Collision Detection for Interactive Environments and Simulations

Madhav K. Ponamgi
Department of Computer Science
University of North Carolina

Abstract: We propose a new collision detection algorithm for polygonal models in interactive environments and simulations based on incremental computations. The algorithm will be implemented as a system that will provide collision detection for large-scale geometric data sets modeled from primitives such as polygons and polytopes. The overall goal is to develop a demonstrable system that can be used with present-day animation and simulation environments.

Contents

1 Thesis Statement

An incremental computation based paradigm can be effective for exact collision detection for non-convex polyhedral models in simulations and interactive virtual environments. The algorithm's performance in environments consisting of hundreds of objects will prove its effectiveness. The implemented algorithm will detect when objects are colliding as well as determine their precise contact points.

2 Introduction

Computer graphics based interactive applications and simulations of physical systems strive to present realistic animations. For them to be convincing, they need to not only render realistic images, but also precisely model object interactions in the simulated environment. The interactions may involve objects in the environment pushing, striking, or deforming other objects. *Detecting collisions and determining contact points* is of fundamental importance to portray these interactions accurately. This thesis proposal discusses the open problems involved in collision detection and proposes a line of investigation to solve them.

The simulated environments we are concerned with can be classified into two broad categories: interactive user applications and simulations. In an interactive application, the user is placed in the simulated environment like an architectural walkthrough via a screen or head-mounted display. The user can navigate around this environment. The user can choose to simply be an observer, or can become an active participant by altering the objects in the environment by pushing, pulling, or acting on them in some fashion. In either case, to give the user a sense of presence, the images and interactions need to be portrayed realistically. Collision detection plays a central role in this: solid objects should not pass through or penetrate each other; soft and hard object collisions should result in deformations; and object collisions in general should result in realistic reactions from the exact point of contact. For the user to see images moving smoothly, the collision detection and reaction need to be computed quickly.

Collision reaction is generally application specific, and can involve force feedback, sound, visual effects, etc. This thesis proposal addresses only the collision detection component. We aim to do the collision detection component efficiently and pass this information onto the application so it can compute an appropriate response.

Simulations of physical systems have similar requirements in terms of collision detection. A simulation of an experiment that is too costly or impractical to build can be a significant aid in the experimental process. A threaded screw insertion[28], an automobile crash test [20], or a building demolition [20] are potential examples of where a simulation of the events is useful. However, for such simulations to be of use, they need to model the dynamic interactions of the experiment with a degree of realism. Collision detection has been identified as a major component of the interactions in these simulations [28, 20], so efficiently dealing with it would have practical implications.

We claim interactive applications and simulations of physical systems are representative of a large class of simulated environment based applications. As a result, any techniques developed to enhance their performance are widely applicable. Hence, accurate and efficient collision detection is of importance in a wide range of graphics applications.

3 Previous Work

Collision detection has had a long history of development in computer graphics, animation, robotics, geometric modeling, and computational geometry. Because collision detection is needed in a wide variety of situations, many different methods have been proposed. Each method makes specific assumptions about the environment and designs a solution around them.

3.1 Previous approaches

Computational geometry literature has several references dealing with the intersection detection of convex polygons and their extension to 3-D, convex polytopes [9]. A convex polytope is the closed intersection of half-planes. For example, a cube with all six faces is a convex polytope. However, a cube with one side or more missing is *not* a polytope since it is not "closed". Many of the algorithms dealing with convex polytopes, although mathematically elegant, are extremely difficult to implement. The best known running time for detecting if two convex polytopes intersect is $O(\log^2 n)$, and it can be shown that the lower bound for this problem is $O(\log n)$ [7, 10]. This algorithm simply detects if there is an intersection, but does not explicitly construct the intersection region.

Collision detection for objects constructed using Bezier patches, B-splines, NURBS, etc. falls generally into the approximate category [25]. Finding exact collisions between objects modeled out of patches is equivalent to finding whether two functions intersect. If we assume the function determining the intersection is a polynomial, the closed form solution can be found only if its degree is strictly less than five. Otherwise, iterative methods must be used. Often times such methods make use of a series of bounding boxes or convex hulls to approximate the patch functions. The approximating volumes are refined to the resolution desired, and if at the final level they overlap, a collision is indicated. Methods based on "interval arithmetic" take a similar approach in finding iterative solutions; they solve constrained minimization problems to determine areas of contact [31].

Most computational geometry literature deals with collision detection (often referred to as interference detection) of objects in a static frame. Objects are at a fixed location and orientation, and the algorithms determine whether they are interfering. In computer graphics applications, where many objects are in motion, a dynamic approach would be better: Objects move only slightly from frame to frame and the collision detection scheme should take advantage of the previous frame's information to compute in the current frame whether objects intersect[4]. Several solutions based on this idea of coherence have been proposed. The one by Lin and Canny has been shown to be highly effective for convex polytopes [24].

Robotics literature deals with collision detection in the context of path planning. Using sophisticated mathematical tools, several exponential running time algorithms have been developed that plan collision-free paths for a robot for restricted environments [6]. The exponential running time makes these algorithms impractical for complex simulated environments.

Collision detection for multiple moving objects has recently become a popular research topic due to increased interest in large-scale environments. For example, the human participants may walk through a building where the tables, chairs, etc. remain stationary. In such an environment, there are N moving objects and M stationary objects. Each of the N moving objects can collide with the other moving objects, as well as the stationary ones. Keeping track of $\left(\begin{array}{c} N \\ 2 \end{array}\right) + NM$ pairs of objects at every time step can become time consuming as N and M get large. Typically the number of collisions is linear in the number of objects involved, so an output sensitive algorithm that scales with the complexity of the problem is desirable.

This multiple body problem is a simpler version of the "N-body" problem faced in gravitation and radiosity calculations [15]. In N-body calculations, an object is influenced by other objects in the environment without being in contact with them. In our multiple body problem, objects alter each other's path only when they come in contact (either directly or indirectly through other objects).

A common approach to the multiple body problem uses bounding volumes. Objects are surrounded by tight fitting bounding volumes. If the volumes overlap, indicating that the objects are near each other, a more precise collision test is computed. The precise collision detection algorithms generally deal with collision detection between convex polytopes[30].

The bounding volume approach is frequently used to deal with non-convex objects as well. An object is broken down into convex pieces, and some type of a hierarchical structure is constructed out of the sub-pieces. To determine whether two non-convex objects intersect, the hierarchy is efficiently traversed and collision detection between the convex sub-pieces is computed. Using techniques similar to these, space partitioning has also been used to deal with multiple objects [31]. Space is partitioned into cells, and objects sharing the same cells are tested for intersection.

There has been work that uses a combination of all these approaches. One recent approach by Hubbard trades time for accuracy [21]. The fundamental problem with this approach is that exact collision points are almost never computed. His algorithm also does not generalize to arbitrary closed polygonal objects.

3.2 State of the Art

For the types of applications and simulations we are considering, the state of the art in collision detection can be improved in several ways. We aim to do exact collision detection for polygonal models based on incremental computation. The state of the art either lacks the generality for the models we propose to use or is not efficient enough to use. We summarize the algorithms from above in the state of the art as follows:

- Computational geometry algorithms to solve static interference and intersection between pairs of convex polytopes [7, 9, 8]. These algorithms are only for convex polytopes. They also do not address the problem of multiple objects moving simultaneously.
- Collision-free path planning algorithms in robotics [6]. These algorithms expect motion as a closed form function of time, which is not possible in interactive environments where the user's behavior is unpredictable.
- Approximate collision detection schemes. We are aiming for exact collision detection, at least to the resolution of the models and the input devices.
  - Bounding box based algorithms [2, 19, 18]. These algorithms decompose the objects into tiny cubes (called voxels). Collision detection is done by finding objects with overlapping voxels. The collision accuracy in these techniques is not to the resolution of the models. They also do not exploit the frame to frame coherence that exists in most graphics applications and as a result their published collision detection time is high [18].

  - Sphere based algorithm [21]. This method cannot deal with arbitrary non-convex polygonal models. It cannot construct a medial axis (an exact internal Voronoi region) for an arbitrary 3-D polygonal object, which it uses for collision detection. In such situations, it does only approximate collision detection.
  - Root finding algorithms [14, 16]. These apply only to convex objects and these techniques do not generalize to non-convex polygonal models.

- Space subdivision methods [32, 31]. These methods are targeted towards specific applications, such as molecular docking. Space subdivision methods become a bottleneck when the objects become non-uniformly distributed during the course of a simulation. In some sense, the algorithm we propose can be considered an adaptive space subdivision approach.

- Exhaustive all possible feature pair testing algorithms [5, 26]. These techniques exhibit poor performance times as the complexity of the models increase. Hence, they are unsuitable for complex simulations or interactive applications that we are considering.

- Incremental algorithms for exact collision detection between pairs of convex polytopes [4, 24]. These ideas have been proven to be effective for convex models. We plan to extend these ideas to non-convex polygonal models.

None of the algorithms in the state of the art describe an efficient collision detection method for general polygonal models in interactive environments. It can be shown that each of the approaches either lacks the generality to handle non-convex polygonal objects or does not effectively deal with the interactive nature of the applications and simulations we are considering.

4 Overview of Our Approach

We propose a method based on *coherence* to find the *exact contact and collision points* between non-convex polygonal objects.

Definition: *Temporal and geometric coherence* is the property that the application state does not change significantly between successive time steps or frames. The objects move only slightly from frame to frame. This slight movement of the objects translates into geometric coherence, because their geometry, defined by the vertex coordinates, changes minimally between frames. The underlying assumption is that the time steps are small enough that the objects do not travel large distances between frames.

In many situations, assuming coherence is reasonable [4, 30]. Further, its assumption can lead to highly efficient incremental algorithms [24, 30]. The worst case running time for these algorithms is still $O(n^2)$ where n is the number of features involved, but evidence suggests this case rarely occurs [30, 24, 23].

Using coherence as our central theme, we present an incremental algorithm for collision detection between non-convex polygonal objects. The algorithm description consists of three phases: pre-processing, contact determination algorithm, and implementation. There are several new computational problems that are addressed in the pre-processing stage. The following is a brief list of the algorithmic sub-problems the overall effort addresses. We will go into details in the next section.

- Identifying for each non-convex object the convex, concavity, and boundary features.
- Computing a "cap" over each set of concavity features.
- Detecting penetration between two convex polytopes using pseudo-internal Voronoi regions using an incremental algorithm.
- Detecting which caps are colliding when two convex polytopes inter-penetrate.
- Constructing a hierarchical structure on each concavity for exact collision detection.
- Constructing envelopes over each concavity for exact collision detection as an alternate method to hierarchical collision detection.

Our approach to collision detection for non-convex objects builds on our previous work for multiple moving objects [30]. There we presented a scheme for efficient collision detection for hundreds of moving complex convex objects. The goal is to apply a similar technique again, treating a pair of non-convex objects as many convex objects interacting.

This approach though differs from a convex decomposition of the objects and finding collisions between all convex sub-pieces. Convex decompositions are not good for several reasons: an optimal decomposition is NP-complete [13]; even a simple decomposition is expensive to compute; and a decomposition introduces many spurious edges and faces to an object.

For each non-convex object, our technique begins by statically computing its convex hull and a hierarchical representation of each of its concavities. During the dynamic phase we use our previous work to determine when pairs of convex hulls are interacting [30]. Using an extension of this work, we determine when and where the convex regions of a pair of hulls are intersecting. Using these regions, we use the hierarchical representation of each polytope to find the exact contact points. This process is fully explained in the last two appendices.

4.1 Demonstration of Thesis

The algorithm will be shown as efficient as well practical. To demonstrate its usefulness we plan to show it working in several complex environments and simulations.

- The new house model.
- Walkthrough of house, submarine and automobile models.

- Insertion of threaded screws.
- A multi-body simulation of a ring of tori.

These datasets are representative of real applications that are used in simulated environment applications. Developing a collision detection system that works with them provides a thorough test of the algorithms. Success on these datasets will ensure that the ideas are generally applicable.

In addition to demonstrations, we plan to make performance measurements. We will make direct comparisons to relevant work such as by Philip Hubbard [21] and others.

4.2 Contributions

The main contribution is an incremental-computation based collision detection algorithm to compute exact contact points between non-convex polygonal models. There are a number of interesting sub-problems that are resolved while addressing the main goal.

- An $O(n \log n)$ algorithm for computing a non-convex object's 3-dimensional convex, concavity, and boundary features.
- An $O(n^2)$ algorithm to compute 3-dimensional caps over an object's concavity features.
- An incremental algorithm based on pseudo-internal Voronoi regions to compute when and which caps are intersecting between two convex polytopes.
- A coherence based hierarchical bounding-volume algorithm to compute contact points between colliding features lying beneath caps.
- An algorithm for computing envelopes over a concavity feature set.
- An algorithm to compute exact contact points using the envelope data structure and pseudo-internal Voronoi regions.
- An implementation that can be used as a collision detection library.

4.3 Schedule

- Thesis proposal – Spring 1995.
- An algorithm to compute boundary loops and caps – December 1994.
- Take the oral examination – April 1995.
- Implementation of an algorithm for loops, caps, and penetration detection – March 1995.
- Implementation of contact determination algorithm – June 1995.

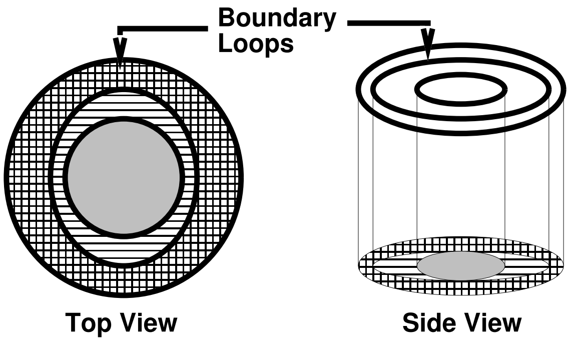

Figure 1: Nested boundary loops.

- Demonstration of thesis tasks – November 1995.
- Defend thesis – Spring 1996.

5 Appendix 1: Details of Our Approach

In this section the details of the collision detection algorithm are described. Several data structures are constructed in the pre-processing stage.

5.1 Classifying Features

Each non-convex object is described as a set of faces; each face consists of a list of vertices. The faces, edges, and vertices comprising an object are called its features. Our first step in pre-processing is to categorize these features. Taking the convex hull of the object separates its features into three distinct regions. The portions of the object that are coincident with the faces of the convex hull are identified as the *convex features* of the object. The portions that do not lie on the convex hull are identified as the *concavity features* of the object. The vertices bordering the convex and concavity features are the boundary features. The boundary features can form complicated 3-dimensional loops and be arbitrarily nested within each other (The rims of a series of connected cylindrical shells of the same height but decreasing diameter that are nested within each other form a nested collection of boundary loops as shown in Figure 1.).

We identify these regions using a breadth-first search technique. Any face that is part of the convex feature set has to have all of its vertices on the convex hull. Using this observation, we begin with a vertex belonging to the convex feature set and determine if any of its neighboring vertices also lie on the convex hull. This operation is repeated recursively. Applying this for an arbitrary object results in disjoint sets of convex features.

In a similar fashion, the concavity features can be identified. Any face that has one or more vertices *not* on the convex hull, is part of a concavity. Here too, multiple disjoint concavities can result. For example, a sphere with multiple dimples or dents (like a golf ball) is an object with multiple disjoint concavities.

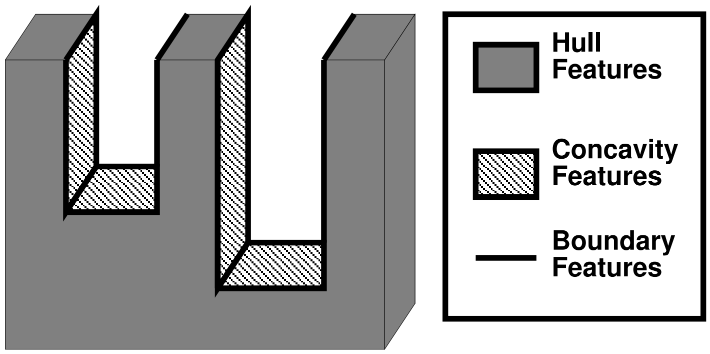

Figure 2: Example of convex, concavity, and boundary features.

Boundary features are formed by the common intersection of convex and concavity features. They can be identified by looking at the faces of a set of convex features which border a concavity (i.e. all faces of a convex feature set adjacent to concavity feature set). So any face that is part of a concavity, but has at least one vertex on the convex hull is part of the boundary. Figure 2 is a simple example showing all three of these regions.

5.2 Computing Caps

We want to compute a cap over each concavity feature set using its boundary features. Multiple caps need to be computed because an object may be composed of disjoint convex and concavity feature sets. The dimpled sphere described earlier requires multiple caps because it consists of multiple concavity feature sets connected by a single convex feature set.

Computing a 3-dimensional cap for a general non-convex polygonal object is a difficult task. There has been some work done on computing 2-dimensional caps (caps for non-convex polygons) [22]. An $O(n^3)$ algorithm for simple 3-dimensional objects was published [3], but an efficient algorithm for the general problem would be a new research result. Figure 3 shows an example of a cap. The problem is difficult because of several reasons:

- The boundary loop of a concavity can be a 3-dimensional structure. The cap for such a structure has to cover multiple sides of the object.
- The boundary loop can have holes.
- The boundary loop can be self-intersecting and can be arbitrarily nested.

Implementing a general cap computation algorithm can be difficult. Problems with co-planar vertices, narrow faces, nested rings, etc. make it a difficult problem. The problem can be compared to computing convex hulls since the caps represent the portions "missing" from an object to make it convex. This problem has been shown to be quite difficult to implement [29].

We have implemented an $O(n \log n)$ cap computation technique that handles simple 3-dimensional loops with no holes. The idea uses Meister's theorem to compute a cap by

Figure 3: Example of cap computation.

triangulating its boundary region [29]. Subtracting the original faces of the object from the convex hull we can construct the caps in a more efficient manner. An efficient cap computation is necessary, because if the collision response is a deformation, the cap needs to be recomputed. The classification of the objects' features and cap(s) computation ends the pre-processing of the data set. There are several geometric degeneracies that can occur in the data set, these are addressed in section 6.

5.3 Detecting Penetration with Pseudo-Internal Voronoi Diagrams

After the pre-processing stage, an object's *convex features + caps* form its convex hull. The penetration status of a pair of non-convex objects is determined by a hierarchical algorithm. Initially, using the result of our earlier algorithm [30], we determine if the bounding volumes surrounding a pair of objects are overlapping. If the bounding volumes are overlapping, we check to see whether the objects' convex hulls are intersecting.

Detecting when two convex polygonal objects are intersecting using a coherence based method is a new result of ours [23]. In the previous Lin-Canny algorithm [24], the exterior of the polytope was partitioned. Then an algorithm to "walk" across the the features of the polytopes was defined that computes the closest features between the objects as long as they are *not* intersecting. To detect an inter-penetration condition, the internal region of each polytope is also partitioned using pseudo-internal Voronoi diagrams. Now the closest feature walk can traverse inside the objects, and the inter-penetration status between pairs of objects can be detected. We plan to extend this result to determine the precise intersection points of the two convex objects (i.e. which caps are involved in the intersection). A coherence based algorithm to do this would also be a new result, because the existing results handle the static version of the problem [8].

If the convex hulls are intersecting, we enter another layer to determine exactly which

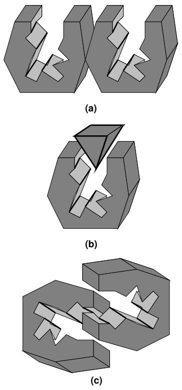

Figure 4: Non-convex objects interacting.

features are colliding. The collision between the objects' the convex hulls has three cases which can occur multiple times:

1. The collision is between convex feature sets on both the objects. This is a "real" collision, and there is no need to traverse any hierarchy.

2. The collision is between a cap of one object and a convex feature set of the other object (See Figure 4 with a pyramid entering a cavity). We want to determine if the convex feature set collides with any of the faces underneath the cap belonging to the cavity. For this case we proceed with another layer of the hierarchical collision detection.

3. The collision is between caps of both objects. We want to determine which features underneath the caps may be involved in a collision.

Each concavity feature set is represented as a bounding box hierarchy. The leaves of the hierarchy are the bounding boxes of the individual faces. For efficiency, multiple adjacent faces are grouped together in a single leaf node (the number of faces in a single leaf node can be adjusted). The hierarchy is constructed using an octree or a vertex space partition tree algorithm. At the top-level, the entire concavity is represented as a single bounding box. We need to decide how many cuts along each axis we will make to subdivide the box further. For example, splitting the box in half along the x, y, and z axis results in eight children for the top-level box. The subdivision of the child boxes continues recursively until the threshold number of faces is reached in the leaf boxes. We plan to experiment with various configurations for this bounding box hierarchy. A robust implementation for arbitrary polygonal models can be difficult because the cutting planes used to construct the hierarchy introduce new faces on the object. Properly constructing and keeping track of all these faces introduced by these space-subdivision planes requires significant special case coding. In addition, as the object moves this axis-aligned hierarchy needs to be updated efficiently.

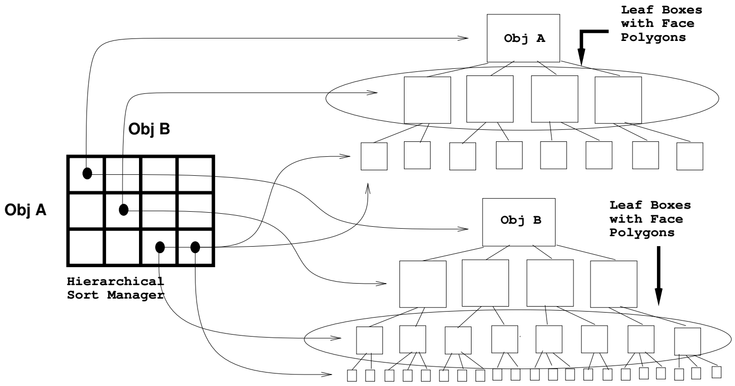

Figure 5: Two object hierarchies.

5.4 Hierarchically Computing Exact Contact Points

When an object penetrates a cap, we want to determine if it collides with any faces of the concavity beneath the cap. Because the penetrating object is also represented as a hierarchy of bounding boxes, our goal is to efficiently traverse the two hierarchies and determine *exact collision points* between the two non-convex objects. We do this by re-applying the *Sweep and Prune* algorithm from [30] on the hierarchies.

Initially, we begin with the roots of the hierarchies and determine which of the children overlap. Determining overlap status is done via sorting the bounding boxes as described in the *Sweep and Prune* algorithm. Sorting and determining children's overlap status is done recursively until we reach the leaf nodes of both hierarchies. At this last level, we check to see if the the faces of the relevant leaf nodes are intersecting. An intersection of one or more faces belonging to leaf nodes means a collision. The exact contact points are computed by finding the intersection of the polygons and passed to the application to plan an appropriate response. An example of hierarchies is shown in Figure 5.

The level by level sorting of the hierarchies' bounding boxes is tricky to manage both in terms efficiency and memory usage for objects composed of thousands of polygons. We wrote a bounding box management scheme that allocates the space for data once, but can have multiple pointers to it. So instead of allocating this data value at multiple levels in the hierarchy, it efficiently swaps pointers during the sort. It also allocates only as much memory per pair of objects as the number of features of the objects that are interacting (as opposed to allocating memory for all possible feature pairs for the objects). This is *critical* because an inefficient all pairs allocation quickly results in memory exhaustion.

The data structure also keeps the data from each sorting pass to use on the next pass. The previous frame's sorted bounding box lists, which indicate the objects' features' coordinates, usually change little from frame to frame, so the lists in the current frame, with the updated coordinates, should require only a minimal amount of swapping. We have implemented several simulations using this hierarchical data structure with good results. We plan to further refine this code.

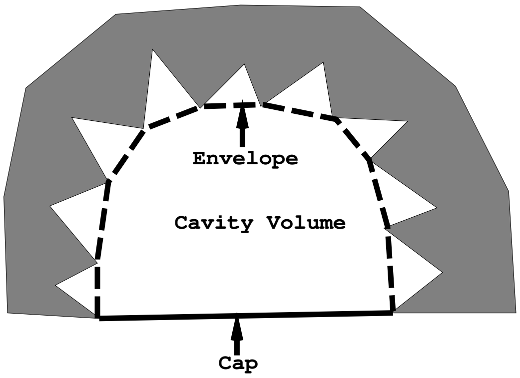

Figure 6: An Envelope of a 2-D Concavity.

5.5 Constructing and Using Envelopes

There are other ways to handle this last phase of the algorithm to determine exact contact points. One method we propose to explore involves a different type of hierarchical structure that uses pseudo-Voronoi diagrams used in penetration detection.

Each concavity can be covered by a simpler polygonal mesh. This "envelope" covers the entire concavity. The penetrating object tests for collisions with the envelope. If a collision has occurred with a feature from the envelope, the object then proceeds to test the features below the envelope's offending feature. This process can be carried on recursively with multiple layers of envelopes. Figure 6 shows an example of a one level *2-D* envelope.

There are several interesting aspects about this approach. The first concerns defining the criteria of what constitutes a "good" envelope. An envelope is required to do the following:

- Form a 3-D convex volume over the concavity's features. The object that has penetrated the concavity can be efficiently tracked only inside a convex volume using pseudo-Voronoi regions. The new tracking technique checks to see which pseudo-Voronoi regions the bounding box surrounding the penetrating object occupies. When the object's bounding box falls outside all the pseudo-Voronoi regions, this indicates a possible intersection with the envelope.
- While forming the 3-D convex volume, it must use the cap covering the opening to the concavity. Using this criterion we can determine when the object has left the concavity and begin re-using the *sweep and prune* higher-level collision detection algorithm. Using the cap also gives a starting point on constructing the envelope.
- The envelope should have polygonal facets that neatly partition the concavity into sub-cavities. When the object intersects the envelope's facet, these facets below are tested for collision. Limiting their number (exactly how many is open for research) would make this test more efficient.

Constructing the envelopes efficiently will be explored further. At the moment, our approach generates a series of slices that connected into a convex volume occupying a concavity. The slices are generated by sorting the vertex coordinates of the concavity and generated convex polygons at each level. This algorithm can be improved made more precise. This problem has not been found in the literature, so it would be an interesting geometric problem to pursue.

6 Appendix 2: Issues in Building a Collision Detection System

Implementing the algorithms outlined is a non-trivial task for several reasons. An implementation has to not only deal with many special geometric cases, but also has to worry about memory efficiency. The memory management problem described in the hierarchical bounding box data structure will have to be dealt with several times. Moreover, implementing the pre-processing and dynamic phases of the algorithm, combined with handling memory management issues, will add a great deal of complexity to the code. The basic implementation described earlier is well over $10,000$ lines and we expect it will become significantly larger.

In addition to implementation difficulties, we expect to encounter robustness problems related to geometric and numeric degeneracies. Implementing geometric algorithms has been shown to be a difficult task because of the many degeneracies that can occur [12, 17]. Our implementation will not handle every type of degenerate input, but we plan to handle common ones and be aware of the implications of bad input.

6.1 Geometric Degeneracies

Geometric degeneracies occur when the data specifying the geometry of the objects is flawed in some fashion. This condition immediately leads to problems in the pre-processing of the algorithm and later affects the dynamic phase. Some of the degeneracies we expect to encounter are:

- Coincident polygons. This affects the feature classification algorithm and leads to missed collisions.
- Co-planar polygons. In several parts of the overall algorithm, we make use of the convex hull of an object. It is a well-known problem that taking the convex hull of co-planar faces (hence vertices) causes numerical problems [29].
- Backfacing polygons. The Voronoi cone data structure for the external and internal walk used in the penetration detection phase breaks down in the presence of backfacing polygons.
- T-vertices. A vertex that rests on an edge but is not one of its endpoints affects the feature classification algorithm.

We expect to correct some geometric degeneracies such as coincident polygons and backfacing polygons by pre-scanning the data. However, we do not guarantee removing all types of errors in the data.

6.2 Numeric Degeneracies

Input data that is free of geometric degeneracies can still run into numerical problems. Floating point computations for geometric algorithms are susceptible to numerical errors [17]. There are a number of situations where the collision detection algorithm can run into floating point problems:

- Construction of the bounding box hierarchy. While grouping the faces into boxes, orthogonal cutting planes are used which cut the existing faces. These cuts can result in long thin faces which are difficult to represent in floating point arithmetic.
- Computing distances. A square root is used to decide which level of the hierarchal algorithm to invoke. Square root computations are prone to numerical error.
- Exact contact determination. We explicitly compute polygon intersections. Floating point computations can produce inexact results.
- Device problems. The tracking device used in the walkthrough is imprecise. The imprecise position location can lead to spurious collisions being detected.

We plan to introduce tolerances and perturbations when doing numeric computations. The perturbation methods attempt to randomnly distribute degenerate data (such as co-planar vertices) by small perturbations of the original data [12]. These reduce the possibility of error but do not eliminate it [17]. We do not plan to deeply pursue numerical analysis of geometric algorithms.

References

[1] P.K.Agrawal and M.Sharir. " Red-blue intersection detection algorithms with applications to motion planning and collision detection." SIAM J. Computing (1990) 19:297-321.

[2] A. Alonso, N. Serrano and J. Flaquer. "Solving the Collision Detection Problem" Computer Graphics and Applications (1994) pp. 36-43.

[3] A. Appel and P. Will. "Determining the Three-Dimensional Convex Hull of a Polyhedron" IBM Journal of Research and Development, Nov. 1976 pp. 590-600.

[4] D. Baraff. "Curved surfaces and coherence for non-penetrating rigid body simulations" ACM Computer Graphics (1990) 24(4):19-28.

[5] J. Boyse. " Interference Detection Among Solids and Surfaces" Communications of the ACM (1979) pp.3-9.

[6] J. Canny. " Collision Detection for Moving Polyhedra" IEEE Trans. on Pattern Analysis and Machine Intelligence (1986) pp.200-209.

[7] B. Chazelle and D. Dobkin. "Intersection of Convex Objects in Two and Three Dimensions" Journal of the ACM (January, 1987) pp.1-27.

[8] B. Chazelle. "An Optimal Algorithm for Intersecting Three-Dimensional Convex Polyhedra" IEEE Symposium on Found. of Comp. Sci 30 (1989) pp.586-591.

[9] B. Chazelle and D. Dobkin. "Detection is Easier than Computation" Annual ACM Symposium on Theory of Computing (1980) pp.146-153.

[10] D. Dobkin and D. Kirkpatrick. "Fast Detection of Polyhedral Intersection" Theoretical Computer Science (1983) pp.241-253.

[11] P. Dworkin and D. Zeltzer. "Determining the separation of preprocessed polyhedra – a unified approach" Proceedings of the Fourth Eurographics Workshop on Animation and Simulation (1993).

[12] I.Emiris and J.Canny "A general approach to removing degenracies" Proceedings of 32nd Annual Symp. Found. of Computer Science, 1991, pp. 405-413.

[13] M. Garey and D. Johnson. "Computers and Intractability" W.H. Freeman and company, New York (1979)

[14] E. Gilbert, D. Johnson and S. Keerthi. "A Fast Procedure for Computing the Minimum of a Quadratic Form on a Convex Set" IEEE J. Robotics Automation vol.4 (1988) pp.193-203.

[15] L. Greengard. "The Numerical Solution of the N-Body Problem" Computers in Physics, 1990, pp. 143-152.

[16] E. Gilbert and C. Foo. "Computing the distance Between General Convex Objects in Three-Dimensional Space" IEEE Transactions on Robotics and Automation 6 (February, 1990) pp.53-61.

[17] S. Fortune and C. van Wijk. "Numerical stability of algorithms" 7th Annual ACM Symp. on Computational Geometry, pp. 334-341.

[18] J. Hahn. "Realistic Animation of Rigid Bodies" ACM SIGGRAPH 1988 pp. 299-308.

[19] B. Herzen, A. Barr and H. Zatz. "Geometric collisions for time-dependent parametric surfaces" Computer Graphics (1990) 24(4):30-48.

[20] J.E. Hopcroft. "Electronic Prototyping" IEEE Transactions on Aerospace and Electronic Systems, Vol. 24 (2): 663-667.

[21] P. Hubbard. "Interactive collision detection" Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality (1993) October.

[22] P. Loutrel. "Determination of Hidden Edges in Polyhedral Figures" Tech Report 400-145, Dept. of EE, New York University, September 1966.

[23] M. Lin, D. Manocha, and M. Ponamgi "Fast algorithms for penetration and contact determination between non-convex polyhedral models" Submitted for publication, September, 1994.

[24] M. Lin. "Efficient Collision Detection for Animation and Robotics" PhD Thesis, University of California at Berkeley, December 1992. Department of Electrical Engineering and Computer Science.

[25] M. Lin and D. Manocha. "Interference detection between curved objects for computer animation" Models and Techniques in Computer Animation (1993) pp. 43-57.

[26] M. Moore, J. Wilhelms. "Collision Detection and Response for Computer Animation" ACM SIGGRAPH (1988) pp. 289-298.

[27] J. Nevins and D. Whitney. "Concurrent Design of Products and Processes" New York, McGraw-Hill 1989.

[28] E. Nicolson and R. Fearing. "Compliant Control of Threaded Fastner Insertion" Int. Conf. on Intelligent Robots and Systems, Osaka, Japan, November 1991.

[29] J. O'Rourke. *Computational Geometry in C* Cambridge University Press, 1994.

[30] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. "Exact Collision Detection for Multi-Body Environments" Symposium on Interactive 3-D Graphics, Monterey, CA, April 1995.

[31] J. Snyder, A. Woodbury, K. Fleischer, B. Currin, and A. Barr. “Set operations on polyhedra using binary space partitioning trees” ACM Computer Graphics (1987) vol. 4.

[32] G. Turk. “Interactive collision detection for molecular graphics” Master’s thesis, Computer Science Department, University of North Carolina at Chapel Hill (1989).