overlapping, then clearly no separating axis exists. So, testing the 15 given axes is a sufficient test for determining overlap status of two OBBs.

This algorithm has been implemented and we compare its performance with other hierarchical data structures such as trees of spheres and axis-aligned bounding boxes. We found superior performance using OBB-Trees on *parallel close proximity configuration* or *near contact situation*, which are the most challenging scenarios for any collision detection algorithm in dynamic simulation and virtual prototyping applications. It can robustly and accurately detect all the contacts between large complex geometry composed of hundreds of thousands of polygons at interactive rates.

5.3 Contact Determination Between Spline Models

We introduce hierarchies based on *spherical shells* [25, 23] and use them for proximity queries. A spherical shell corresponds to a portion of the volume between two concentric spheres and encloses the underlying geometry. We devise efficient algorithms to compute such volumes and fast overlap tests between two shells. The overlap test is only slower by a small factor (2 or 3 times as compared to the overlap test between two OBB's). Their main advantage comes from the fact that they provide local cubic convergence to the underlying geometry. By *cubic convergence* we mean that the bounding volume approximates the surface accurately up to the second order (if the surface is expressed as a Taylor's series). Therefore, there are fewer *false positives* in terms of overlap tests between the bounding volumes and the underlying primitives. This results in improved overall performance for proximity queries between two objects in close proximity configurations or between two objects with high-curvature surfaces. However, our results show that the local cubic convergence is restricted to elliptic (unlike hyperbolic) regions of the surface.

6 Collision Response

Given the ability to detect collisions at interactive rate, the application need to generate a suitable collision response. The central problem in collision response is the computation of collision impulses [42]. Accurate computation of impulses arising between colliding, sliding, rolling bodies is critical to the physical accuracy of the impact response.

There are two major approaches in computing collision impulses: the constraint-based method and the impulse-based scheme. Each has its own advantages. Constraint-based approach [1, 2, 48] is suited to model the motion constraints imposed by other contacting bodies. A perfect example is modeling a hinge joint or linkage bodies (such as human motion dynamics). On the other hand, impulse-based method is simple and easy to implement. In addition, impulse-based approach does not need to keep track of contact states (resting, rolling, sliding, colliding, etc.) and it presents a unified computational model for computing the collision impulses for impact response.

Impulse-based dynamics [17, 37] assumes no explicit constraints on configuration of the moving objects. When the objects are not in contact, they are traveling in ballistic trajectory. Otherwise, all modes of contacts are modeled via series of impulses applied to the objects, whether they are bouncing around, sliding through or resting on top of each other. Obviously, this approach is very *collision intensive*, i.e. the simulator will need to check for possible contacts frequently. However, because of our extremely efficient algorithms for collision detection, it is now possible to compute the motion dynamics of rigid bodies in nearly real time on current personal PCs or graphics workstations [37].

7 Application Demonstration

We have tested the collision detection algorithms on numerous cases including: interaction with architecture walkthrough and mechanical CAD model of a power plant, dynamics of threaded screw insertion, interlocked toridal chain motion, bowling, non-holonomic

motion of a ball on a spinning platter, etc. In such a large-scale environment, we have thousands, even millions of complex objects and we need to simulation interaction among the user and the virtual world. The complexity of mega models poses a grant challenge to real-time collision detection and dynamic simulation. We have been collaborating with the Walkthrough Group at UNC Chapel Hill to integrate the system into their architecture walkthrough, a multibody simulation environment and walkthrough of a coal-fired powerplant. The preliminary results have shown great promises in achieving interactive collision detection on the current personal computer and graphics workstations. For more information and demonstration of our algorithms and systems, please refer to

http://www.cs.unc.edu/~geom/collide.html

for a catalog of videos, examples, and technical reports.

8 Public Domain Software Packages

Most of public domain systems are applicable to polygonal models and some are also applicable to large environments composed of multiple moving objects. It is nearly impossible to compare different algorithms and systems fairly, since their performance varies, depending on the simulation environments (models, varieties of contacts, query types, motion description, etc.) and other factors. Here we only list them in the chronological order of their release and briefly describe their special characteristics.

I-COLLIDE is an interactive and exact collision-detection library for environments composed of many convex polyhedra or union of convex pieces, based on the expected constant time, incremental distance computation algorithm [29, 28], linear programming and sweep and prune technique to check for collision between multiple moving objects [10]. It is available at

http://www.cs.unc.edu/~geom/I_COLLIDE.html

RAPID is a robust and accurate polygon interference detection library for pairs of unstructured polygonal models. It is applicable to *polygon soups*, i.e. models that contain no adjacency information and obey no topological constraints. It is most suitable for close proximity configurations between highly tessellated smooth surfaces [15] and available at

http://www.cs.unc.edu/~geom/OBB/OBBT.html

V-COLLIDE is a collision detection library for large dynamic environments [20], and unites the nbody processing algorithm of I-COLLIDE and the pair processing algorithm of RAPID. It is designed to operate on large numbers of static or moving polygonal objects to allow dynamic addition or deletion of objects between time steps and available at

http://www.cs.unc.edu/~geom/V_COLLIDE

Distance Computation between Convex Polytopes is an enhanced and dynamic version [7] of the distance routine of Gilbert, Johnson and Keerthi (GJK), which allows the tracking of the distance between a pair of convex polyhedra. It requires a list of all the edges in each convex polyhedra for best performance. Its performance is comparable to Lin-Canny [29, 28] convex polytope overlap test. It is available at

www.comlab.ox.ac.uk/oucl/users/stephen.cameron/

SOLID is a library for interference detection of multiple three-dimensional polygonal objects (including polygon soups) undergoing rigid motion and deformation specified by vertex arrays. Its performance is slightly slower based on some benchmarks that we have tested on and its applicability is comparable to that of V-COLLIDE. It is available at

http://www.win.tue.nl/cs/tt/gino/solid/

V-Clip, or the Voronoi Clip, algorithm is a low-level collision detection algorithm for polyhedral objects [36] – an improvement of the closest-feature tracking algorithm using Voronoi regions [29, 28]. It operates on a pair of convex polyhedra, or nonconvex hierarchies of convex pieces. In addition to distance computation, it can also report estimated penetration points and estimated penetration distance between overlapping models. It is available at

http://www.merl.com/people/mirtich/vclip.html

QuickCD is a general-purpose collision detection library, capable of performing fast and exact collision detection on highly complex models. QuickCD can handle unstructured inputs consisting of a soup of polygons. It is based on hierarchies of k-dops [21, 22] and available at

http://www.ams.sunysb.edu/ jklosow/quickcd/

PQP is a library for performing three types of proximity queries on a pair of geometric models consisting of polygon soups, using OBBTreees and a family of bounding volumes called *swept sphere volumes*. The queries are collision detection, distance computation and tolerance verification [26]. It is available at

http://www.cs.unc.edu/~geom/SSV

9 Future Work

Despite abundant wealth of the literature in collision detection, there are several open research issues. Much remains to be done on detecting contacts between deformable models accurately and efficiently. Although an algorithm based on bounding volume hierarchies can be used to check for collision between two deformable models by recomputing the bounding volumes given modified vertex coordinates; it is, however, unclear if such an approach is the most efficient and suitable for large deformation. Many applications call for the development of algorithms for fast distance computation between general geometric models, possibly by utilizing pre-computed bounding volume hierarchies or adaptive hybrid hierarchies which consist of different types of bounding volumes [26]. In dynamic simulation, computing collision response requires robust and interactive computation of the closest features or contact points between general geometric models, as well as rapid calculation of penetration distance. This problem is especially difficult for those models with smooth surfaces and many concavities. Integration of geometric, numerical and analytical methods is probably necessary to design solutions to address these problems. There are also new challenges in applying collision detection algorithms to haptic rendering [16] or proximity queries on massive models [47], which consist of millions of primitives and are often too large to fit in the main memory. These may include developing external memory algorithms, dynamic pre-fetching techniques, SIMD implementation or parallel computing methods for collision detection.

Acknowledgments

We would like to thank Dinesh Manocha, John Canny, Jonathan Cohen, Stefan Gottschalk, Shankar Krishnan, Gopi Meenakshi, Brian Mirtich, Amol Pattekar, Krish Ponamagi and the Walkthrough Group at the University of North Carolina, Chapel Hill for their collaboration and the architecture model for demonstration. We are also grateful to ARO, DARPA, Ford, Honda, Intel Corporation, NSF, ONR, Sloan Foundation and University of North Carolina at Chapel Hill for their research funding support over the years.

References

[1] Baraff, D. (1989). Analytical Methods for Dynamic Simulation of Non-Penetrating Rigid Bodies, ACM Computer Graphics, 23 (3): pp. 223-232, July 1989.

[2] Baraff, D. (1990). Curved Surfaces and Coherence for Non-Penetrating Rigid Body Simulation, ACM Computer Graphics, 24 (4): pp. 19-28, 1990.

[3] Barequet, G., Chazelle, B. Guibas, L, Mitchell, J. and Tal. A (1996). Boxtree: A hierarchical representation of surfaces in 3d. in the Proceedings of Eurographics'96.

[4] Bouma, W. and Vanecek, G. (1991). Collision detection and analysis in a physically based simulation, in the Proceedings of Eurographics workshop on animation and simulation, pages 191–203, 1991.

[5] Cameron, S. (1990). Collision detection by four-dimensional intersection testing, in the Proceedings of IEEE International Conference on Robotics and Automation, pages 291–302, 1990.

[6] Cameron, S. (1991). Approximation hierarchies and s-bounds, in the Proceedings of ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, pages 129–137, Austin, TX, 1991.

[7] Cameron, S. (1997). A comparison of two fast algorithms for computing the distance between convex polyhedra, IEEE Transactions on Robotics and Automation, 13(6):915–920, December 1997.

[8] Cameron, S. and Culley, R. K. (1986). Determining the minimum translational distance between two convex polyhedra, in the Proceedings of IEEE International Conference on Robotics and Automation, pages 591–596, 1986.

[9] Canny, J. F. (1986) Collision detection for moving polyhedra, IEEE Transactions on PAMI, 8:200–209, 1986.

[10] Cohen, J., Lin, M., Manocha, D. and Ponamgi, M. (1995). I-Collide: An interactive and exact collision Detection system for large-scale environments, in the Proceedings of ACM Interactive 3D Graphics Conference, pages 189–196.

[11] Cremer, J. F. and Stewart, A. J. (1994). The Architecture of Newton, A General Purposed Dynamic Simulator, Proceedings of the IEEE International Conference on Robotics and Automation, May 1994.

[12] Dobkin, D. P. and Kirkpatrick, D. G. (1990). Determining the separation of preprocessed polyhedra – A unified approach, in the Proceedings of the 17th Internat. Colloq. Automata Lang. Program., V. 443 of Lecture Notes in Computer Sciences, pages 400–413. Springer-Verlag.

[13] Garcia-Alonso, A., Serrano, N., and Flaquer, J. (1994). Solving the Collision Detection Problem, IEEE Computer Graphics and Applications, 13 (3): pp. 36-43, 1994.

[14] Gilbert, E. G., Johnson, D. W. and Keerthi, S. S. (1988). A fast procedure for computing the distance between objects in three-dimensional space, IEEE J. Robotics and Automation, Vol RA-4:193–203.

[15] Gottschalk, S., Lin, M. and Manocha, D. (1996). OBB-Tree: A Hierarchical Structure for Rapid Interference Detection, the Proceedings of ACM SIGGRAPH'96, pp. 171-180, New Orlean, LA, 1996.

[16] Gregory, A. Lin, M. Gottschalk, S. and Taylor, R. (1999). H-COLLIDE: A Framework for Fast and Accurate Collision Detection for Haptic Interaction, the Proceedings of IEEE VR'99, pp. pp. 38-45, March 1999.

[17] Hahn, J.K. (1988). Realistic Animation of Rigid Bodies, ACM Computer Graphics, 22 (4): pp. 299-308, 1988.

[18] Hoffmann, C. M. (1989). Geometric and Solid Modeling, Morgan Kaufmann, San Mateo, California.

[19] Hubbard, P.M. (1993). Interactive Collision Detection, Proceedings of IEEE Symposium on Research Frontier in Virtual Reality, October 1993.

[20] Hudson, T., Lin, M., Cohen, J., Gottschalk, S., and Manocha, D. (1997). V-collide: Accelerated collision detection for VRML. In the Proceedings of ACM Symposium on VRML, pages 119–125, 1997.

[21] Klosowski, J., Held, M. and Mitchell, J. S. B. (1996). Real-Time Collision Detection for Motion Simulation within Complex Environments, ACM SIGGRAPH Visual Proceedings, p.151, 1996.

[22] Klosowski, J., Held, M., Mitchell, J. S. B., Sowizral, H. and Zikan, K. (1998). Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs, IEEE Trans. on Visualization and Computer Graphics, v. 4-1: p. 21-37, 1998.

[23] Krishnan, S., Gopi, M., Lin, M. C., Manocha, D. and Pattekar, A. (1998). Rapid and Accurate Contact Determination between Spline Models using ShellTrees, in the Proceedings of Eurographics'98, 1998.

[24] Krishnan, S. and Manocha, D. (1997). An efficient surface intersection algorithm based on the lower dimensional formulation, ACM Transactions on Graphics, 16(1): p.74–106, 1997.

[25] Krishnan, S., Pattekar, A., Lin, M. and Manocha, D. (1998). Spherical shell: A higher order bounding volume for fast proximity queries, Proceedings of the Third International Workshop on Algorithmic Foundations of Robotics, 1998.

[26] Larsen, E., Gottschalk, S., Lin, M. and Manocha, D. (1999). Fast Proximity Queries with Swept Sphere Volumes, Technical report TR99-018, Department of Computer Science, University of N. Carolina, Chapel Hill, 1999.

[27] Latombe, J.-C. (1991). Robot Motion Planning, Kluwer Academic Publishers.

[28] Lin, Ming C. (1993). Efficient Collision Detection for Animation and Robotics, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1993.

[29] Lin, Ming C. and Canny, J. (1991). Efficient algorithms for incremental distance computation, Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1008–1014, May 1991.

[30] Lin, M. C. and Gottschalk, S (1998). Collision Detection between Geometric Models: A Survey, in the Proceedings of IMA Conference on Mathematics of Surfaces.

[31] Lin, M. C. and Manocha, D. (1995). Fast interference detection between geometric models, the Visual Computer, 11(10): pp. 542–561, 1995.

[32] Lin, M. C., Manocha, D. and Ponamgi, K. (1994). Fast contact determination between general polyhedral models, in the Proceedings of IEEE International Conference on Robotics and Automation, Vol. 4, pp. 602-208, May 1994.

[33] Manocha, D. and Demmel, J. (1994). Algorithms for intersecting parametric and algebraic curves I: simple intersections, ACM Transactions on Graphics, 13(1): p.73–100, 1994.

[34] Manocha, D. and Demmel, J. (1994). Algorithms for intersecting parametric and algebraic curves II: multiple intersections, Computer Vision, Graphics and Image Processing: Graphical Models and Image Processing, pages 81–100, 1995.

[35] Megiddo, N. (1983). Linear-time algorithms for linear programming in R3 and related problems, SIAM J. Computing, 12:pp. 759–776, 1983.

[36] Mirtich, B. (1998). V-Clip: Fast and Robust Polyhedral Collision Detection, ACM Transactions on Graphics, 17(3), pp. 177–208, 1998.

[37] Mirtich, B. and Canny, J. (1995). Impulse-Based Simulation of Rigid Bodies, ACM Symposium on Interactive 3D Graphics, pp.181-188, 1995.

[38] Moore, M. and Wilhelms J. (1988). Collision Detection and Response for Computer Animation, ACM Computer Graphics, 22 (4): pp. 289-298, 1988.

[39] Naylor, B., Amanatides, J. and Thibault, W. (1990). Merging BSP trees yield polyhedral modeling results, in Proceedings of ACM SIGGRAPH, 1990, pp. 115–124.

[40] Preparata, F.P. and Shamos, M.I. (1985). Computational Geometry, Springer-Verlag, New York.

[41] Quinlan, S. (1994). Efficient distance computation between non-convex object, in the Proceedings of IEEE International Conference on Robotics and Automation, pp. 3324–3329, 1994.

[42] Routh, E. J. (1905). Elementary Rigid Dynamics.

[43] Samet, H. (1989). Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods, Addison Wesley.

[44] Seidel, R. (1990). Linear programming and convex hulls made easy, in the Proceedings of 6th Ann. ACM Conf. on Computational Geometry, p.211–215.

[45] Thibault, W. and Naylor, B (1987). Set Operations on Polyhedra Using Binary Space Partitioning Trees, ACM Computer Graphics, 4, 1987.

[46] Wang, Y. and Mason, M. (1987). Modeling Impact Dynamics for Robot Operations, Proceedings of the IEEE International Conference on Robotics and Automation, pp. 678-685, May 1987.

[47] Wilson, A. Larsen, E. Manocha, D. and Lin, M. (1999). Partitioning and Handling Massive Models for Interactive Collision Detection, in the Computer Graphics Forum, September 1999.

[48] Witkin, A., Gleicher, M., and Welch, W. (1990). Interactive Dynamics, ACM Computer Graphics, 24 (2): pp. 11-22, March 1990.