

Efficient Collision Detection for Real-time Simulated Environments

by

Paul Jay Dworkin

B.S. Mathematics, Carnegie-Mellon University (1985)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Master of Science

at the

Massachusetts Institute of Technology

June 1994

© Massachusetts Institute of Technology, 1994

All rights reserved

Author _____
Paul Dworkin
Program in Media Arts and Sciences
February 4, 1994

Certified by _____
David Zeltzer
Principal Research Scientist
MIT Research Laboratory of Electronics

Accepted by _____
Stephen A. Benton
Chairperson, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Batch
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

JUL 13 1994

Efficient Collision Detection for Real-time Simulated Environments



by

Paul Jay Dworkin

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning

on February 4, 1994

in partial fulfillment of the requirements for the degree of
Master of Science



Abstract

This thesis describes a new model for rapid physical simulation called *sparse dynamics*. The method employs a simplified object representation to quickly identify likely object interactions. These are then flagged for more detailed analysis. As actual collisions are rare in a sparsely populated environment, efficiency is greatly increased. The first phase uses deterministic Newtonian mechanics to predict future collisions analytically, obviating the need to simulate small uniform time steps. The detailed phase makes use of a near-constant time polyhedral distance tracker to minimize the expense of complicated objects.

The current system can handle interactions involving linear and accelerated motions (including gravity), wind resistance, user manipulations, and to some extent supporting contacts and friction. Methods are discussed to further extend this list. Timings indicate sparse dynamics provides a large speed improvement over more traditional methods. In particular we are able to simulate full collision detections for a simple environment containing 500 polyhedra at real-time speeds. Even in highly realistic or complicated environments, we expect a speed up by an order of magnitude or more.

The code implementing the work described here is also made available for public use.

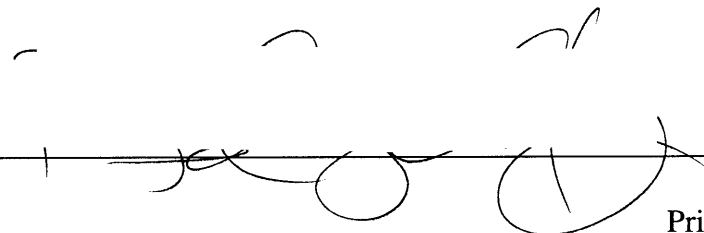
Thesis Supervisor: David Zeltzer

Title: Principal Research Scientist, Research Laboratory of Electronics

This work was supported in part by NHK (Japan Broadcasting Company)

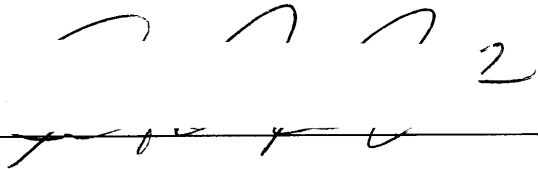
Thesis Committee

Advisor



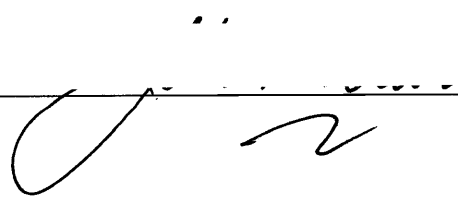
David Zeltzer
Principal Research Scientist
MIT Research Laboratory of Electronics

Reader



Alex Pentland
Associate Professor
MIT Program in Media Arts and Sciences

Reader



John Williams
Associate Professor
MIT Department of Civil Engineering

To


Carole Joan Meier

Howard Gerry Dworkin

Sine qua non

Table of Contents

Appendices

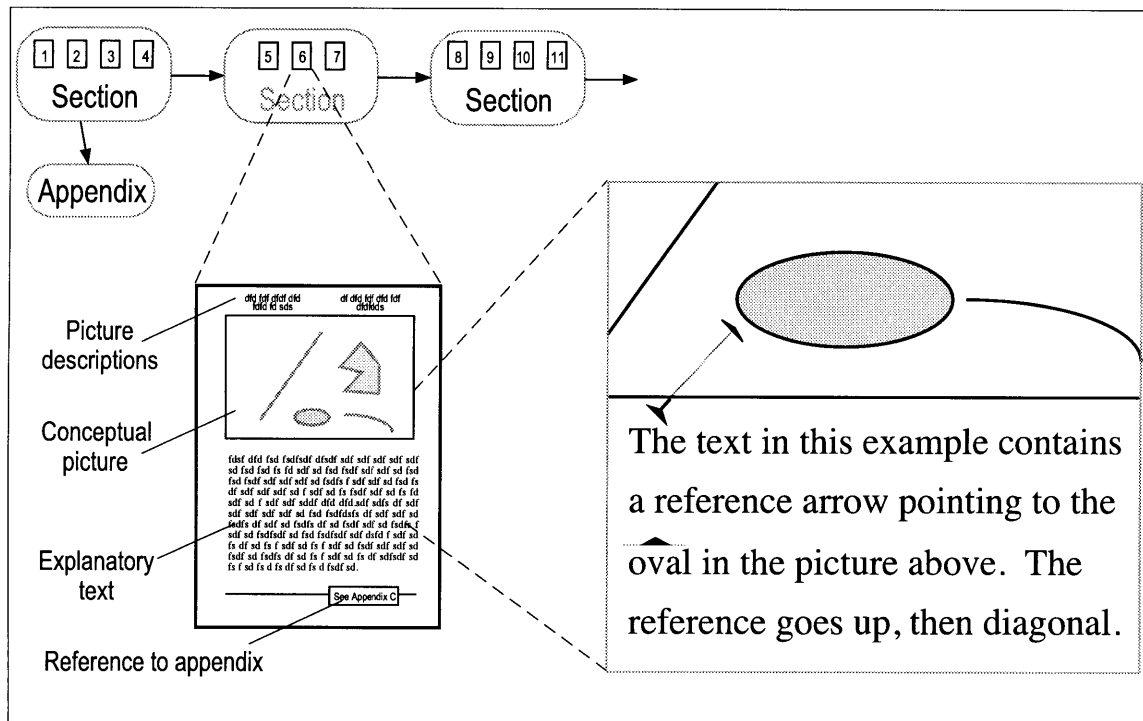
 Main Story	1. Thesis Overview 5	
	Thesis Format	→ A. Discussion of Formatting Issues 40
	2. Introduction 7	→ B. Notes on Virtual Reality 42
	Applications, Problem Context, Current State of the Art, Conventional Methods, Our Approach	→ C. Related Work 42
	3. The Sparse Dynamics Model 13	
	The Sparse Environment, A Sparse Element, Sparse Interaction, The Path Equation, Analytical Solutions, Extended Contact, Other Forces	→ D. Discussion of Sparsity 44
		→ E. Root Convergence 44
		→ F. Analytical Solutions 46
		→ G. Handling Extended Contact 48
		→ H. Adding Other Forces 50
	4. Sparse Event Handling 21	
	The Event Queue, Stale Events	→ I. Cost Analysis of the Sparse Pass ... 52
	5. Collision Detection 24	
	Lin and Canny Distance Calculation, Interface to the Sparse Model	→ J. L&C Details and Improvements 56
		→ K. Calculation of Zone Events 58
6. Collision Response 27		
7. Results 28	→ N. Pictures of System Operation 63	
Result graphs		
8. Four Dimensional Analysis 32		
The Hextree, Hextree Use	→ L. Hextrees 59	
9. Discussion, Summary 35		
Future Work, The Dynamics Engine, Other Levels of Detail, Conclusion, Acknowledgments	→ M. Sparse Dynamics Code 60	

1.0 Thesis Overview

This thesis describes a new method for rapid physical simulation called *sparse dynamics*. The format of the document is somewhat unusual. See the following page for details.

Section 2 describes the background of the dynamics problem and the shortcomings of conventional methods in attempting to address it. Section 3 introduces our new dynamics model and discusses its application to various dynamic effects. Section 4 presents the discrete simulation system used to schedule events for the sparse phase of operation. Section 5 examines an algorithm for detailed collision analysis and its integration with the sparse phase. Section 6 briefly discusses collision response. Section 7 provides timings and other empirical results of the system's operation. Section 8 introduces a hierarchical data structure useful for tracking object proximities and improving efficiency. Section 9 discusses future directions for the work and concludes with several applications.

Each page or section may refer to appendices for additional detail. The appendices contain specific or technical information not required in the basic flow of discussion, but useful for the in-depth reader. Implementation details and numerical calculations will be found there.



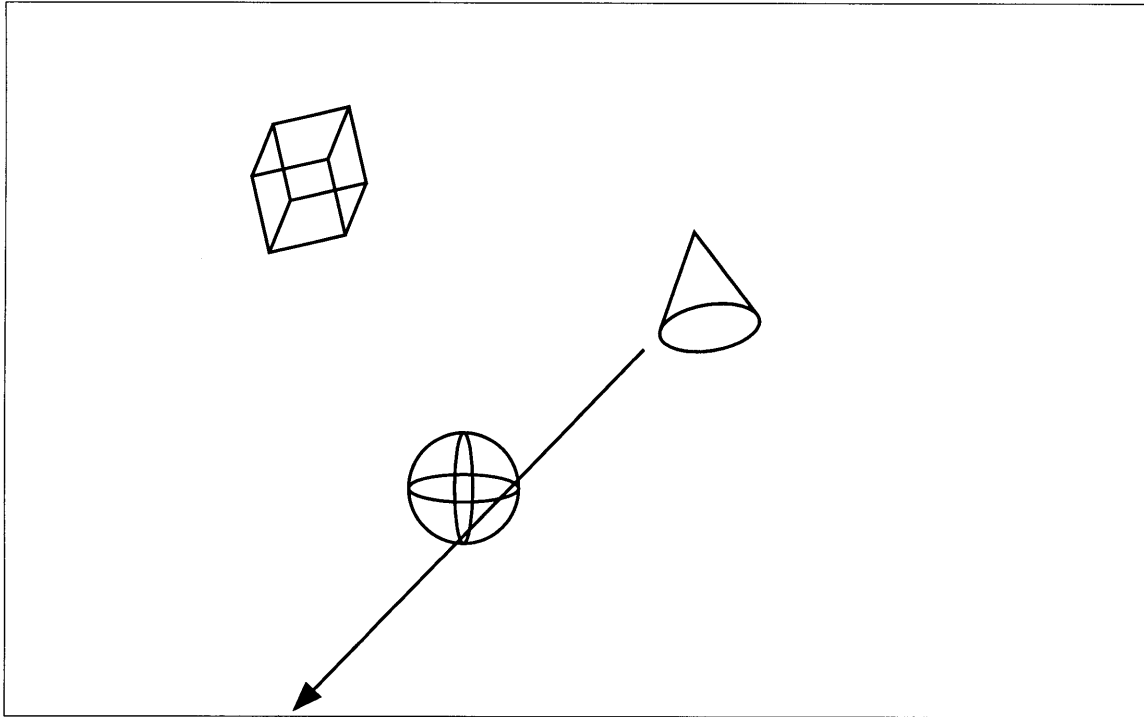
1.1 Thesis Format

The format of this thesis has been motivated by the desire to make a rather abstruse topic comprehensible. Each page in the main body contains a large illustration above a block of explanatory text. The intention is for the reader to come away with one clear notion per page, as a framework into which to integrate subsequent information. These pages are linked together to form the “story” of the work. Many of the main pages refer to appendices in which technical details are discussed. In addition, short descriptions are commonly placed above the illustrations to point out the main issues to be covered.

The explanatory text may refer directly to parts of the illustration by means of *reference arrows*. An arrowhead (\blacktriangleright) appearing above a word or phrase can be followed directly upward to a matching arrow just below the illustration. From there, a dotted line runs to a third arrow which indicates the intended object. A hollow arrow indicates a region while a solid arrow indicates a specific object. The text may thus refer to elements in a picture without interrupting the flow of the discussion. If the reader finds this reference method distracting, it may be safely ignored.

For readers wishing a quick overview of the work, study of the pictures may be sufficient. For a thorough, nontechnical understanding, the accompanying text may be read as well. Those concerned with in-depth coverage may refer to the appendices as needed.

Reality does not look like this



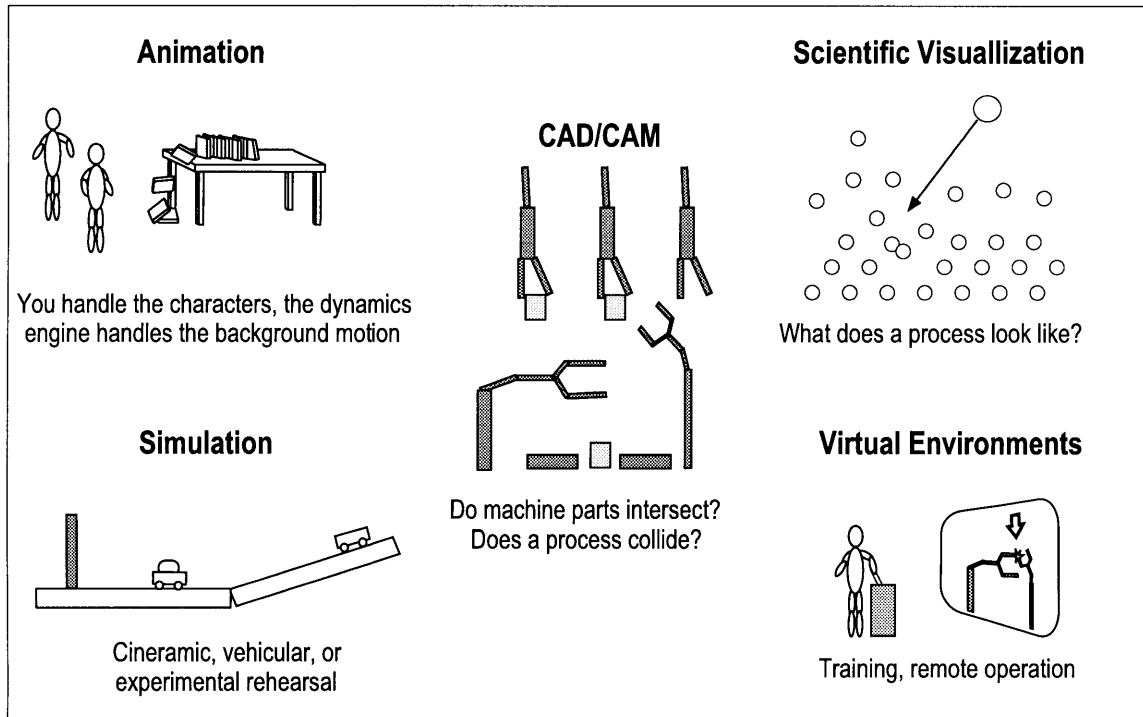
2.0 Introduction

Simulated environments have been an area of increasing interest over the past few years. The basic motivation is that some information is more easily understood in surroundings which approximate our real world experience. Although the subject has received intense scrutiny, the picture above remains an unpleasantly apt characterization of the current “state of the art.” Our lack of real-time simulation techniques is particularly hampering as interest grows in areas such as scientific visualization.

The reality we are trying to model is governed by various physical laws. To provide an acceptable system, we must emulate these laws with sufficient accuracy. The complexity of the equations involved has given rise to a general sense that the problem can not be done “right” in real-time. Accuracy and efficiency are commonly held to be mutually exclusive.

In fact, while some aspects of the problem will remain intractable, we can do much better than we are presently. We intend to demonstrate methods which can be applied to make the problem more tractable without hindering the pursuit of accuracy. Application of analytic techniques common in other fields can likewise benefit this one. Specifically, we exploit economies of spatial and temporal coherence in much the same way as has been done with rendering in previous years. We hope to thereby derive speed improvements which are equally as dramatic.

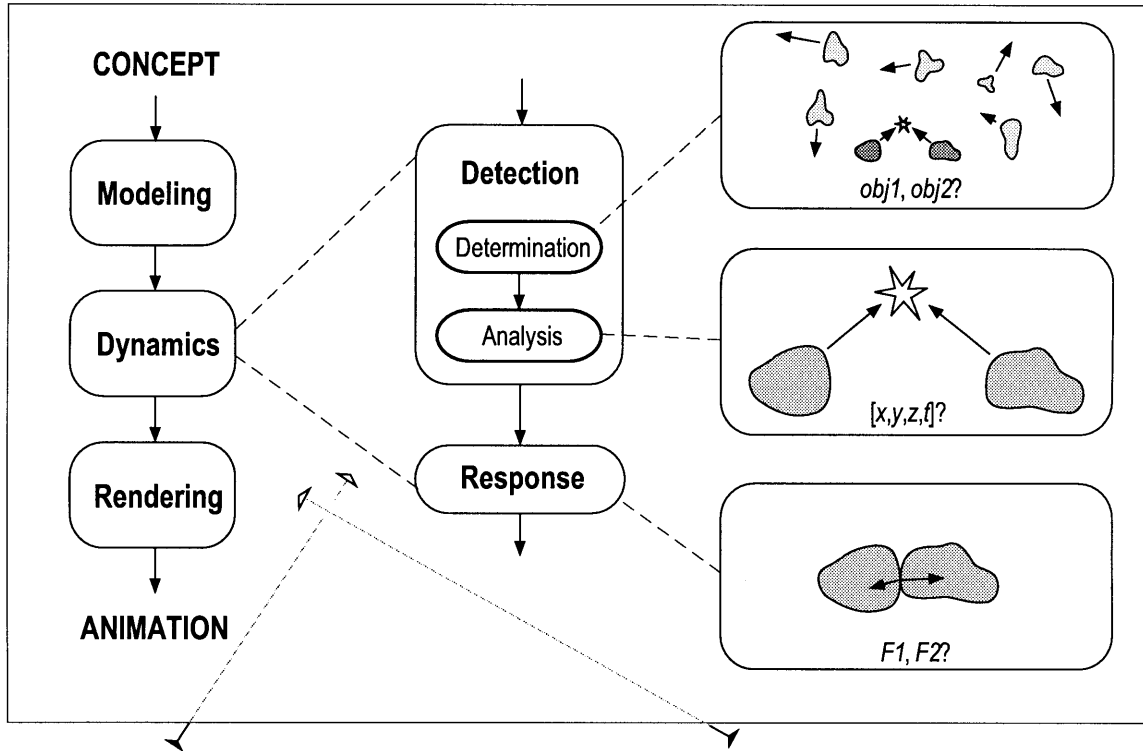
Examples of applications which would benefit from real-time dynamics



2.1 Applications

Many types of applications would benefit from the availability of real-time dynamics.

- An animator could use the dynamics engine to handle the motions of background objects while concentrating on character movements.
- Many types of scientific visualization tasks can be depicted with Newtonian dynamics, particularly molecular simulations.
- In both the construction and operation of machinery, there is often a need to determine whether parts will intersect or whether there are clear paths of operation. Real-time dynamics makes these determinations available to the designer at the time the assembly is configured. Packages currently available can handle such interactions for only a few moving parts at a time.
- In situations where a setup or experiment is expensive to construct, it is useful to first prototype the system via simulation to see that it operates correctly. Refinements can then be made at the (less expensive) virtual stage.
- Lastly, much of the work on “virtual environments” has been retarded due to the lack of realistic physics models.



2.2 Problem Context

The field of computer graphics can be separated into three areas: modeling, dynamics, and rendering. For a simulated environment, the modeling may usually be handled off-line, but the dynamics and rendering must be accomplished in real-time. Of the two, dynamics lags far behind in terms of capability. Current hardware can render hundreds of objects within a single frame time, but can compute the dynamics for only a few.

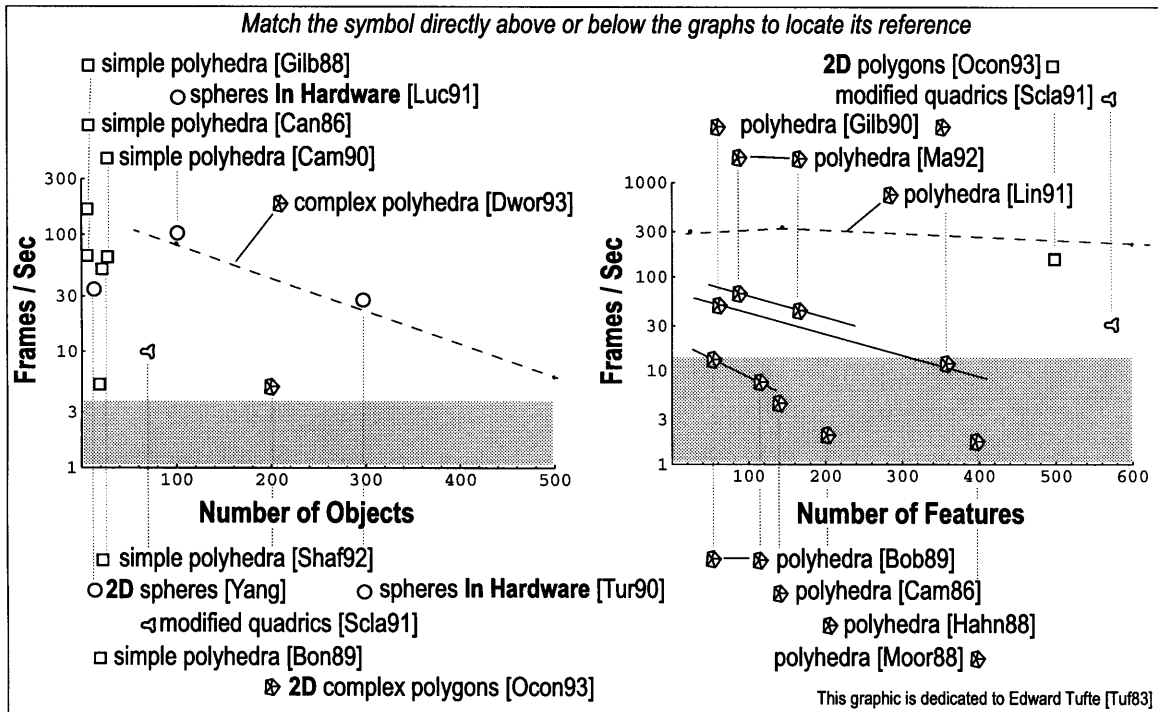
To model dynamic behavior, we must simulate the equations of motion for various objects. For the most part this is an evaluation and integration task. However, discontinuities appear at the points where objects collide. These situations are difficult to detect and so exact a high price if we wish to handle them.

Collision handling is the business of finding the where and when of these situations. [Moor88] divides the field into *detection* -- locating the where and when of a collision, and *response* -- determining its results. In the area of detection, we may make a further distinction between *determination*-- the business of deciding which of the objects in an environment come into contact, and *analysis*-- finding the time and location of that contact.

Empirical tests suggest that the detection phase incurs most of the cost of collision handling. This also makes intuitive sense-- response is only employed in the relatively uncommon event of a collision, while detection must operate among many objects at all times. It is therefore this area to which we will devote our attention.

Reported speeds of various
N-body collision algorithms

Reported speeds of various
two-body collision algorithms



2.3 Current State of the Art

There are currently a variety of methods for detecting object collisions. It is difficult, however, to gauge their performance potential. Most of the work that has been done has concentrated on accuracy rather than performance. As speed is not a priority, execution times commonly go unreported.

The above scatter plots contain samplings of what numbers are available. In each graph we have grayed the area representing sub-real-time performance. These numbers are gleaned from various sources under widely different conditions. As such, they can not be considered a comparative study. They are intended rather to give a general view of the state of the art.

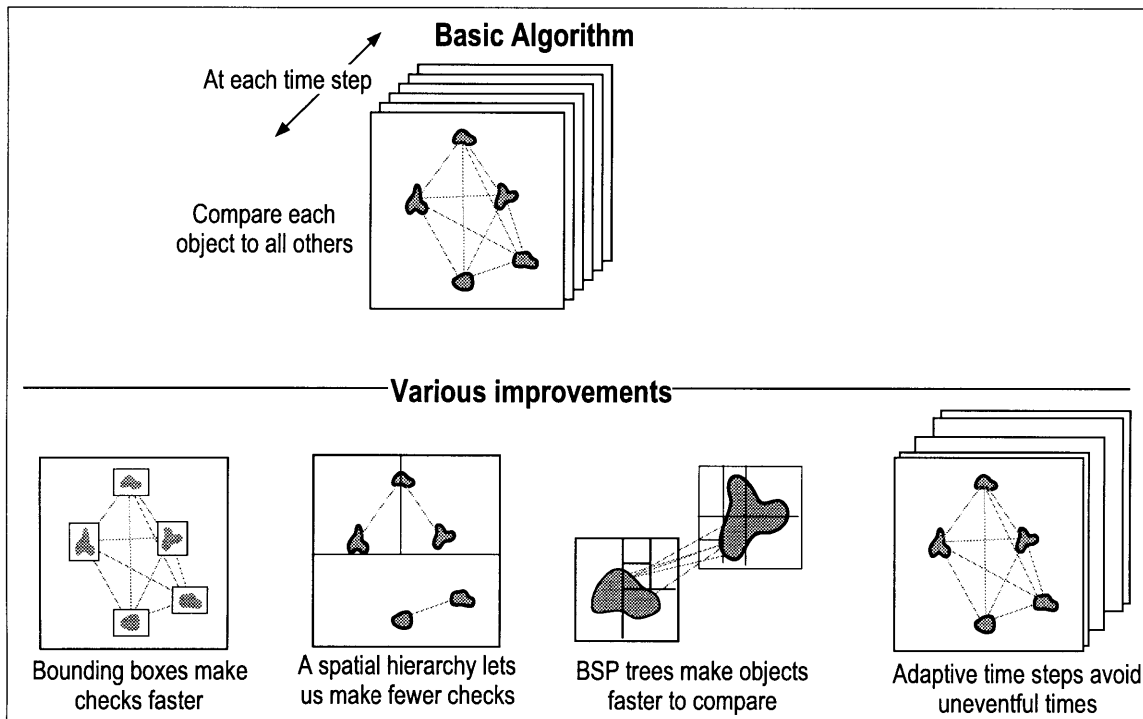
The left graph plots timings for algorithms which simulate many bodies. As we can see, little has been done for complex objects, and what has been done is in the range of tens of objects. The right graph plots algorithms which handle a single pair of objects. Here we see a great deal of work with complex objects, but in addition we see the reason it has not been applied to multi-body simulations. The speed of most of those methods prohibit their real-time use.

We have grouped the work into general classes, indicated by the symbols and captions. "Simple polyhedra" refers to algorithms tested on cubes and the like. "Complex polyhedra" indicates numbers of faces in the tens or hundreds. The [Moor88] data point is derived from a local implementation.

The work described in this thesis is represented by the dotted lines. We can see that it remains in the real-time realm well into the hundreds of objects, representing something of an improvement.

See the appendix for a detailed review of related work.

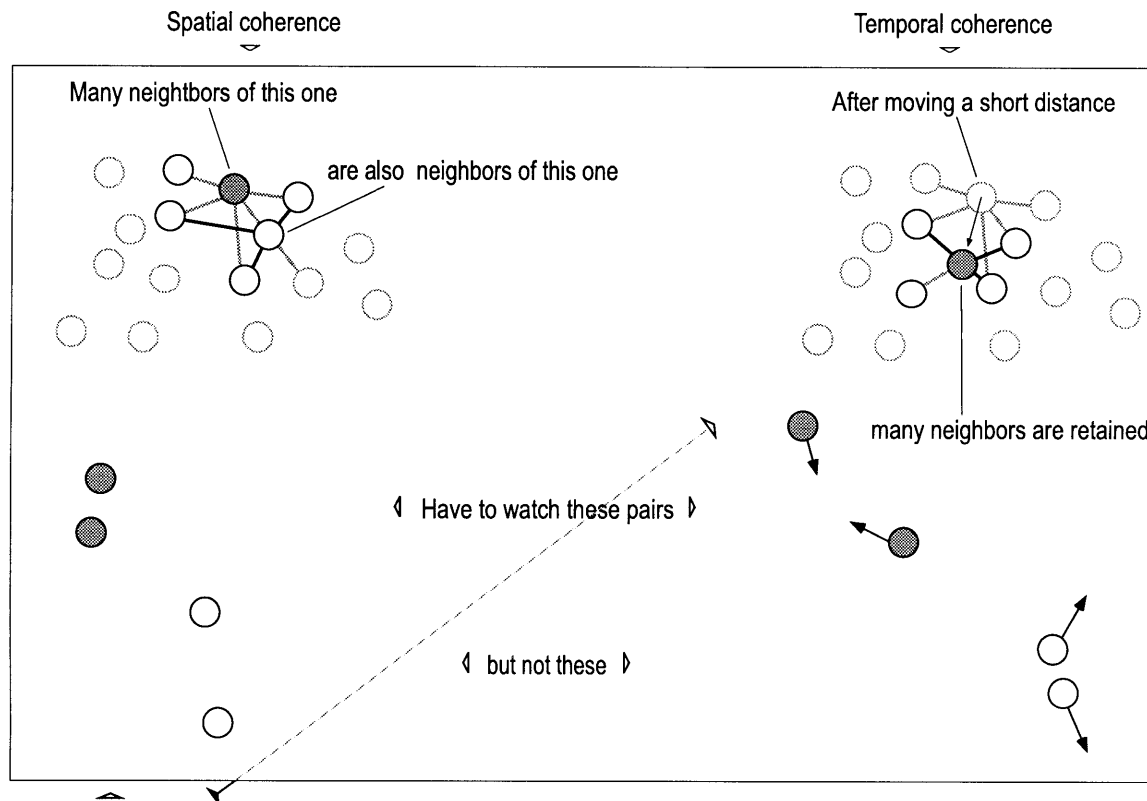
> See Appendix C (p.42)



2.4 Conventional Methods

Most current methods are variations of a basic algorithm: time is divided into small regular intervals. At each step, we update the position of each object and then check all others for possible conflicts. A number of improvements can be made to this basic method. Common examples are described in [Moor88]. Bounding boxes may be used to cheapen the cost of object comparisons, but do not alter the basic N^2 nature of the problem. A hierarchical space representation can be used to reduce this to $O(N \lg N)$. However, the motion of objects requires that the tree be updated or rebuilt at each clock step. This maintenance cost decreases the speed advantage gained. Hierarchical representations, e.g. BSP trees, can also be used to compare features of one object to another. Adaptive time steps may be used as well, to concentrate on times of greater activity.

Despite these improvements, the remaining expenses are still prohibitive. Much time is spent updating object locations, even though they are rarely examined. Also the cost of the inter-object comparisons is unacceptably high even as rarely as it does occur. Hubbard identifies three specific problems with the conventional method [Hub93]: the need for small time steps, the need to match each object against all others, and the expense of making inter-object comparisons. The necessity of addressing all three of these concerns simultaneously requires compromises in the efficacy or abilities of each. There can be no question that these topics have been individually addressed by previous research. However, as we see from the data on the previous page, none of these singular optimizations has been sufficient.



2.5 Our Approach

Our basic intuition is that objects should not have to look for collisions if they are not near anything. More precisely, we wish to divide our attention unequally, giving more to those objects most likely to interact. The fact that objects obey physical laws means their motions are predictable. This results in a game of prediction and estimation. If our prediction of the situation can be made accurate, we will be rewarded with reduced running time. Note that the intention is not to reduce the accuracy of our solutions, but rather the amount of time required to arrive at them.

We take advantage of spatial and temporal continuity to make our predictions.

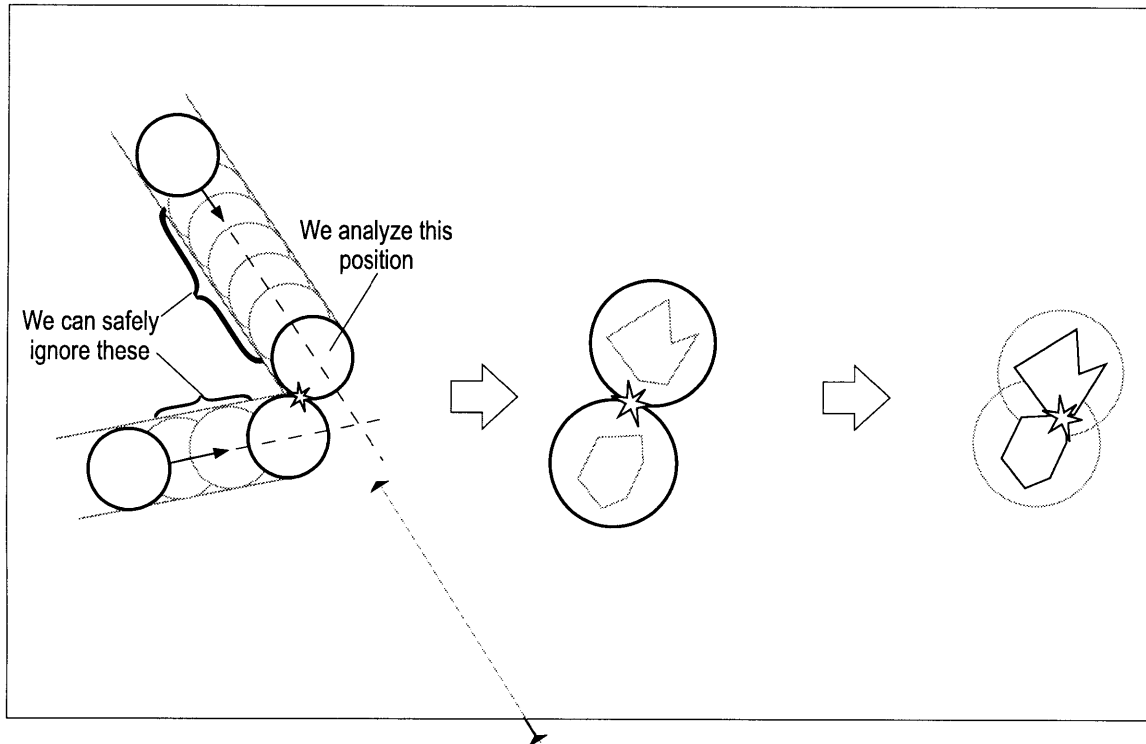
Space is coherent: if two objects are neighbors, they also likely share neighbors in common. We can exploit this adjacency with hierarchical representations to decrease the number of comparisons which must be made. Likewise, the surfaces of objects are coherent: points near each other on an object also share neighbor relations. We can exploit this to speed up inter-object comparisons using graph traversal algorithms.

Time is coherent as well: things move continuously from place to place. Thus if an object had a certain relationship to the world a short while ago, that relationship retains some validity. We can exploit this to avoid small time steps by constructing predictive models of motion and discrete event queues.

It is essential that our choice of methods be able to accommodate the operation of all of these coherencies at once. The next section describes the model chosen for this.

The sparse representation is used for initial comparisons

The detailed representation is used for close interactions



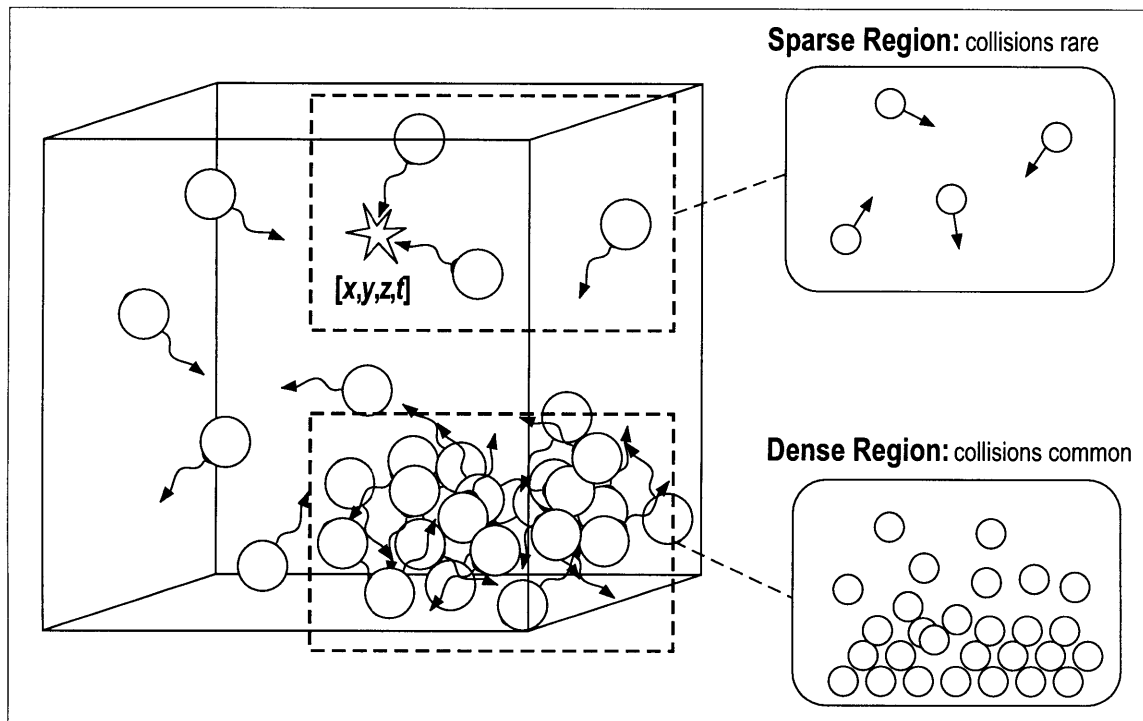
3.0 The Sparse Dynamics Model

We wish to construct a representation to optimize the identification of pairs likely to collide. For each object we construct a simplified representation, including a model of the object's current motion. We then use this model to predict the earliest time at which something could happen to alter its parameters. Assuming our predictions are conservative, we may safely ignore the object until that time.

The predictions are handled using discrete event simulation. Instead of stepping by regular time intervals, the clock moves forward from one event to the next. If the object's motion model is accurate enough, we need not even update object positions between events as they may be derived analytically for any specified time.

To predict the next event for an object, we compare its projected motion against that of others and locate the first possible time of conflict. At that time, either our prediction will have been correct, or the object must again search for its next potential encounter. Each event may thus result in additional future events. If our predictions are accurate, our attentions will remain focused on those objects in close proximity.

When we find a situation which exceeds the accuracy of the sparse model, it is handed to the detailed phase for analysis. The detailed phase uses a graph algorithm to track the smallest distance between the two objects for as long as they remain in proximity. If this distance ever becomes zero, we have detected a collision. At that point, the collision response is computed and then each involved object selects a new future target.



3.1 The Sparse Environment

Our new model is intended for use when objects are widely separated and collisions unlikely. We term this kind of interaction *sparse*. The simplified representation of each object is called a *sparse element* and a situation in which efficiency benefits from such analysis is termed a *sparse environment*.

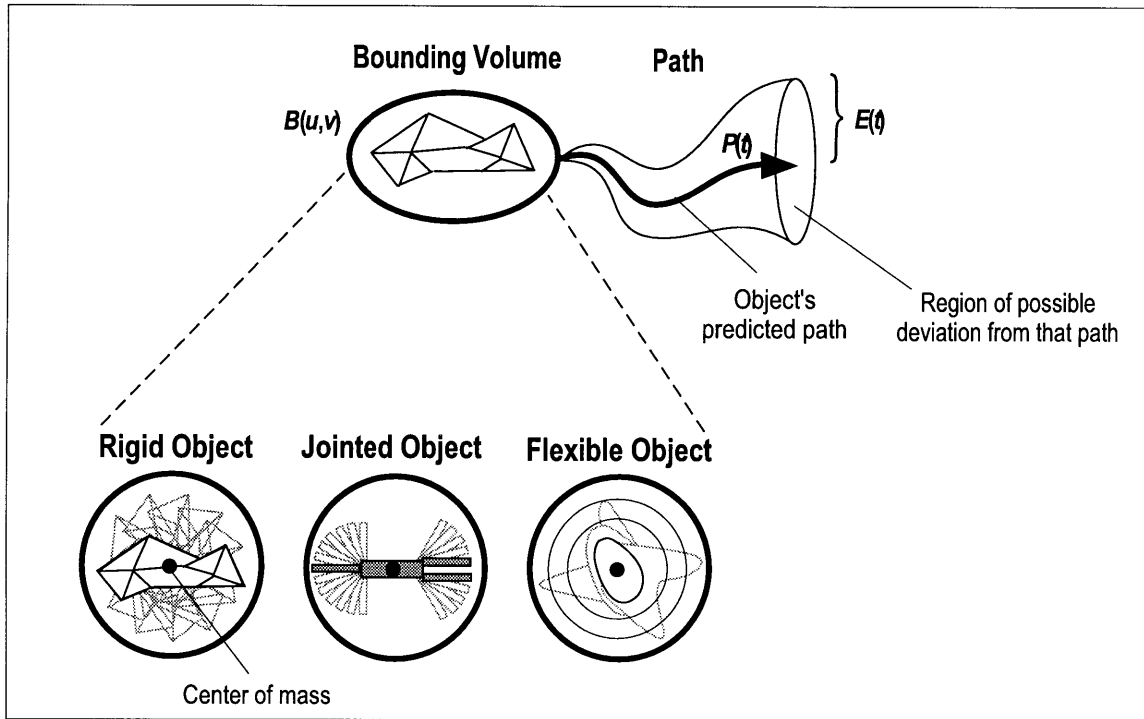
If objects are packed too tightly together, our predictions will become increasingly inaccurate and the efficiency advantage will be lost. We term these environments *dense*. For these regions and situations, other methods must be used. Since these areas are highly populated we may not care as much about individual locations. If so, we can use aggregate methods to analyze them. Approximate dynamics models may also be more acceptable in such situations. Section 9.4 considers this.

We characterize a region of space as sparse or dense in a qualitative manner. In a sparse environment, collisions are rare and the dominant effects are momentum and gravity. When collisions do occur, they tend to involve high energy transfer, meaning that the objects do not remain in contact for long. By contrast a dense case involves frequent collisions and the contacts are often extended. The dominant effects are then friction and support. Objects can therefore slide against each other in complex fashions.

For those regions where the sparseness condition holds, collisions can be identified with extreme rapidity. We believe that these criteria obtain in many common situations. For example, when constructing a virtual environment, space must be mostly empty in order to allow the user to see. The same holds true of narrative animation.

> See Appendix D (p.44)

An object element in the sparse dynamics model



3.2 A Sparse Element

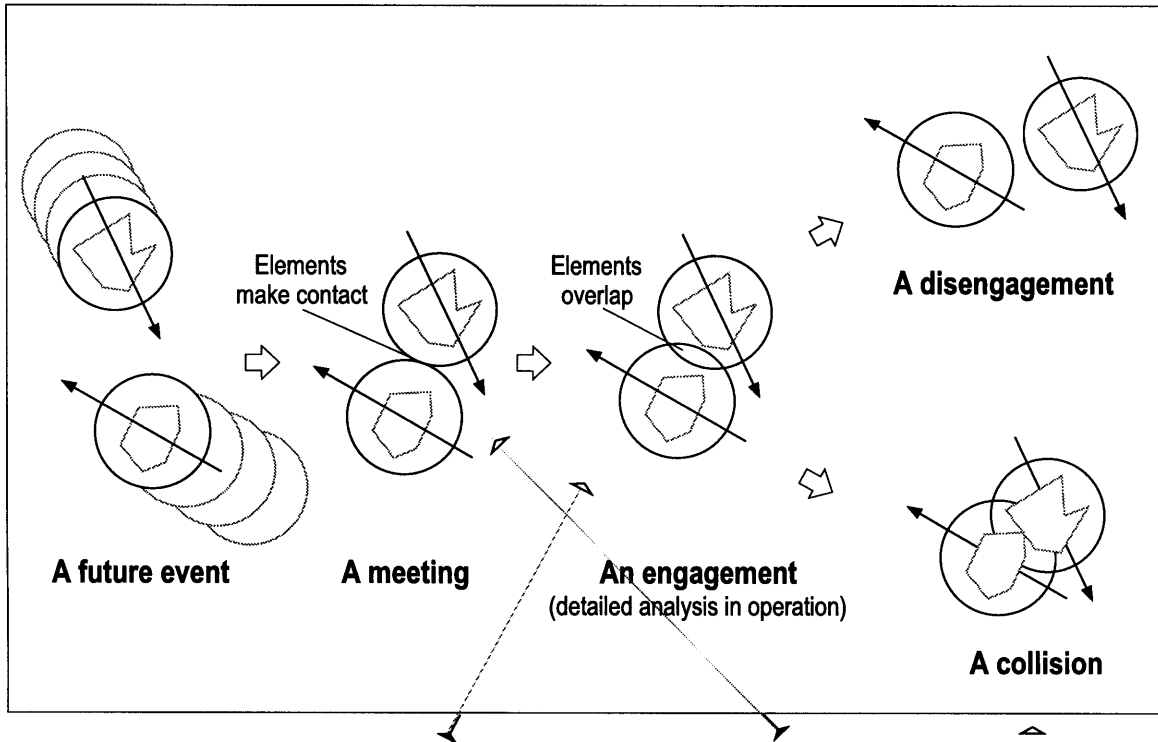
In the sparse model, each object is represented by an *element* consisting of a *bounding volume* and a *path* the element follows.

Bounding Volume. The bounding volume $B(u,v)$ is a shell wrapped around the object to simplify predictions. The shell should be conservative in that it encounters obstacles before the object itself does. Outside of B , the object is considered to have no effects. As such, the model can not represent field effects such as gravitational attraction which extend beyond an object's physical limits.

Bounding spheres are well suited for this purpose. They are cheap to compare, and if positioned at an object's center of mass, they can encompass all possible rotations the object can make. For kinematic linkages, this bounding sphere must be large enough to cover all configurations of the structure. For flexible objects, we must either place a limit on their malleability, or allow the sphere to change size over time. Within these criteria it is desirable to make the sphere as small as possible. The larger the sphere, the more territory that will have to be checked for its possible effects.

Path. The path function $P(t)$ is an analytic description of the object's motion over time. We prefer the function to be a low order polynomial to simplify predictions. As such, we maintain an error term representing the amount the actual motion might diverge from our approximation. Obviously, if the error term becomes too large, we lose predictive efficiency. This problem will be treated in Section 9.4.

The evolution of an element relationship

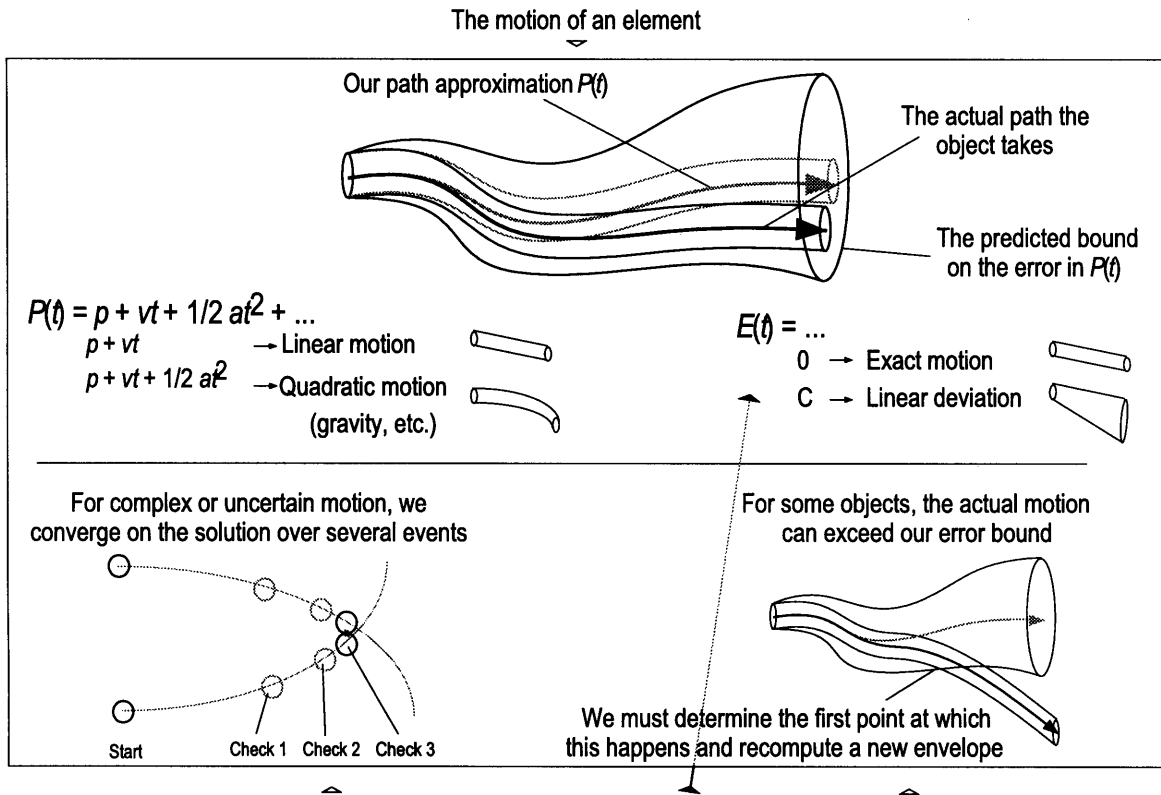


3.3 Sparse Interaction

When two elements first come into contact, they are said to *meet*, and for the duration of time they intersect, they are said to be *engaged*. This is to distinguish from the case of a *collision* which occurs between the actual objects that the elements contain. It is important to realize that when elements meet, it does not guarantee that the objects themselves will ever collide.

While an element is engaged, it still might have another sparse meeting with a third party. To watch for this, when an element becomes engaged it must perform a new global meeting check in the same manner as when it has a collision. A result of this is that an element may have more than one engagement at a time. That is, its bounding sphere may be passing through a number of others, none of which have yet made contact. If the object does undergo a collision, these other relationships must be maintained despite the object's new trajectory. Thus at a collision, we must cycle through an element's current list of engagements and update them.

The advantages of the sparse model are that exact values can be returned for collision locations and there is no accumulation error. Also, the rate of rendering can be selected independent of the simulation. The major disadvantage is the restricted nature of the model. If an environment is not sparsely populated, or if objects remain in contact for extended periods, the speed advantage can be lost. We address these concerns in subsequent sections.



3.4 The Path Equation

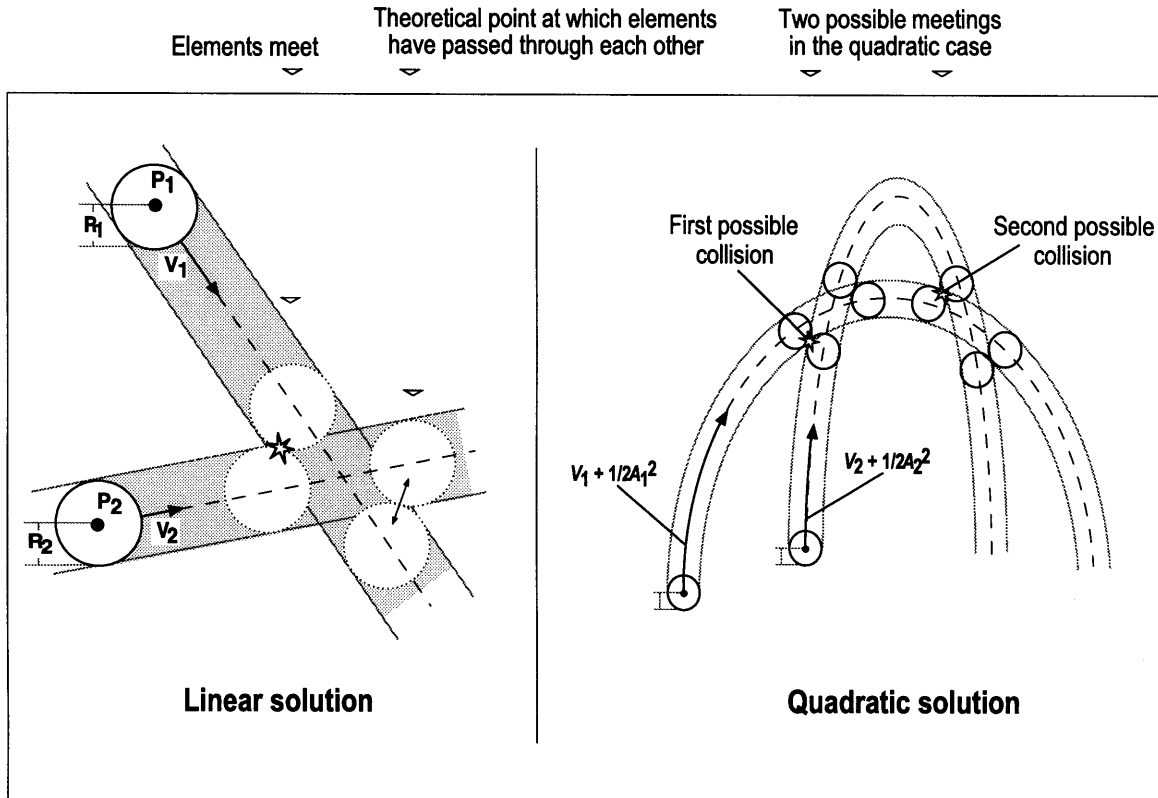
The efficiency of our method is based on being able to predict element meetings ahead of time. To do this we require an accurate formulation for the position of each element over time. The basic form of this function will be a polynomial in t with terms for the object position, velocity, and perhaps acceleration. As our motion model becomes more accurate, this equation will gain additional terms.

If equations are of low order (e.g. linear motion), intersections may be found analytically. If not, numerical methods must be used. The value resulting from this need not be exact as long as it is conservative. If a time is returned no later than the actual meeting, we can reliably converge on the true solution via future events. Clearly, the better our model and the more accurately we evaluate its roots, the less events we will require to home in on the contact. We therefore must balance the cost of various models and approximations with the precision to be gained from them.

In addition to the path equation, we provide an error term which is used to bound the object's actual motion about $P(t)$. The error term is meant to contain hard to characterize small effects, error left over from approximations in $P(t)$, and indeterminacy due to user input. The error term results in a gradual increase in the size of the element's bounding sphere. This represents our increasing uncertainty about the object's actual position in the future.

It is sometimes possible that an object can exceed the error bounds we set. In those cases, we must periodically check the object position for violations.

> See Appendix E (p.44)



3.5 Analytical Solutions

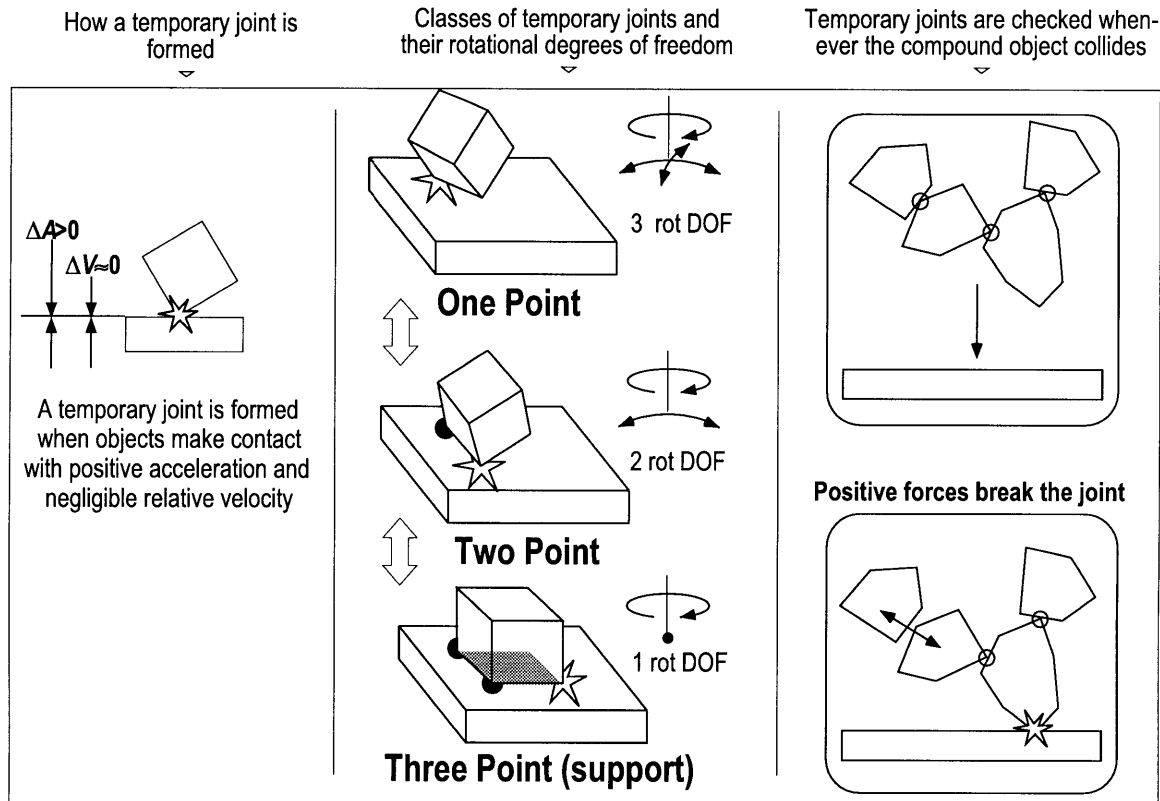
To find the meeting time of a pair of elements, we need the point in time at which their bounding volumes are just touching. If the object motions are of low order, we can compute this analytically.

For linear motion, the resulting equation is a quadratic. The solution has two roots, representing the time at which the objects meet and the time when they have passed through each other. If the roots are imaginary, then the elements do not become close enough to make contact. If the roots are real and the first of them is later than the current time, we have detected a potential future meeting. We compare the element in this way to each other element in the system. The earliest of all potential meetings found is selected as the element's next event. Note it is clearly inefficient for us to compare against all other elements. Methods to avoid this are discussed in Section 8.

A similar derivation can be made for objects undergoing acceleration. In this case, the path equations contain a second order term and so result in a 4th order equation. Methods exist for finding the exact roots of such equations. Any higher order terms, however, will necessitate numerical techniques.

The error term may be included in these calculations as well. As long as it is of no higher order than the path itself, it will not increase the order of the solution.

> See Appendix F (p.46)



3.6 Extended Contact

The basic sparse dynamics model assumes that objects move about independently, interacting only at instantaneous collisions. For extended object contact, which is common in the natural world, we must augment the model.

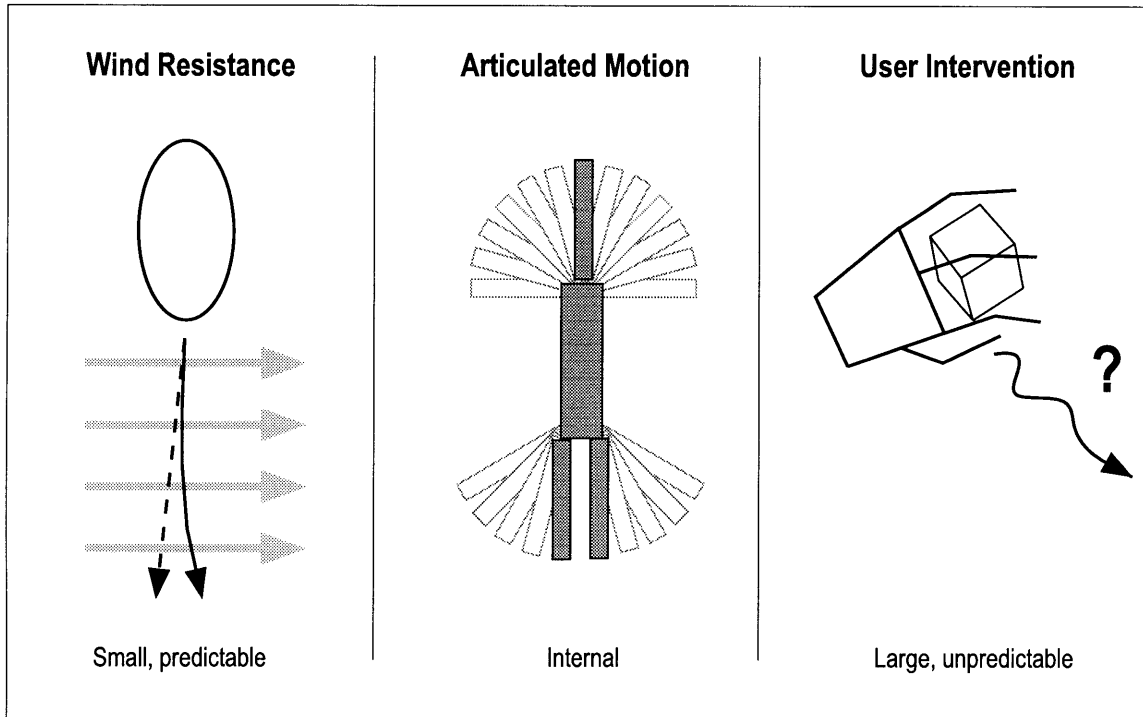
Extended contact occurs when objects collide with a positive relative acceleration and negligible relative velocity. We mark the point of impact as a *temporary joint*. The relative velocity component is then removed from the object motions so that they remain in contact. For the duration of the contact, the two objects are combined into a single articulated figure. A temporary bounding sphere is generated to encompass the new composite “object.” There are a number of methods for determining the internal motions of such jointed figures. See for example, [Sch91].

These compound objects are generated and altered on the fly. When such a compound object is involved in a collision, all of its temporary joints are checked to see if they receive a force sufficient to break them. Clearly this method does not scale well for large numbers of contact points. Additionally, loops in the kinematic structure are currently not handled well. Both of these conditions are indicative of a dense environment and should be handled as such.

Objects in continuous contact also bring up the issues of support and friction. See the appendix for a discussion of these.

> See Appendix G (p.48)

Examples of other possible forces we can include in our model



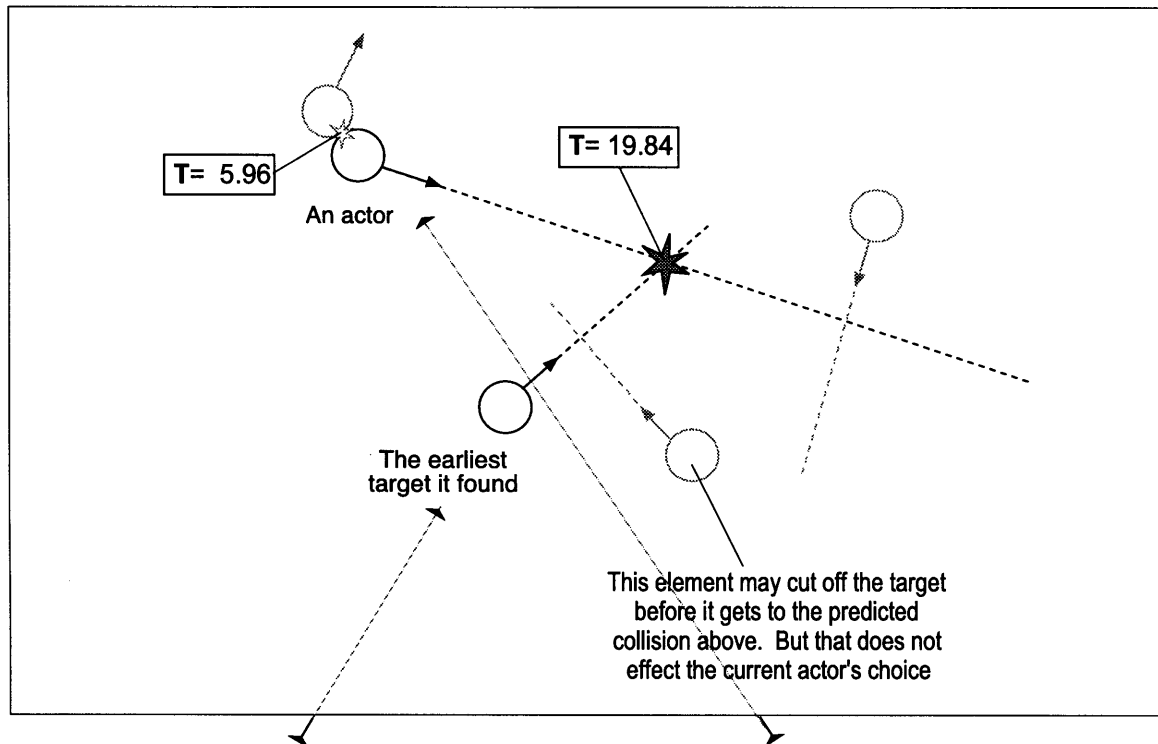
3.7 Other Forces

There are a number of other effects which commonly influence the motion of real world objects. These are such forces as wind resistance, hinges and mechanical linkages, and user interaction.

Our general procedure for including these will be:

- If the effect on the object motion is large, add a term for the effect to $P(t)$. If the effect is non-polynomial, construct a polynomial approximation and add a conservative error term to $E(t)$.
- If the effect is small, add the term only to the envelope $E(t)$.
- If the effect has an unpredictable component, place a bound on it and add that bound to $E(t)$. Calculate a conservative bound on how quickly $P(t)$ could escape $E(t)$ and queue checking events with that frequency.
- If the effect is internal to the object, determine what the aggregate behavior for the body will be and use that in $P(t)$. Implement a new module to determine internal configurations for render purposes and detailed phase interactions.

This element chooses this target for a meeting here



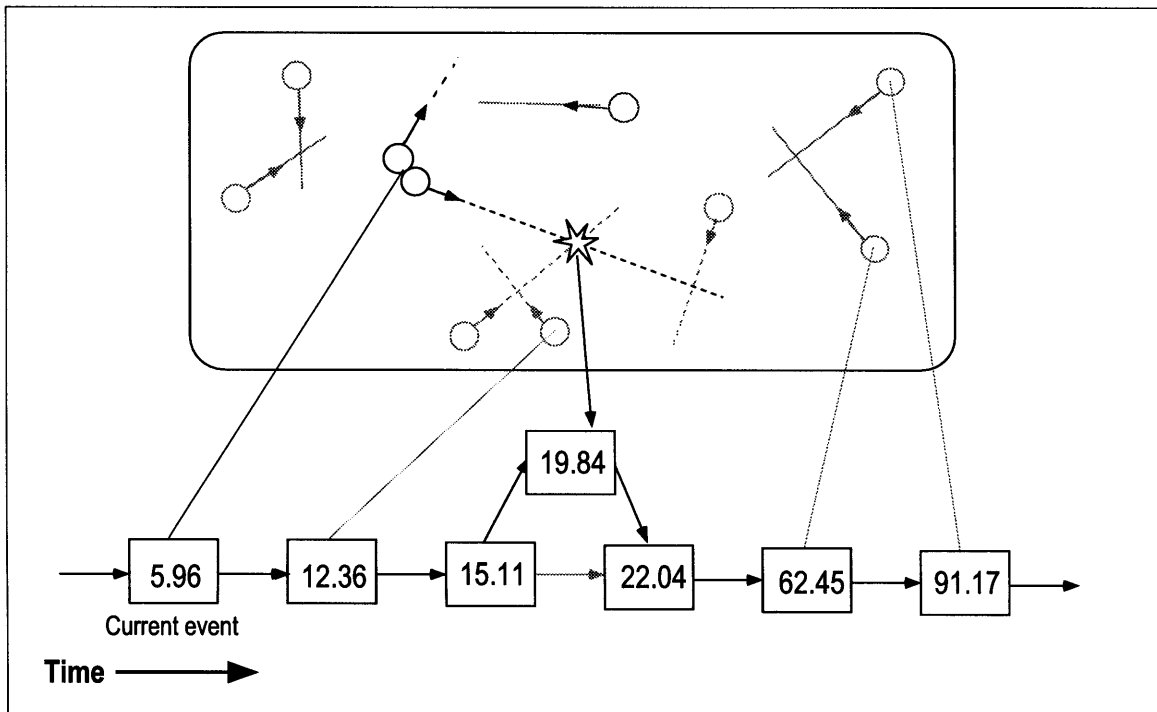
4.0 Sparse Event Handling

Given an environment populated with sparse elements, we need to determine which will meet, and when. We can use a discrete simulation queue to accomplish this.

Each element maintains a prediction of its next anticipated meeting. When a collision occurs, the object is diverted to a new trajectory. It must then make a new prediction of its next encounter. This can be found by testing the element against all others and choosing the earliest meeting found. When an element is looking for its next meeting, we term it an *actor*. The element identified by its search is termed its *target*. It is possible for an element to find *no* future event. In that case, the element will remain on its present course forever unless some other element is diverted into its path.

It is important to note an asymmetry in the prediction algorithm. When an actor determines its target, the target is not notified of that fact. It is not until the time of the actual meeting that the target is affected. It might be possible to make predictions more accurate if objects shared information. However their independence allows us to be more confident of missing no collisions.

Likewise, it is irrelevant to the actor whether its target will be diverted before their encounter. It must only determine that it need do nothing *sooner* than that time. Things may change in the meanwhile, but that will be the responsibility of those objects doing the changing.



4.1 The Event Queue

Each element's predicted meeting is stored in a time-ordered event queue which will therefore contain $O(N)$ entries. The queue may be implemented as a heap or other efficient data structure. When objects collide, they make new predictions which are inserted into the queue in time order. At each collision, any object whose motion has been altered will need to predict its next target. In a pairwise collision, this means two searches against the universe at a cost of $O(N)$.

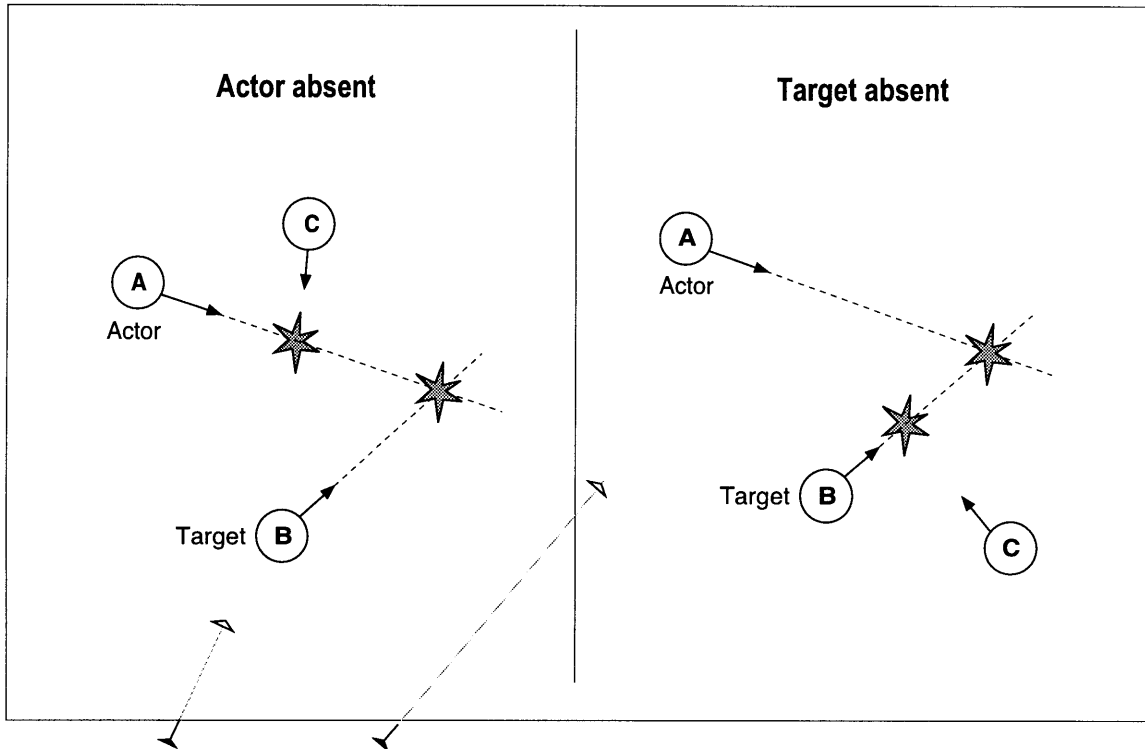
A similar queueing method is suggested in [Lin92]. However, their method keeps track of all pairs of objects at all times, thus requiring $O(N^2)$ events.

Outline of Algorithm. Before simulation begins, each element locates its first target and queues an event for it. This initialization step is somewhat slow as it requires N^2 nontrivial comparisons.

On each iteration of the system, we remove the earliest event from the queue and set the simulation clock to its time. The two elements and their point of contact are passed to the collision analyzer (see section 5). The analyzer may elect to queue additional events to more precisely track the collision point between the two objects. If a collision finally results, each element calls the meeting predictor with its new trajectory and queues the new event found by it.

Between events, the simulation is simply stepped forward at the rendering rate. At each step, the positions of the objects are updated according to their trajectories and the renderer is called.

> See Appendix I (p.52)



4.2 Stale Events

The price of our conservative prediction method is that predictions can be invalidated by subsequent events. An object will miss a predicted meeting if it is diverted from its path before the event arrives. We term such predictions *stale* and must avoid acting on them.

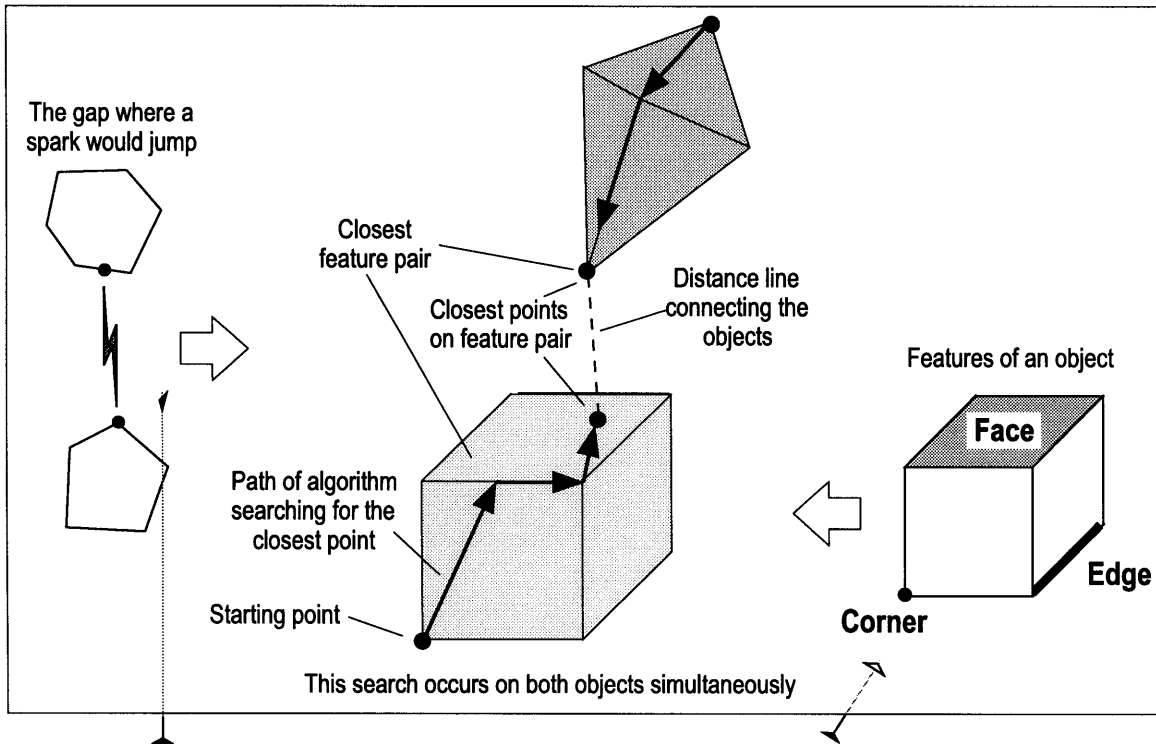
Fortunately stale events are easily detected. Before processing each event, we check the timestamps of the elements involved. If any of them have been affected more recently than the time at which the prediction was made, the prediction is now invalid. We distinguish two types of stale events: *actor-absent* and *target-absent*.

Actor-absent. In this case, an element makes a prediction and then is struck before the prediction occurs. When such an event comes up on the queue, it can be safely ignored. The target element will continue through the predicted meeting point unaffected and the actor itself will have already located a new target at the time it was struck.

Target-absent. In the other case, an actor makes a prediction but its target is diverted. When the event arrives, the actor will have nothing with which to collide. At this point, the actor will need a new target. It must therefore reinvoke the meeting predictor just as it would have if a collision had occurred.

If there are many stale events, efficiency will suffer. This is one reason our method is applied only to sparse environments. When events are rare, events also become invalidated more rarely. An increase in stale events is one of the measures we can use to detect dense situations. In the tests done for this thesis, the proportion of stale events ranged between 20% and 50%. See Appendix I.2 for a description of how this affects performance.

The algorithm moves along the surface of the objects toward the smallest gap separating them



5.0 Collision Detection

When the bounding shells of elements intersect, we must analyze the situation with more detailed methods. Sparse analysis can be used as a first phase for most any collision algorithm. Any object representation which can be placed in a bounding sphere can be accommodated. The collision detection methods employed can be relatively complex as most potential collisions have already been ruled out in the sparse phase.

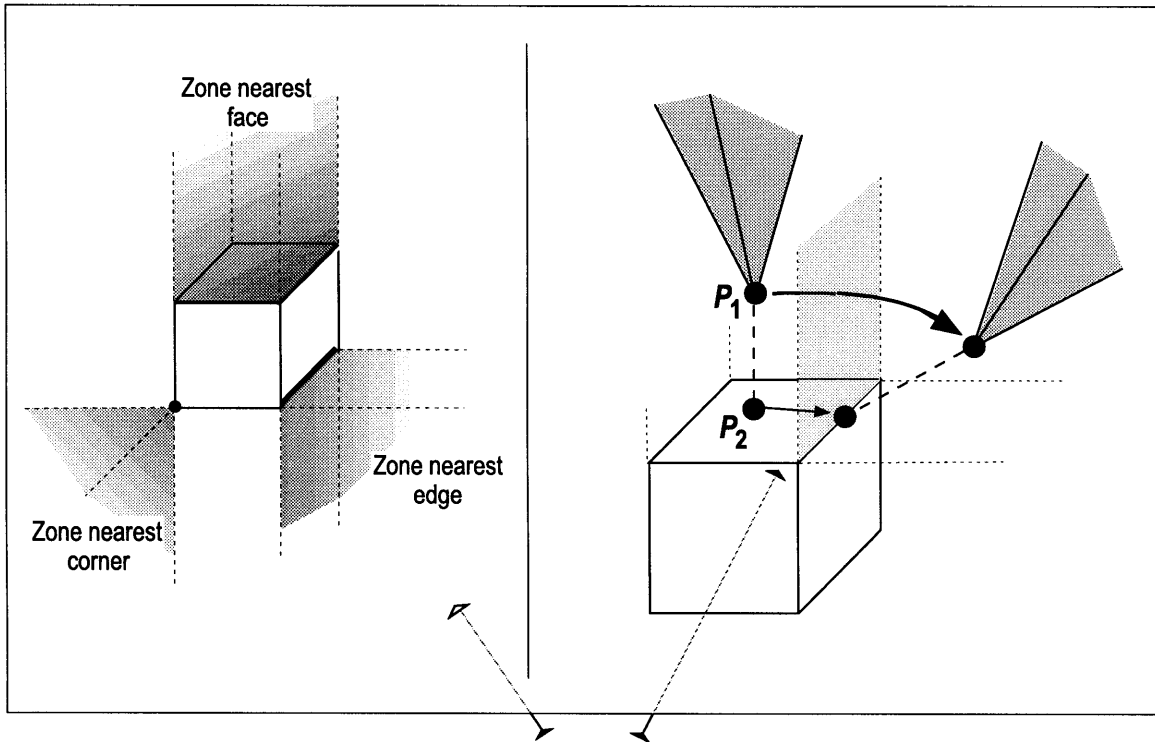
For this thesis, we have developed a polyhedral collision detector based on work described in [Lin91, Lin92, Lin93]. The method runs quickly and fits easily into the sparse model.

The Lin and Canny algorithm is based on the geometric properties of convex sets. For a given pair of convex polyhedra, there must exist a line of minimum distance connecting them. Intuitively, this is the gap where a spark would jump from one to the other. Note that in some situations this line can be multiply defined. In these cases, we are free to select any connecting line having the minimum length.

The algorithm divides each polyhedra up into its component features: faces, edges, and vertices. If we start with any two points on the surfaces of the objects, we can follow the gradient along the object's surfaces "downward" toward that gap.

Examples of the regions
closest to various features

As P_1 moves from the zone nearest the face to that nearest the edge, P_2 slides continuously along the surface of the cube



5.1 Lin and Canny Distance Calculation

The tracking of points on the surface of a polyhedron can be made extremely efficient through appropriate preprocessing. We assign a “nearness zone” to each object feature, that is, the zone of space outside the object which is closer to that feature than any other. Each zone will be delimited by planes.

If a point which was in a given zone has moved, we need only test it against the zone’s bounding planes to see if it has broken through any of them. If so, we know the point will then be in the adjacent zone, and thus closest to that zone’s feature. Each object keeps track of the point on itself closest to the other, using this method.

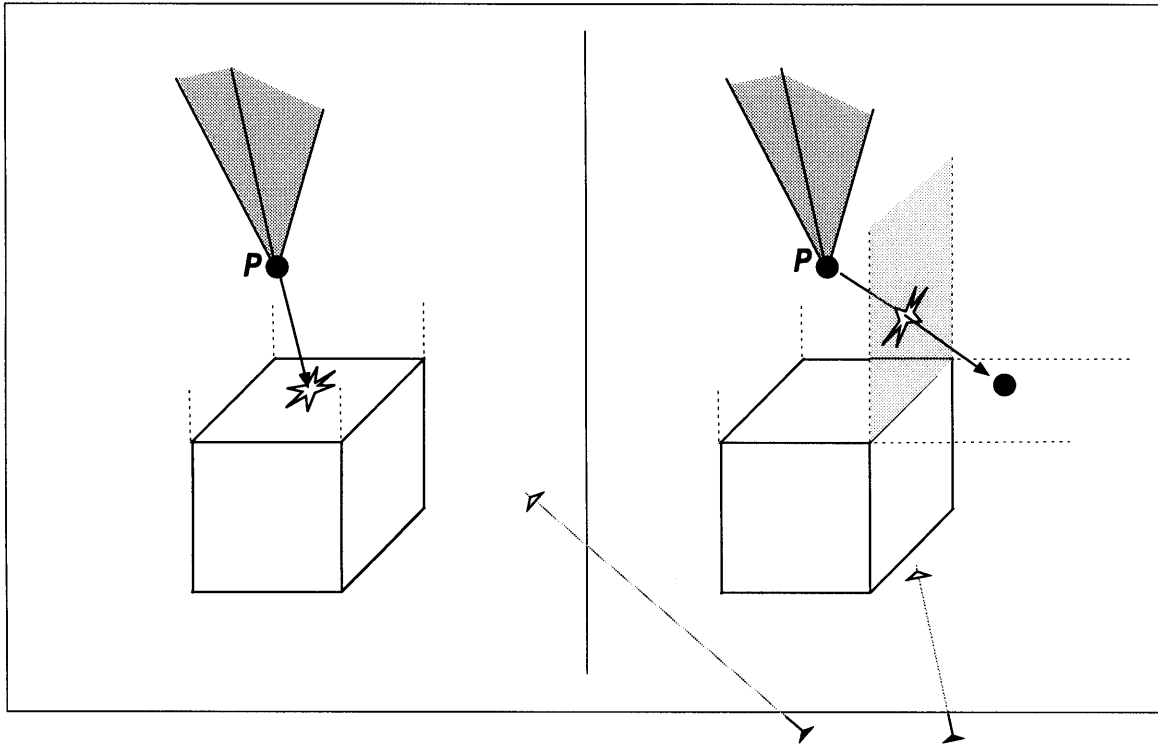
The algorithm begins by selecting an arbitrary feature on each object. It then looks to see if any bounding walls around either of the features is violated by the point on the other. If so, we move our anchor on that object to the neighboring feature on the other side of the violated wall. This process is repeated until no walls are violated.

Once the closest points have been established, they may be easily maintained over time. As the objects move, we begin our search from the previous optimal points. The new closest points will likely be nearby. Clearly if the length of the connecting line ever goes to zero, we have detected a collision.

See the papers cited on the previous page for a more complete discussion of the algorithm.

P will either hit the cube's face...

...or pass through one of the walls surrounding its zone and become closest to a new feature



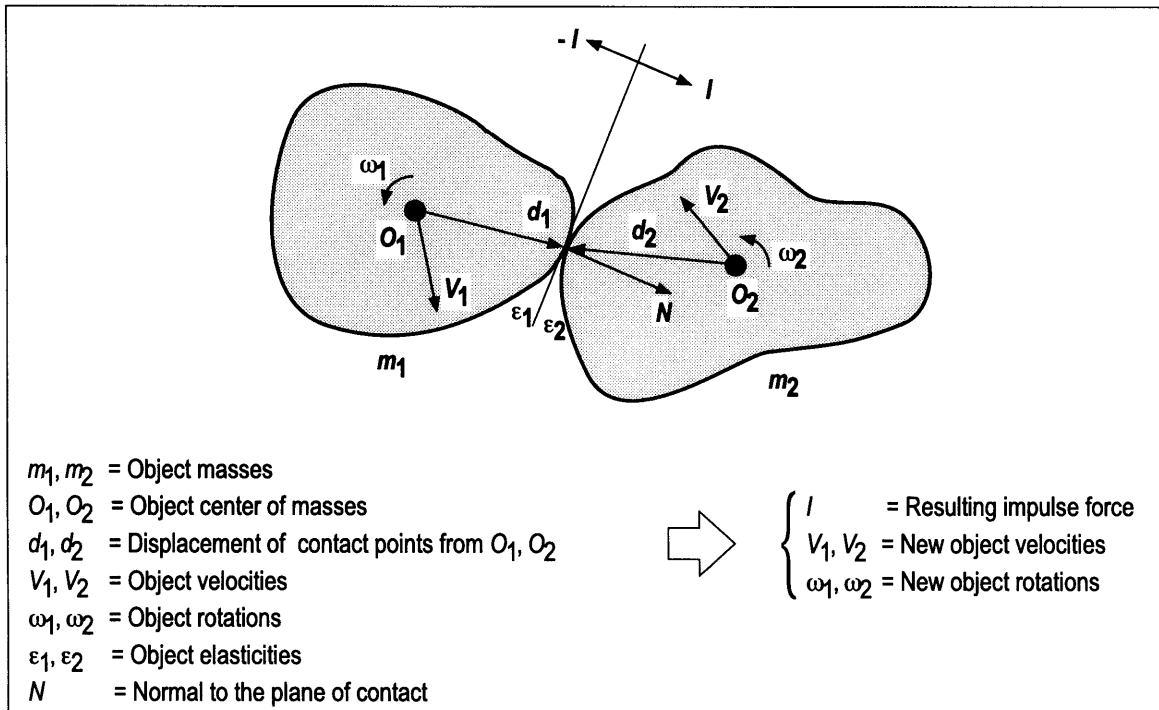
5.2 Interface to the Sparse Model

When elements meet, their closest features must be tracked for as long as they are engaged. We could do this by stepping at small time intervals and checking to see if they have collided or changed closest feature pairs. However, we would prefer a method analogous to that of the sparse phase-- one that allows us to identify key events in the object motions and disregard the rest.

The Lin and Canny algorithm provides this opportunity. Given objects with known closest features, one of two things must happen: the feature pair will collide, or a new pair will become closest. Given the deterministic nature of object motion set out in Section 3, we can predict this. The motion of the closest feature points on each object can be projected until they encounter either one of the zone walls or the feature itself. The soonest such encounter for either object is queued as the next event for the pair. Barring outside influence, no collision will occur between the objects until then. As before, if the object motions are complex, exact time predictions will not be possible. In those cases, we resort to the same convergence techniques discussed in Appendix E.

Note that when an object pair enters into the detailed phase, each must still search for and queue its next sparse encounter. An object can be engaged with multiple others at one time and these situations must be detected.

Model of object impact using impulsive forces



6.0 Collision Response

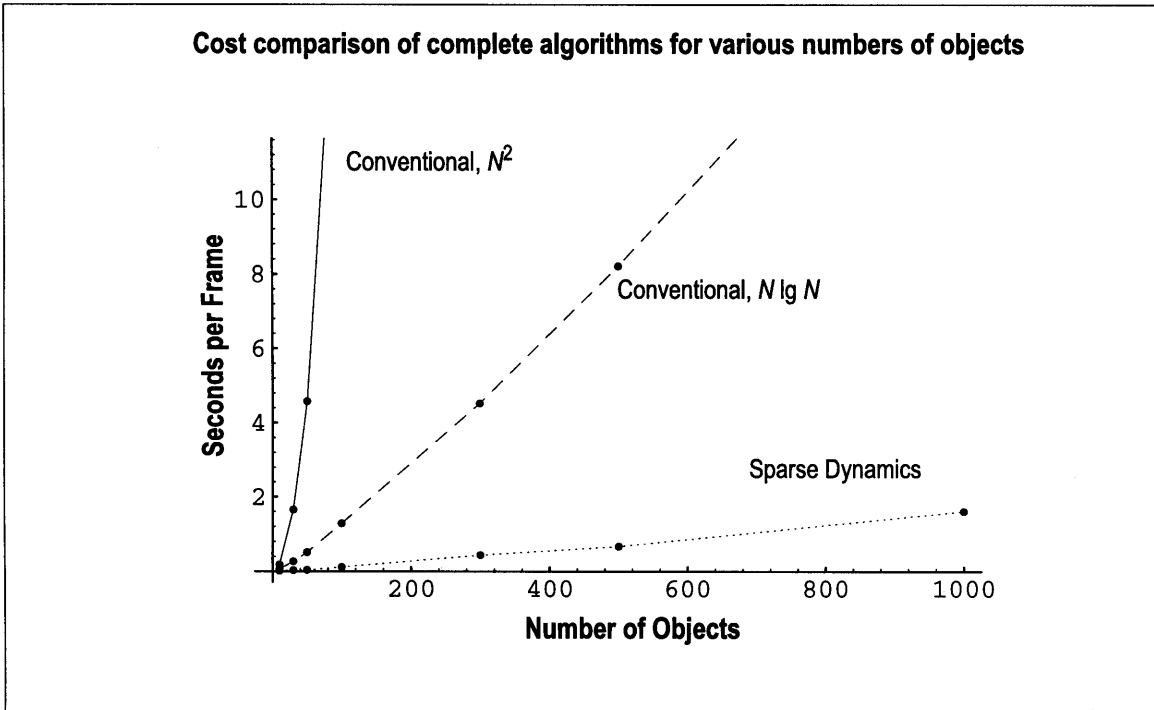
When the detection module determines that two objects have made contact, the situation is passed to the response routines. At this point, various methods may be used to resolve it.

Simulated springs are common. They are somewhat analogous to our conventional collision detection algorithm in that they require small time steps and accumulate integration error. They likewise are easy to program, but costly to simulate.

We are currently using the other common response technique -- impulse forces. This method computes instantaneous forces at the point of impact to push the objects apart. As these forces are not applied over time, they do not hinder a non-time-stepped simulation. Impulses provide realistic looking motion and some level of physical validity. See [Hahn88, Moor88, Gold60, Mac60] for more detailed discussion.

The major failing of impulse forces is that the model breaks down when contacts become extended. With current methods, we are spending about 10% of our computation time doing collision response. As more computer power becomes available, more complicated means could be employed. Aspects of analysis could include surface compression, finite element deformation, radiated shock waves, and inelastic effects.

See [Bat85, Will87, Will88, Will92, Bai93, Bra91, Baraff *et.al.*] for discussions of these.



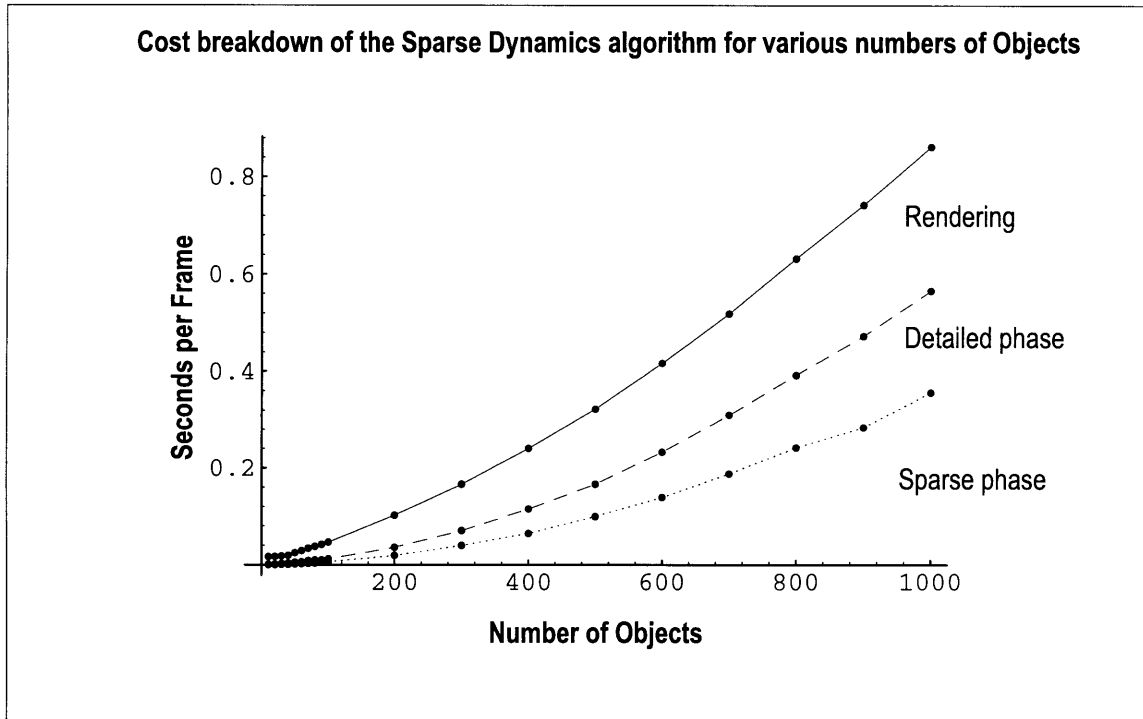
7.0 Results - Cost Comparisons

The various algorithms described have been coded in C on a Silicon Graphics Skywriter (33 MHz RS3000). See Appendix M for a description of the code. Timings were made for the conventional and sparse methods, for various numbers of convex polyhedral objects. The polyhedra had randomly chosen complexities from 24 to 600 faces. The graph reports the times per frame and includes rendering.

As the number of objects in the simulation increased, we increased the size of the environment commensurately. This provided a universe of constant density (5%) so as to make timings comparable. The timings indicate that the sparse model is able to maintain real-time rates for simulations of objects numbering in the hundreds.

The sparse dynamics times above are for accelerational motion with $E(t)=0$. See Appendix I.2 for a characterization of the effect that inclusion of an error term would have. If all objects have large error envelopes, the simulation will be drastically slowed. However, in the usual case, only a few objects, such as those under the control of the user will need this treatment.

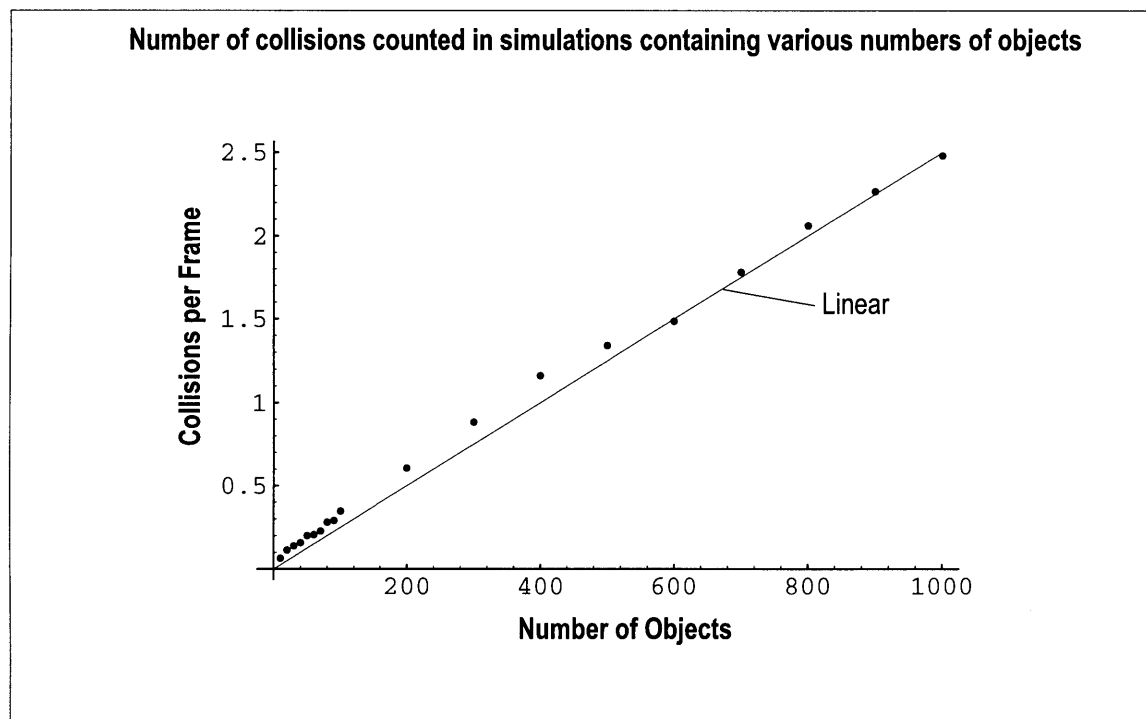
See the appendix for pictures of the system in operation.



7.1 Results - Cost Breakdown

This graph shows the proportion of time consumed by various parts of the sparse dynamics system. The dotted line represents sparse detection only. The dashed line includes the cost of the polyhedral algorithm. The solid line includes rendering. The top line of this graph, matches the lowest line on the previous page.

We see that rendering and polyhedral collision detection are roughly linear in the number of objects. As discussed in Appendix I.3, the sparse phase increases faster than this, but with an extremely low constant. At 1000 objects, it is still an acceptable fraction of the simulation time.

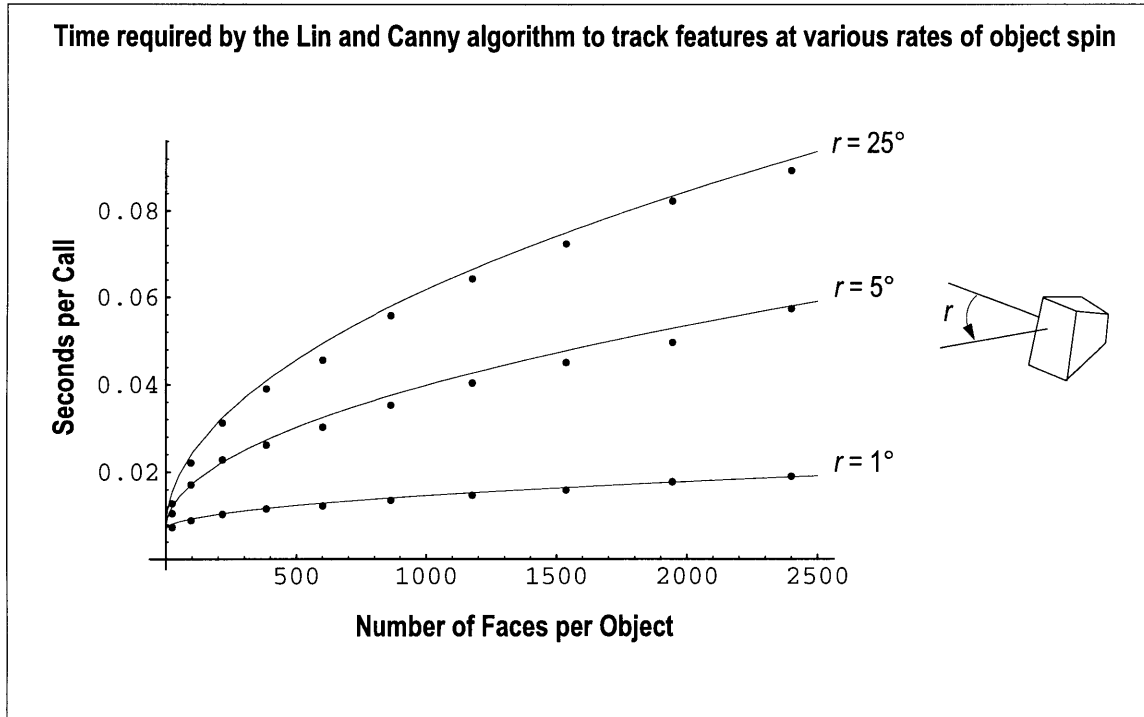


7.2 Results - Collision Counts

We also examine the number of collisions occurring in the previous simulations. This is to verify that the sparse system is working correctly and to gain a handle on the number of events as N increases.

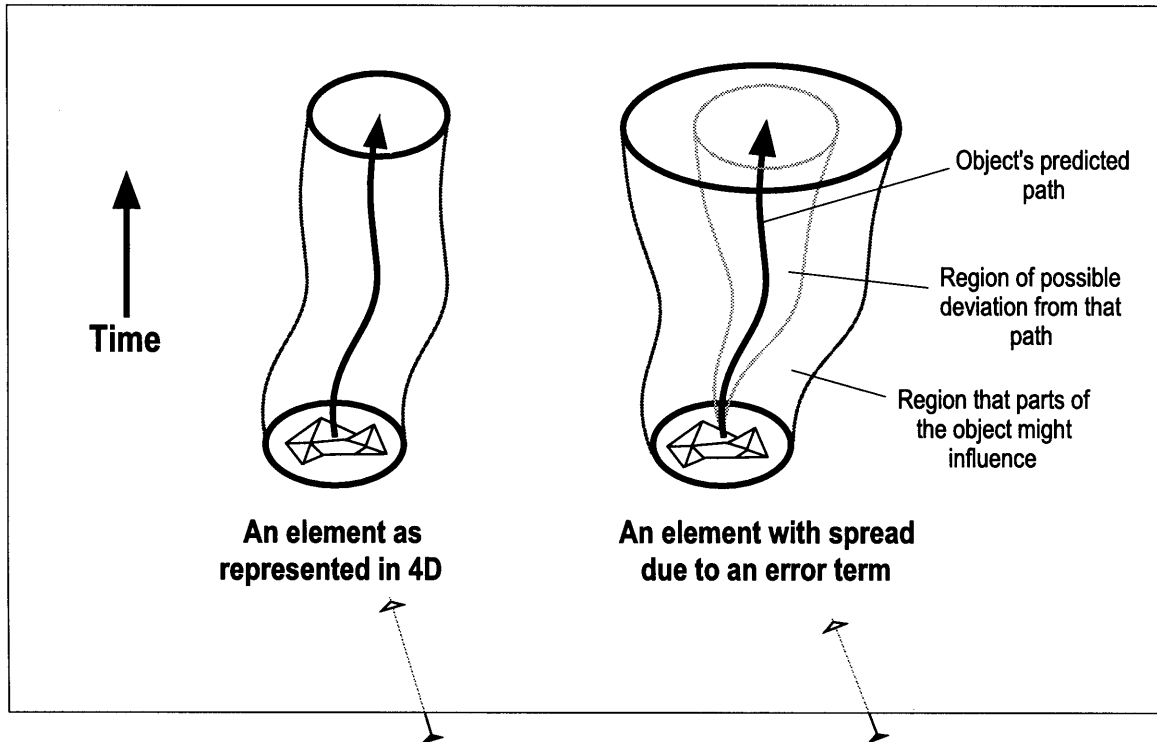
We plot the number of collisions occurring in the simulation as a function of the number of objects. A well known result from physical chemistry states that the number of collisions should be linearly proportional to the number of particles in the system [Cas83]. The graph shows agreement with this.

These numbers also closely match the collision counts from the conventional simulations. The values do not match exactly due to differences in the location of collisions. The conventional method contains small errors due to its time stepped nature. As in any chaotic system, small deviations can lead to a significantly different outcome.



7.3 Results - Feature Tracking Cost

This graph presents timings for the polyhedral collision detection algorithm discussed in Section 5. We plot the cost of the algorithm for rotating objects with various numbers of faces. The three sets of data points are for fixed rotation angles of 1°, 5°, and 25° per frame. The solid lines are each proportional to \sqrt{N} with differing scales. These results corroborate the order calculation for this algorithm made in Appendix J.2.



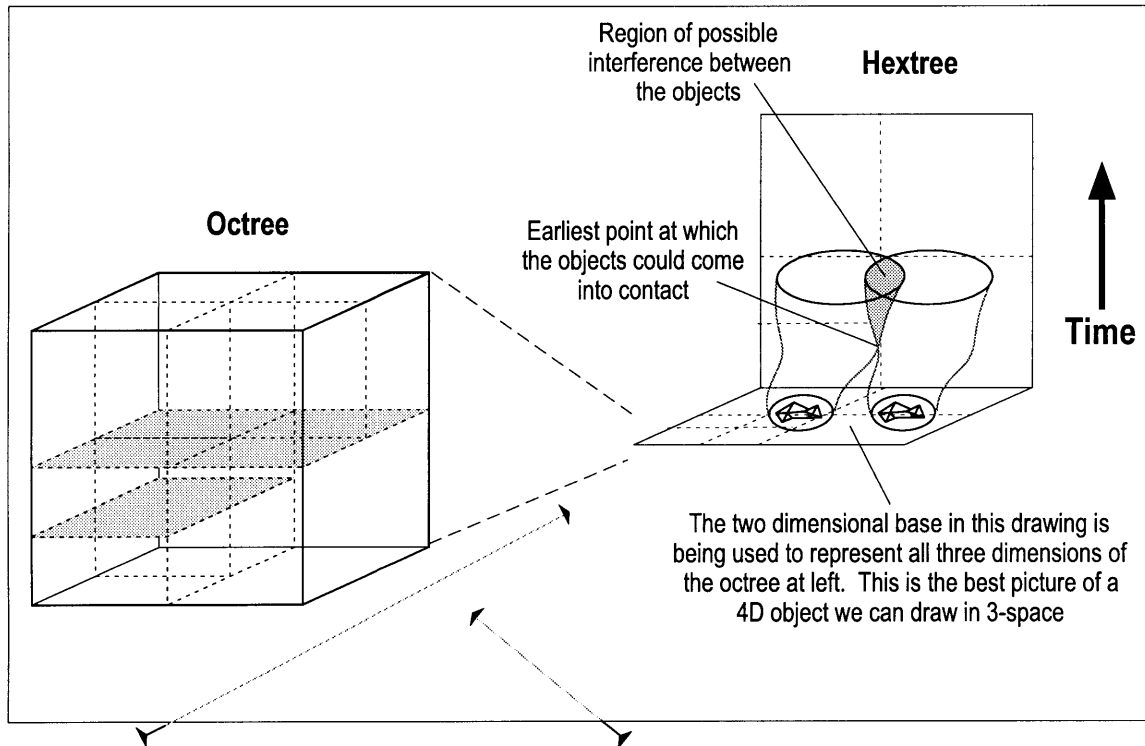
8.0 Four Dimensional Analysis

As noted in Appendix J, the majority of the cost of sparse dynamics is incurred comparing object trajectories at event times. Clearly this could be made more efficient if we had some notion of objects' relative proximities. The difficulty is that the nearness relations change over time. In a moving environment, static proximity relations will require constant updates.

Four dimensional analysis provides a way to overcome this problem. As an object moves, we can think of it as tracing out a 4D tube through time. Although complicated, this four dimensional shape is of fixed form over all time. Use of 4D analysis reduces the space-time collision detection problem to one of static interference detection. If we calculate the earliest time a pair of tubes intersect, we have found the first possible meeting of the two objects. This idea is described in [Cam90].

Note that 4D analysis is not the same as computing the intersections of swept volumes. Swept volumes are compared in only three dimensions. Any intersection found in this way must still be checked to determine if the objects were in the same place *at the same time*.

We can model the error term as causing an object's path to spread out into a field of possible positions as we move into the future. This represents our growing uncertainty of the object's position as time progresses. The tube traced by the object thus spreads into a cone. The larger our error, the faster the cone's spread. Methods similar to this have been discussed in [Hub93, Foi90].

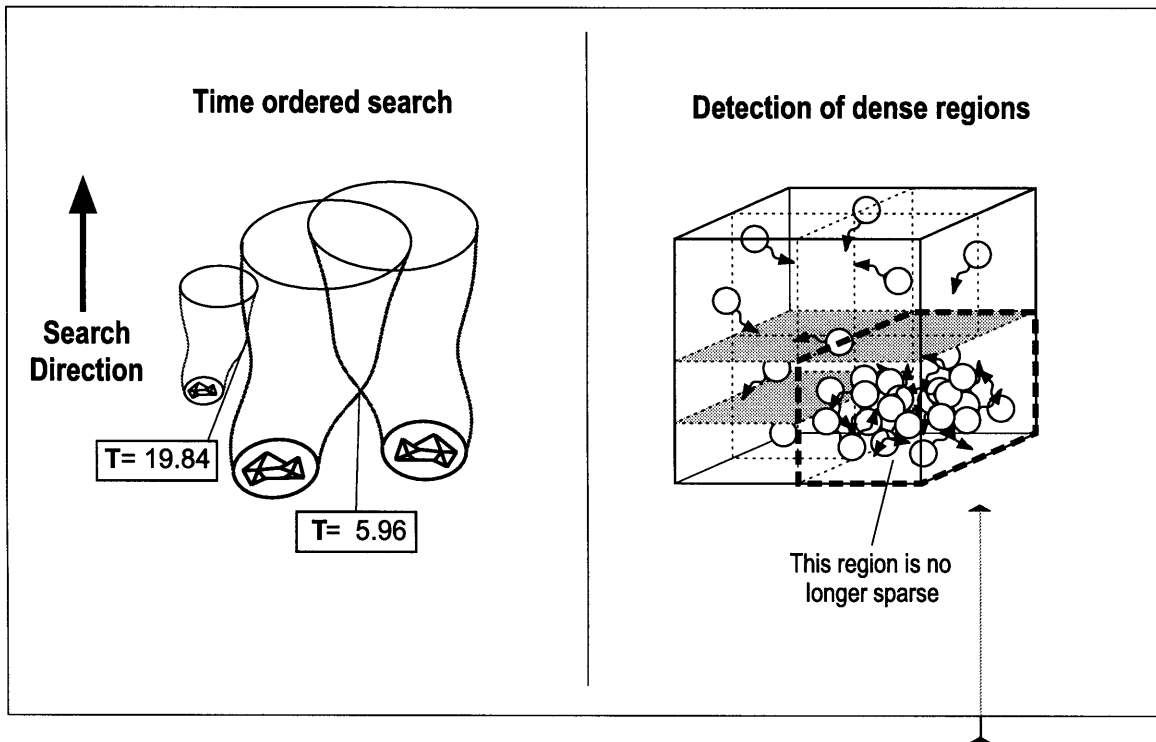


8.1 The Hextree

Our path intersection method from Section 3.5 has already been doing four dimensional analysis implicitly. The new concept introduced here is that we can explicitly model the shape of the swept cones and enter them into some form of hierarchical data structure. We can then refer to this structure to obtain the proximity of elements both now and in the future. This structure need be updated only in the case of collisions, which as we have noted, are rare.

A typical data structure for representing object proximities would be a spatial subdivision tree whose leaves contained element positions (e.g. an octree). Unfortunately, this would require us to update the tree at each frame-time.

This can be remedied by moving to the four dimensional equivalent of an octree, generally termed a *hextree* [Gill81, Ben75]. The extra dimension of this tree is used to represent the passage of time. Each element's representation in this space-time continua is a smear representing the space that the element will occupy as time passes. Similar methods are explored in [Duf92].



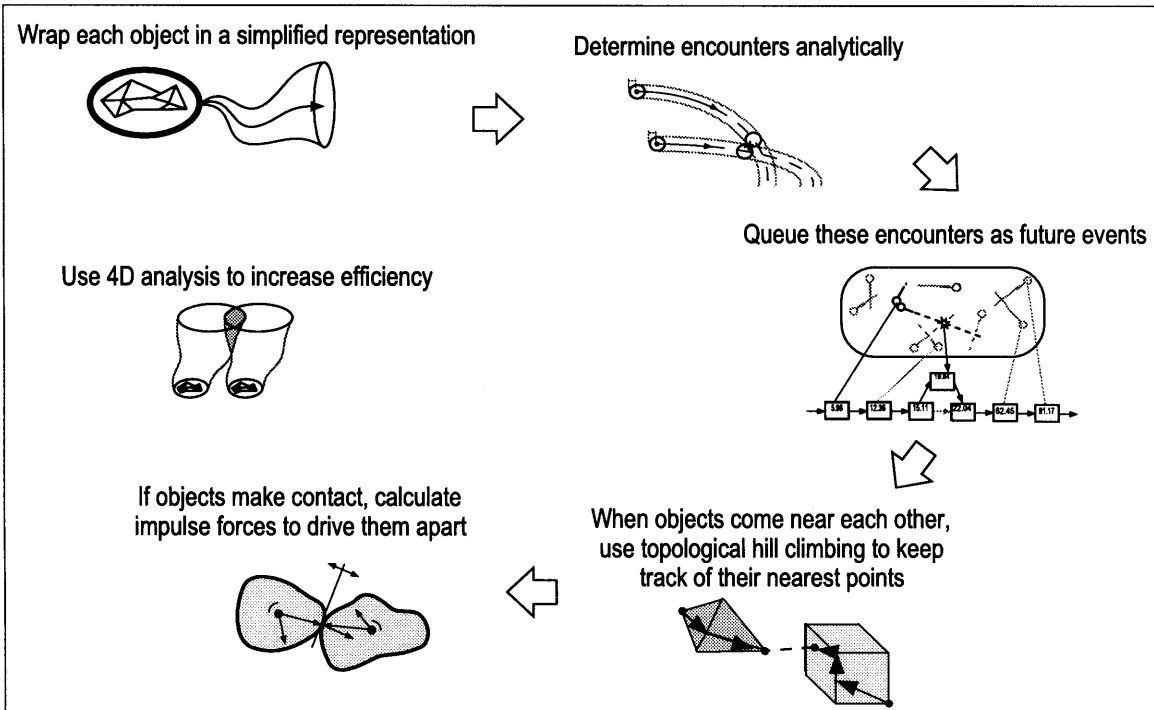
8.2 Hextree Use

If we use the hextree to check an object's neighbors in order of space-time proximity we can stop as soon as the first hit is found. Methods for such searches are well established, though they will need to be extended somewhat for four dimensions.

A preliminary hextree has been implemented as described. There is a fair amount of overhead in keeping the tree up to date. Results show the method breaks even at about 500 objects. With more than this, efficiency is improved. The implementation is currently quite naive and doubtless the break even point could be lowered considerably.

The Dense Case. Sparse analysis loses efficiency in highly populated environments. In such situations, many collisions are occurring and object positions are difficult to predict. The hextree can be used to identify such situations. Highly populated regions of space appear as particularly deep tree hierarchies. If the simulation is overburdened by the cost of a dense region, we can degrade the model of physics being applied in the region in order to maintain real-time operation. This is discussed in more detail in Section 9.4.

Summary of the sparse dynamics method

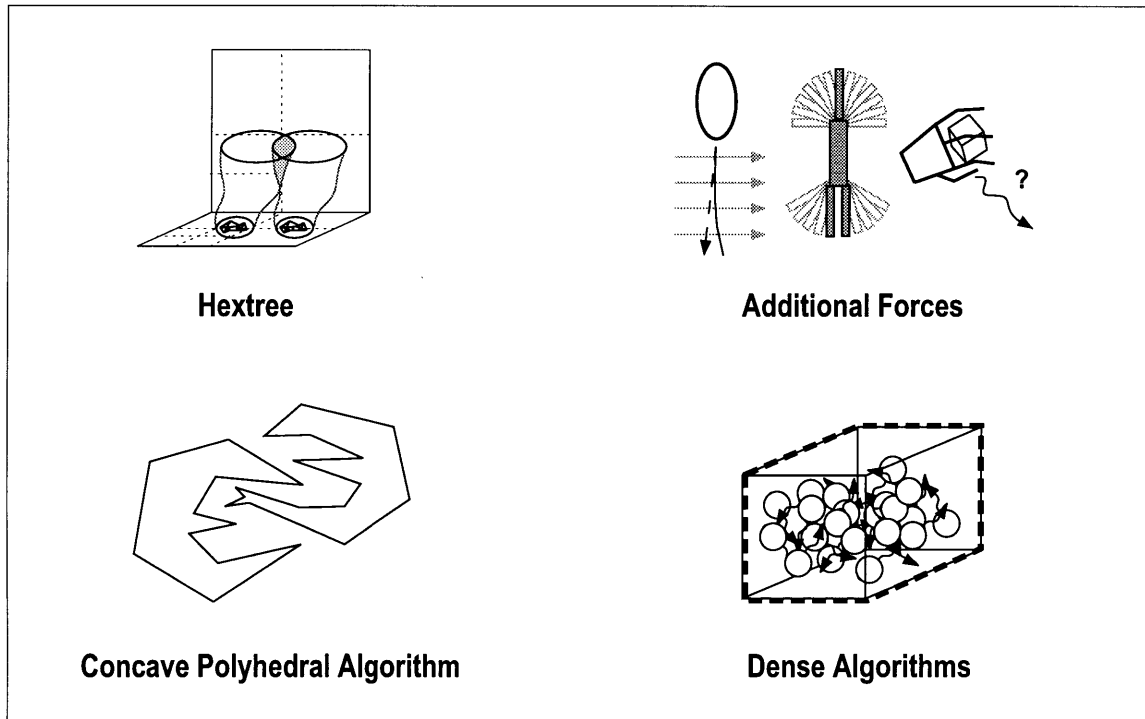


9.0 Discussion

9.1 Summary

In this thesis, we have presented a system for efficiently modeling the motions of large numbers of polyhedra. We established that a majority of the time in such systems is spent searching for object collisions and so introduced a new model to streamline that process. In the case where objects actually do make contact, we employed a hill climbing graph algorithm to quickly locate the point.

We have discussed how to handle linear and accelerational motions, wind resistance, and user interaction. We also introduced a model which attempts to handle extended contact and the effects resulting from it. We have tested the algorithm to ascertain the validity of its operation and finally suggested a time/space hierarchy which could be used to make the method more general and efficient.



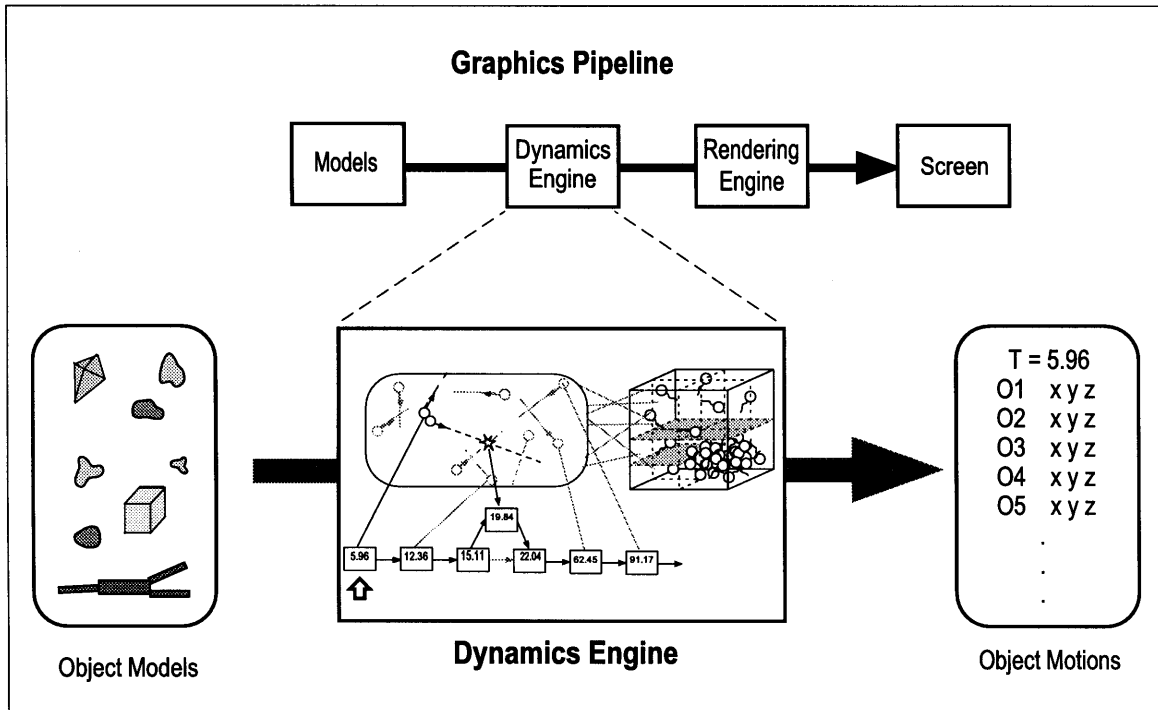
9.2 Future Work

The largest project currently underway is the completion of the hextree subsystem. A large amount of theoretical work remains to be done to establish the various formulas of optimality associated with the tree's operation.

We would also like to make the algorithms for predicting collisions more accurate and robust. Many speed improvements are possible, though we are not currently well motivated in this area. The algorithm now runs sufficiently fast for our current purposes, and in any case, faster than the renderer.

A more important concern is to increase sparse dynamics' generality, its handling of extended interactions, and the types of object representations it can accommodate. Dense interactions also need a more thorough treatment.

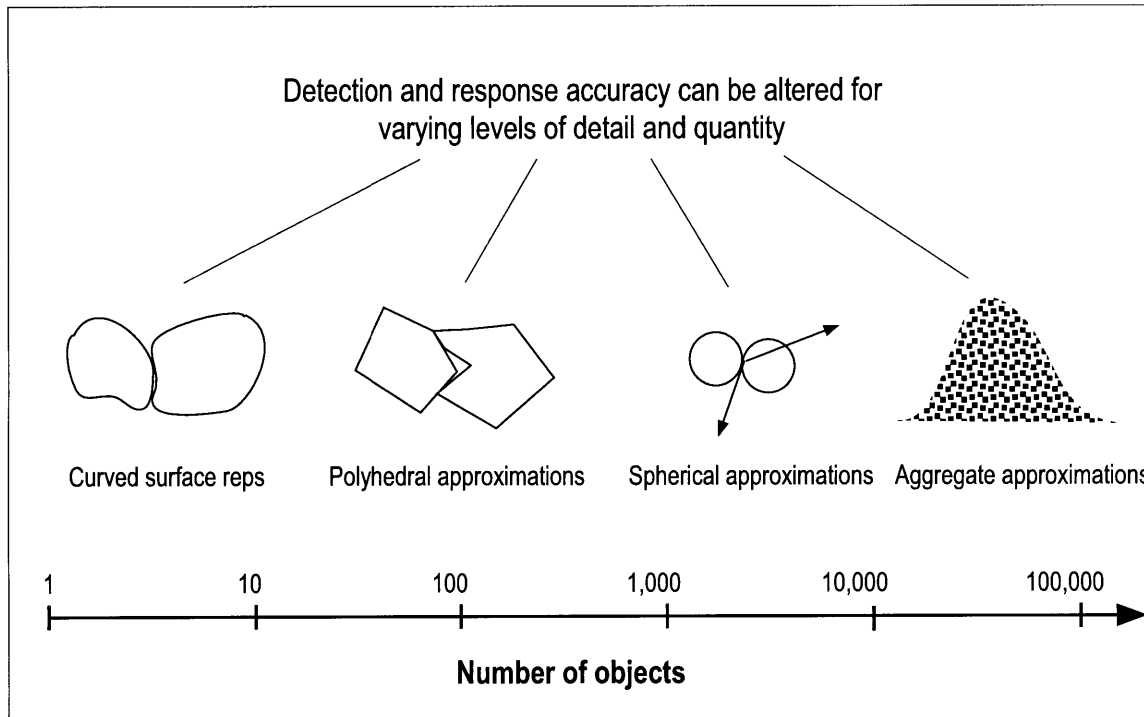
The next few pages describe some longer term goals for the work.



9.3 The Dynamics Engine

We envision the result of this work as a black box *dynamics engine* into which we feed our simulation model, and from which issues periodic reports on the state of that world. This mechanism would be similar to the rendering engine which workstations now contain and could, in fact, precede it in the graphics pipeline. The renderer would contain a simplified dynamics model analogous to the sparse model. It would be independently capable of updating the analytic positions of objects for rendering purposes. This would be interrupted periodically by the dynamics engine when events or user input altered an object's predicted course. The bounding spheres would also make the renderer's job easier, as it would know which objects were likely to overlap during rendering. Some of the rendering hardware already available could likely be turned to these purposes.

The goal of providing workstations with such a dynamics engine is to allow the use of dynamics to become as ubiquitous as rendering is now. Many designers would find applications for such a capability, if it were readily available.

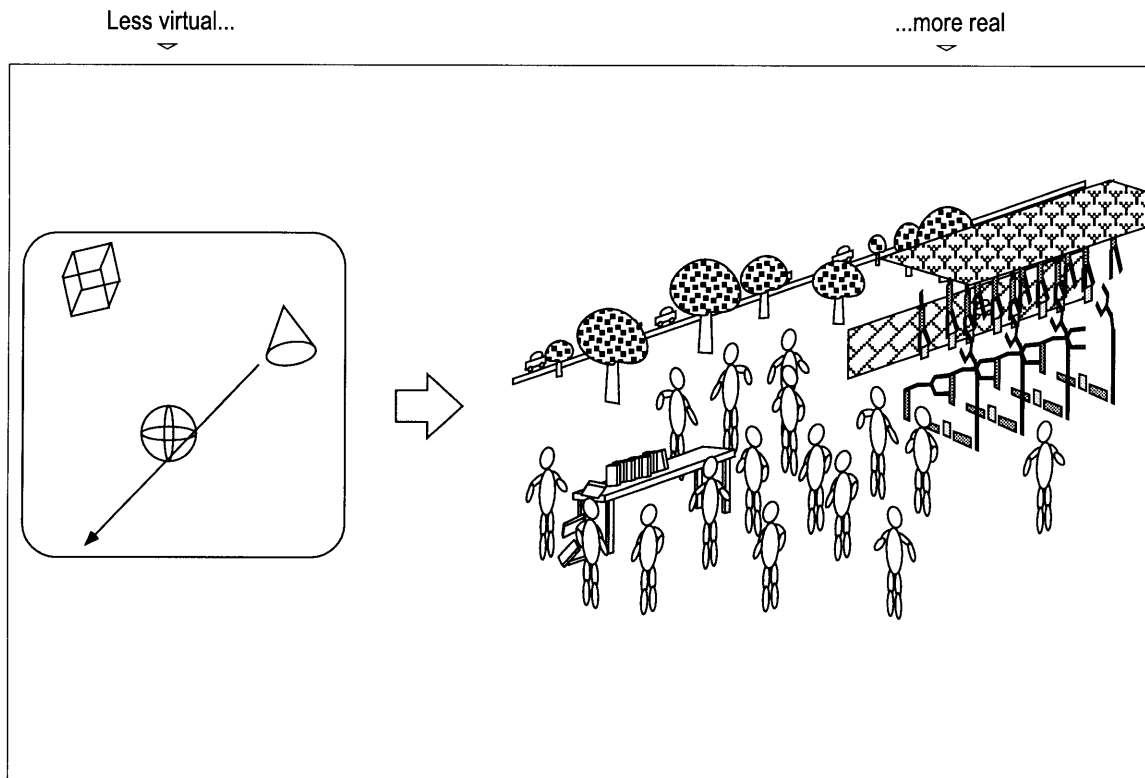


9.4 Other Levels of Detail

Two levels of detail, sparse and detailed, have been described here. There could be others, however, to take advantage of coherencies at other scales. The sparse and detailed levels were both “exact” methods in that their efficiencies did not result in imprecision in the simulated outcome. However, as the scale grew larger, this would be of less importance. Even if a simulation is to be very precise, the scientist is unlikely to care about the position of every particle among thousands. What would be important is the aggregate behavior. This behavior is susceptible to characterization and modeling just as object trajectories are. The issue is to have a sufficient model at each scale to accurately present the physical behavior. It is the attention of the user which should inform the system where to expend its resources. Two example levels of detail:

- We can treat small objects as spheres with rotations. When colliding, we can look up the objects’ faces in a table to determine the proper deflection angle. This method is analogous to the rendering technique, bump mapping.
- Dense areas can be treated as a single aggregate mass. This mass would be analytically modeled as a single flexible body with certain properties and cohesiveness.

The ultimate goal would be to make the dynamics engine *scale independent*, that is, capable of simulating a physical environment of any size, time-scale, or density. When few objects were present, detailed and accurate motion would result. As the scale or number of objects increased, the engine would progressively fall back on more global or aggregate techniques. Although individual accuracy would be lost, the mass behavior of the system could still be observed.



9.5 Conclusion

We have described a new model for simulating dynamic interactions. The model takes advantage of previously unexploited coherencies to make real-time simulation tractable for usefully sized environments. In all, we believe that the widespread availability of a real-time dynamics engine could have as much impact on graphics research as rendering advances have had in the past.

See the appendix for a description of the sparse dynamics code.

9.6 Acknowledgments

This work was made possible by the contributions of many people, the most important of whom I hope to thank more personally than with just a notice.

David Sturman provided the initial inspiration which started me on this path of research. If you like the format of the document, you may thank Mark Lucente and Steven Drucker for their inciteful discussions on form and content. Mr. Drucker is also to be commended for unflagging mathematical support. I am likewise grateful to Sundar Narasimhan for his initiation into the arcane world of numerical optimization. Thank are also due Linda Peterson who has provided much more than just administrative assistance.

My advisor, David Zeltzer, has exhibited the patience of kings and I can only hope the finished product proves compensatory. I am indebted to Ming Lin, John Canny, and Dinesh Manocha for both their research and ongoing discussions. Finally, I owe the deepest of thanks to Diane Judem for assistance without which there would be nothing to be thankful for.

> See Appendix M (p.60)

Appendices

A.0 Discussion of Formatting Issues

The format of text has remained basically unchanged since the invention of set type [Gut1455]. Lines of text scan side to side, top to bottom, providing a linear narrative punctuated by serial headings. While any discussion can be forced into such a form, some are not well served by it. Those topics whose nature is graphical-- that is to say spatially composed-- are more likely to benefit from a pictorial description. However, due to typographic difficulties, pictures have been relegated to a second place status-- providing backup for what is explained in print.

Recent instrumentation has made it possible to rethink this hierarchy, and this thesis is an attempted example. Our topic is particularly suited to a predominantly pictorial format as it is wholly graphical in nature. Having spent a fair amount of time casting about for the “right way to put something” it is a relief to find that a few deft lines may often do the trick.

In addition to relieving the author of some difficult prose, the form has other virtues. Although intended mainly for paper, this format would easily lend itself to on-line use and manipulation. Identifying parts of illustrations with pieces of text is useful as an automated searching tool as well as providing some description of picture contents for indexing purposes. In an age rife with visions of multimedia and hypertext systems, it seems odd not to try applying them to this most basic of tasks.

This is hardly a call for abandonment for the printed word in favor of pictograms, rather each media should be used in light of its strengths. Pictures provide a quick and intuitive understanding, while words are more specific and precise. Thus the form of this thesis, in which we lead with pictures and back them up with technical appendices.

While its wild success I rather doubt, I will be content if the experiment at least does not hinder the reader’s comprehension. I feel some leeway to attempt a thing of this kind seeing as the work was done at the *Media Laboratory*. Experiments in the format of written media would seem well within its chartered territory.

A.1 Sources

My personal experience is that I can not learn a concept until I have a concrete picture to hang it on. Even if the notion is purely mathematical, I require a terrain in which to set the concepts. I recall technical papers by their characteristic illustrations, even if the illustrations are not germane to the problem at hand. When reading a technical document, I suspect that we all look at the pictures first. Yet the feeling lingers that this is “too easy,” that a proper comprehension must be paid for by slogging through difficult prose.

I was made aware of the power of the graphical format by Gonick's series of pictorial introductions to various technical fields [Gon90, Gon91]. His *Cartoon Guide to Genetics* particularly, instilled in me more lasting biological comprehension than had years of scholastic endeavor.

A particularly vivid example also comes to mind from my collegiate Algorithms course. After spending a week puzzling through various categories of sorting methods in Sedgewick [Sed83], we were shown a short animation entitled *Sorting out Sorting*. In ten minutes, the concepts went from nebulous to obvious.

Another inspiration in this regard is Edward Tufte and his books on the pictorial display of information [Tuf83, Tuf90]. He says "What is to be sought in designs for the display of information is the clear portrayal of complexity. Not the complication of the simple; rather the task of the designer is to give visual access to the subtle and the difficult-- that is, the revelation of the complex."

In a more practical vein is the periodical *Scientific American*. Its articles are written in such a manner that their main points can be understood from study of the illustrations alone. I had read it this way for years before I discovered this feature was intentional. I think we may therefore term it an intuitive interface.

A source closer to hand makes a similar point. Concerning his recent book, *Society of Mind* [Min86], Minsky has said that, given the thickness of his topic, he was loath to introduce more than one concept per page. This does not mean the book reads slowly. A single idea can contain details, corollaries, and ramifications. But by leading with a single idea, the reader is given a context in which to place later impressions.

A skeptical reader might question whether a lavish use of illustrations serves merely bulk up the document and so avoid some "hard work." In reply, I can say that the time spent in construction and layout of the illustrations was at least comparable to that needed to write an equivalent body of prose. The advantage to be gained (one hopes) was not a shortening of my time, but an improvement in the reader's comprehension. As to the thickness of the document, I note that my page count has been *reduced* as a result of adopting this approach. I came to realize that, with appropriate illustration, descriptions could become quite succinct. As the master of eloquence William Strunk puts it

Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentences, for the same reason that a drawing should have no unnecessary lines and a machine no unnecessary parts. This requires not that the writer make all his sentences short, or that he avoid all detail and treat his subjects only in outline, but that every word tell.

[Str79]

No more need (or can) be said.

B.0 Notes on Virtual Reality

The illustration on page 7 depicts the current state of the art in virtual environment research. We hope that this may seem amusingly quaint to future readers. Removing the hyperbolic trappings currently attending the field, we see the following themes:

- few objects (there always seems to be 3)
- crude representations
- a gesture at physics, if any

Reality, as those who have spent any time there can attest, does not look like this. The physics problem in particular is not going to be easily cudged with increased processing power. We hope in this thesis to bring subtler methods to bear.

The term *virtual reality* itself has taken on a somewhat unwholesome air of late. The problem is an old one-- we have confused what we can do with what we would like to do. In this case, the lure of the imaginary is quite strong. The notion of making dreams into reality is one of humanity's most deeply held. We eagerly embrace any technical advance which might purport to legitimize it.

While this yearning is forgivable, when it works its way into our grant proposals and press releases, it is less so. Not only do we breed false impressions, but we also impede the actual research itself. By promising technology we can not deliver we assure a cold welcome for the hesitant intermediate steps resulting from our pursuit of it. The nuts-and-bolts of simulated environment research seem dull by contrast to the fiction, and so languishes.

This is a case where the concept has run off independent of its applications. It has attained cache independent of any utility. As an example, compare virtual reality technology to that of the typewriter. Could one honestly propose that the former is a more important innovation? Yet we see the typewriter as a humble tool, pursuant of many worthy endeavors, but itself embodying none.

As such, the industry has all the makings of a fad. Reading this in later years, you will be able to judge whether we abandoned such conceits, or the field altogether.

C.0 Related Work

There are currently a number of algorithms to compute inter-object collisions. Moore & Wilhelms [Moor88] give a good overview of the situation, Wilhelms also providing a valuable general introduction in [Wilh87].

To describe the field, we divide the collision problem into parts as in Section 2.2. *Collision determination* refers to establishing likely candidate pairs. *Collision analysis* refers to location of the point in space and time at which the pair collide. *Response* deals with deriving the forces which result from such contacts. The categories can not be said to be distinct and many of the cited papers

deal with more than one of them. We maintain the distinction here so that algorithm costs can be compared to others of like kind.

Collision Determination. Hopcroft introduced an $O(N \lg^2 N)$ algorithm for finding intersection among a set of N static spheres [Hop83]. Cameron introduced four dimensional analysis to handle dynamics in [Cam85] and refined it in [Cam90].

The notion of placing bounds on object motions and predicting future events was discussed by Culley and Kempf in [Cul86]. Foisy [Foi90] follows in a somewhat similar vein. The queueing work presented here can be seen as an outgrowth of these. Similar work is being pursued by Hubbard [Hub93].

Collision Analysis. Boyse described an $O(N^2)$ algorithm in [Boy79] which used an all-to-all comparison between the features of a pair of polyhedra. The algorithm handled static interference and limited forms of motion. In [Cam86], Cameron and Culley reported an algorithm which applied configuration space from Lozano-Pérez [Loz83], and was estimated to be $O(N \lg N)$. Canny also offered a configuration space method [Can86], requiring $O(N^2 \lg N)$, but able to handle arbitrary object motions and configurations.

Gilbert introduced a nearly linear time algorithm in [Gilb88] which converges on the smallest distance between a pair of polyhedra by searching for optimal support planes. The method is extended to some types of curved surfaces in [Gilb90]. Bobrow used constrained minimization with similar intent resulting in an $O(N)$ algorithm [Bob89]. Bobrow's method was refined by [Zeg91] and extended to curved surfaces in [Ma92]. [Nah93] improved on the execution times of this using object velocities to predict future collisions.

Lin and Canny make use of preprocessing in order to reduce a search similar to Bobrow's to nearly constant time [Lin91]. The collision analysis in the current work is based on this last method.

On the theoretical side, Dobkin and Kirkpatrick reported an $O(\lg^2 N)$ method for detecting if convex polyhedra intersect [Dob83] and later a linear time algorithm for finding the minimum distance between them [Dob85]. These results required a preprocessing step in order to generate hierarchical object representations. Most recently they have extended this work to provide an $O(\lg^2 N)$ method for minimum distance, requiring at most linear preprocessing time [Dob90]. The suitability of these algorithms for mechanical implementation, however, remains unclear.

To speed up collision identification, Thibault and Naylor use a hierarchical BSP tree to provide efficient access to object features [Thi87]. [Van91] unifies this with boundary representation information to simplify operation. Turk makes use of a uniform spacial subdivision coupled with bounding sphere trees in [Tur90]. Bounding sphere trees have also been employed by [Pob92] and [Hub93] to gain a similar speed advantage. More recently, interval methods have become popular [Duff92, Sny92, Sny93]. These provide unprecedented capability and accuracy but dramatically increase computation times.

Concerning object representations other than polyhedra, Sclaroff and Pentland report good results with deformed quadrics [Scla91]. There has also been recent interest in calculating collisions among time dependent shapes in [Von90, Bara92, Syn93, Gas93].

Response. Although not directly pertinent to this work, the area of collision response has also received much attention [Hahn88, Bat85, Will88, Bai93, Bra91, Gold60, Mac60].

Baraff examines the computation of response forces for various situations [Bara89, Bara90, Bara91, Bara92]. However, his methods encounter some difficulty due to point sampling inaccuracies and instability and cannot be considered generally applicable.

An earlier version of the work presented here appeared in [Dwor93].

D.0 Discussion of Sparsity

Sparse vs. dense. We may think of the two cases by analogy. Molecules in a gaseous state are in a sparse environment. By contrast, liquid molecules are in a dense one. The line of distinction between the two cases is arbitrary. What differs is the techniques used to analyze them. By choosing the appropriate model in a given situation, we do not alter the result, but we decrease the amount of computation needed to arrive at it.

The density of environments used for testing in this work ranged from 5% to 10%. At this level, sparseness was greatly effective. We think it likely that typical simulated environments will be *less* populated than this. A glance around your office will confirm that it is mostly air. However, there will likely be certain regions which are far in excess of this. In such regions it is unhelpful, and perhaps even deleterious to employ the sparse methods. Predictions will often be wrong, necessitating small time steps and frequent recomputation. Identifying such regions is addressed in Section 8.2.

Sparse vs. detailed. We may also think of the sparse case as being “higher level”. It examines the gross movement of objects, leaving further analysis for the detailed phase. We identify only two levels of detail for objects in this work. However, there could be more. We imagine cascading levels of representations, each containing only the detail needed for that level of checks. We would require adaptive methods for determining what level of detail was appropriate at any given time. The hierarchical structure of Section 8 would also be of use for this.

E.0 Root Convergence

The actual motion of objects in a physical environment is described by ordinary differential equations. We use a numerical integration method, such as Runge-Kutta [Pre88] to determine their

position over time. For the predictive model, we make the simplifying assumption of constant velocities or accelerations. This allows us to use algebraic root-finding methods instead. The approximation is kept conservative through the use of an error bound.

For low order motion, we may solve for these roots analytically. Otherwise, convergence must be used. We must make a distinction between the two kinds of convergence we employ. Numerical root-finding is used to approximate a solution to our path equation. But the equation is in turn an approximation of the object's true motion. Therefore, even if our root-finder is quite accurate, we still may not have located the new contact point. To assure finding it, we must sometimes home in on the time of the collision, taking progressively smaller time steps as we approach it. This is essentially a linearization method common in nonlinear control theory [Cook86].

The use of this method allows the root-finder to be less than perfectly accurate, however. In particular, we can indicate to it how much accuracy we need at any given invocation, depending on how far off the event we are predicting is.

The difference between these and conventional techniques is that the convergence on each collision is carried out by locally adaptive time steps. The progress of the engagement is unrelated to other calculations occurring elsewhere. Also, the time steps used are far from uniform. They, in fact, zero in on the event in a manner like Newton's method, typically by a factor of 2 or more per step.

E.1 Numerical Root Finding

We require a numerical technique for locating the roots of polynomials. We implement Laguerre's Method, a simple iterative root finder [Pre88]. Its cost is a great deal higher than the analytic solutions in the next appendix. However, its speed of convergence is not greatly affected by increases in polynomial complexity. This allows us a great deal of freedom in adding new terms to the path equation.

E.2 The Error Term

We use the error term to accommodate several types of deviations:

- higher order small effects
- uncharacterizable effects
- future indeterminacy
- left over error from path approximations

If the spread itself is not guaranteed conservative due to unbounded indeterminism or noise, we must use a periodic out-of-error-envelope check. The frequency of the check will depend on how fast the path can diverge. If we have no idea, then we must make a check each integration step or sample.

F.0 Analytical Solutions

Our function of motion will have the basic form

$$P(t) = x + xt + \frac{1}{2}at^2 + \dots$$

As our motion model becomes more accurate, this equation will gain additional terms. To find the meeting time of a pair of elements, we solve for the times when the distance between their position functions is less than the sum of their radii

$$\text{Dist}(P_1(t), P_2(t)) \leq R_1 + R_2$$

Including the error term results in

$$\text{Dist}(P_1(t), P_2(t)) \leq (R_1 + E_1(t)) + (R_2 + E_2(t))$$

F.1 Linear motion

Given spheres of radius R_1, R_2 traveling along linear trajectories $P_1 + V_1t, P_2 + V_2t$ with possible deviation dR_1, dR_2 per unit t , they intersect if

$$\text{Dist}(P_1 + V_1t, P_2 + V_2t) \leq (R_1 + dR_1t) + (R_2 + dR_2t)$$

Substituting $R=R_1+R_2$ and $dR=dR_1+dR_2$ this becomes

$$\sqrt{(\Delta P + \Delta V) \cdot (\Delta P + \Delta Vt)} \leq R + dRt$$

$$(\Delta V \cdot \Delta V - dR^2)t^2 + 2(\Delta P \cdot \Delta V - RdR)t + \Delta P \cdot \Delta P - R^2 \leq 0$$

We are interested in the extrema of this case in which the spheres are just touching. This converts the inequality to equality. The result is a second order equation whose zeros can be found with the quadratic formula

$$t = \frac{-\Delta P \cdot \Delta V + RdR \pm \sqrt{d}}{\Delta V \cdot \Delta V - dR^2}$$

$$d = (\Delta P \cdot \Delta V - RdR)^2 - (\Delta V \cdot \Delta V - dR^2)(\Delta P \cdot \Delta P - R^2)$$

If the discriminant d is negative, the solution has no real roots, indicating the elements never meet. Otherwise, the two solutions will represent the point at which the spheres first touch, and the point at which they have fully passed through each other. Since we want the point of first contact, we select the smaller root.

If the result is earlier than the current simulation time, the elements are moving away from each

other. I.e., their point of first contact was in the past and they can not collide. Derivations similar to the above can be found in [Yang, Ald59].

F.2 Gravity

In the general case, objects are in free fall, and experience a force according to the sum of the masses around them. For a method of implementing this cheaply, see [App85]. In this environment, surface effects such as support and friction are rare. When objects make contact, they quickly fly apart again.

Surface effects appear where objects are near a body much larger than themselves. This provides an undeviating force vector (“down”) and a surface to which they are drawn (the “floor”). Although atypical, this situation is of particular interest to humans and so deserves special attention.

A single gravity vector costs nothing to add to the model. In a reference frame where all objects experience the same force, the acceleration terms cancel out in the equations. It is not until an object encounters the floor that we have difficulties. At that point, the floor exerts an addition force. This can be treated as a special case of the next section.

F.3 Thrust

In a physical environment, objects can alter their motions by throwing away part of their mass. We coin the term *rocket* for any object capable of generating its own acceleration in this manner.

Accelerating objects are harder to simulate than the linear case. The derivation is similar to the above, but second order terms are introduced into object motions. This results in a 4th order equation to determine the intersections of a pair of them.

$$\begin{array}{ll} \text{Position:} & P_1 + V_1 t + \frac{1}{2} A_1 t^2 \quad P_2 + V_2 t + \frac{1}{2} A_2 t^2 \\ \text{Radius:} & R_1 + dR_1 t + \frac{1}{2} ddR_1 t^2 \quad R_2 + dR_2 t + \frac{1}{2} ddR_2 t^2 \end{array}$$

$$\text{Dist}\left(P_1 + V_1 t + \frac{1}{2} A_1 t^2, P_2 + V_2 t + \frac{1}{2} A_2 t^2\right) \leq R_1 + dR_1 t + \frac{1}{2} ddR_1 t^2 + R_2 + dR_2 t + \frac{1}{2} ddR_2 t^2$$

Again, substituting $R=R_1+R_2$, $dR=dR_1+dR_2$, and $ddR=ddR_1+ddR_2$ this becomes

$$\begin{aligned} & \frac{1}{4} (\Delta A \cdot \Delta A - ddR^2) t^4 + (\Delta A \cdot \Delta V - dRddR) t^3 + \\ & (\Delta A \cdot \Delta P + \Delta V \cdot \Delta V - RddR - dR^2) t^2 + \\ & 2 (\Delta V \cdot \Delta P - RdR) t + (\Delta P \cdot \Delta P - R^2) \leq 0 \end{aligned}$$

Exact 4th order solutions exist for this [Bey84]. As they are sufficiently complicated, we have

found it simpler to solve this via a fast converging iterative method [Pre88]. This has the advantage of adjustable accuracy, and generalization to the addition of higher order term.

The first and third solutions returned will represent the two times the objects meet. For prediction purposes, we select the first of these that is past the current time. If neither are, then the objects have no future meeting.

In a gravitational environment, the floor can be implemented as a “rocket object” of this type. It exerts a constant upward thrust sufficient to cancel the effects of gravity. It appears to hold its position in space while all objects fall toward it.

G.0 Handling Extended Contact

A non-moving floor (i.e. one which appears to receive none of our momentum on impact) introduces the effects of support and friction. These can theoretically be simulated by the system already in place-- multiple small collisions will sum to the correct overall forces. However, as our method is predicated on the rarity of collisions, it is inefficient to do this. It is preferable to sum up these interactions over time and determine their effects analytically. The effects of these forces have been much studied [Hahn88, Bat85, Bai93, Bara89, Bara91].

A proper treatment of this topic is beyond the scope of the current work. The method presented here is along the lines of Hahn’s, and is intended as the first step in a more general scheme. It produces reasonable motion for small systems and is not unduly expensive. In the long run, we wish to characterize the effect of contact forces and include them in our path-and-error model or in some other phase of the system. We believe that the models provided by sparse analysis would facilitate work in this direction.

Temporary Joints. We note that force is transferred among objects only at discrete regions of contact. We can characterize these regions by object features similar to the methods of Section 5. Each class of motion is then handled analytically.

An object in 3-space with no restrictions on it has 6 degrees of freedom: 3 translational and 3 rotational. The object’s contact with another surface can constrain from 1 to 3 of these degrees. When an object meets the conditions for such a constraint, we explicitly store the contact as a temporary kinematic linkage. Its behavior about the contact will then emulate a robotic joint of that order.

We distinguish three levels of contact “joints” (refer to the illustration on page 19):

- *One-point* contacts rob an object of a translational degree of freedom by forming a barrier to its movements. The region of contact is a point.
- *Two-point* contacts remove a rotational degree of freedom by creating a hinge at the point of contact. The region of contact becomes a line.

- *Three-point* contacts remove another degree of rotation. The region of contact is a polygon. This can also be termed a *support contact*. Note that the term “three-point” is used to encompass three or more actual points of contact.

If an object establishes a contact point with an object that it already shares one, its joint class is increased by one. If a contact point is broken, the class is decreased. Joints move up and down between classes in this way.

In the case of three-point contacts there is the additional question of determining balance. An object is balanced when its center of mass (in the direction of its acceleration) runs through the polygon formed by its support points. If this is not the case, the object will tip in the direction of its center of mass and at least one of its contact points will immediately break. One object’s balance on another must be considered for the whole of both kinematic structures in order to allow for towers, arches, and the like.

A useful trick to reduce the number of checks: if the relative motion energy between two objects is low, they may be checked less often, and if at rest, they may be marked as static relative to each other. For the duration of such a contact, they act as one body and transfer external forces between them.

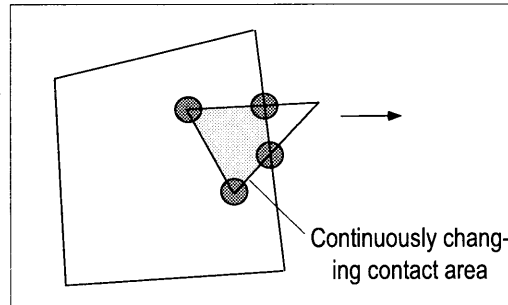
G.1 Friction

Even when constrained by a surface, an object has three other degrees of freedom in which to move: two translational axes and one spin. Objects may slide and twirl against one another, still constrained by their contacts. Friction is the force which opposes this motion. According to the Coulomb model [Bra91], friction is a function of the roughness of each surface, the area of contact, and the force with which they are pressed together.

For one and two point contacts, we must apply a trick. Theoretically, these objects have no area of contact, and thus, no friction. However, edges and corners in the real world are slightly rounded. We therefore attribute a small surface area to edge and corner contacts in proportion to the elasticities of the engaged objects.

We can cheaply calculate the area of the region covered by a contact. Combining this with the contact force and the roughness of each surface, we can compute the appropriate sliding resistance. We determine the acceleration resulting from applying this force at the contact point, perpendicular to its vector. We then add this acceleration term to the path equation in Appendix F.3.

Unfortunately, the area of frictional contact can change continuously over time resulting in a differential equation that is difficult to analyze ahead of time. We currently use a simple approximation that computes the area at the beginning of the contact and at the point where the contact's nature changes. We assume a linear interpolation of the friction force between these two times. This results in a third order term being added to $P(t)$, requiring a sixth order equation to predict intersections.



Each time the contact shape changes form, we must recompute the acceleration term. This includes shifts between joint levels, and changes of contacting features within a given level. We use the path equation to predict when these will occur and queue that time as an event. As always, our prediction may be too conservative, requiring several events to home in on the actual shape change point.

H.0 Adding Other Forces

Additional forces can be included by adding appropriate terms to $P(t)$ and the error term.

H.1 Mechanisms

This includes jointed figures and machinery. Under sparse dynamics, these are considered single objects. An articulated figure can not alter its center of gravity while in free fall, it can only alter its trajectory when in contact with another surface. This is not a problem since the sparse bounding volumes have already been violated by the time the figure contacts another surface.

The problem is thus reduced to finding the internal motion of a jointed figure. This is easier than the general case for the same number of sub-elements as the motions are somewhat constrained. For treatments of this problem, see [Sch91, Jos90].

H.2 Aggregate Forces

Even chaotic forces can be estimated analytically in aggregate. These are forces which are unpredictable on a small scale, but have well defined laws to express their overall effects. A common example is motion through a viscous medium in our case, termed "wind resistance." Wind resistance is a force which resists motion in proportion to its magnitude. As such, we can form an accelerational approximation for it to be added to $P(t)$. For increased fidelity, the term can be made a function of altitude or air pressure, or it can make use of lookup tables to simulate the effects of air currents and weather.

The ability to handle aggregate behaviors is also important in other respects. In Section 9.4,

we wish to gain scale independence by deriving a high level model of colliding objects. The behavior of aggregate bodies is used as the basis for this.

H.3 Unpredictable Forces

Some forces can not be predicted due to their internal volition. That is, the objects have the ability to accelerate from within and employ this ability according to some unknown algorithm. The motions of these can derive from outside measurement or be handed to us by another module. Clearly such objects are not amenable to exact prediction, and thus pose an efficiency problem.

The most nettlesome type of indeterminate objects are those under control of the user. Users often desire to have a “hand” in the simulation which follows their specified movements. We term these *god objects* as they have no physical basis and violate the rules of the simulation. They are explicitly defined to be the hand of god reaching into the simulation and are unconstrained by its physical laws. The solution is to put limits on those objects’ activities. If this troubles the reader, consider-- an environment where an unpredictable agent can reach in and change any aspect, at any time, is not a simulation at all. Since it obeys no consistent rules, what can it be said to model? As a simulation becomes more sophisticated, the ways in which an outside observer can effect it must become higher level and more subtle¹.

So, the user's activities must be constrained in the ways they can affect the progress of the simulation. A workable method is to make the user a “rocket object.” That is, the user becomes the agent which makes the internal thrust decisions. For a typical “floating hand,” this means that fast movement will cause the screen hand to lag somewhat behind the real one. The user can violate this restriction, but at the cost of efficiency. As an example, imagine the user moving instantaneously from one location to another. The hand's disappearance will mean that any object which intended to encounter it will wake at that location with no engagement and have to recompute. The appearance of the hand elsewhere will require an immediate cull of the local neighborhood to determine *its* next encounter.

Even so, the loss of efficiency will be tolerable if the number of god objects is kept low. Fortunately, this is generally the case. Typically, the user wants only one with which to prod and observe the progress of the simulation (a kind of cursor). The maximum number of such objects is limited by the bandwidth of the user's attention-- perhaps, two hands and a head camera.

These restrictions can be relaxed somewhat using the analysis in Section 8.

1. Interestingly, this notion has a theological parallel, the “watchmaker hypothesis.”

I.0 Cost Analysis of the Sparse Phase

A level comparison of the cost of sparse dynamics to that of conventional methods is somewhat difficult. Conventional algorithms incur their dominant costs *per frame*. The sparse model incurs cost *per event*. To compare these, we need a formula for the number of events occurring in an environment. Construction of this formula requires several assumptions.

I.1 Counting collisions

Statistical Mechanics [Moor81] models the number of collisions, C , among spherical particles in an environment as

$$C = 2\sqrt{2}\bar{c} \frac{\pi r^2 N^2}{V^2} \quad \text{per unit volume, per unit time}$$

N =the number of particles

r =the radius of each particle

V =the volume of the environment

\bar{c} =the average speed of the particles

To determine the total number of collisions over a period of time t , we multiply through by t and the volume V

$$C = 2\sqrt{2}\bar{c}t \frac{\pi r^2 N^2}{V}$$

We note that $\bar{c}t$ is a velocity over an interval of time, and thus a distance traveled. We wish to know the number of collisions in terms of some generic unit of distance. To make the estimate independent of scale, it is convenient to choose the distance to be r . The formula then becomes

$$C = 2\sqrt{2} \frac{\pi r^3 N^2}{V}$$

for the number of collisions occurring as all objects move a distance of their own radii.

We would prefer to express the number of collisions in terms of the density of the environment. For an environment containing only uniform spheres, the density can be expressed as

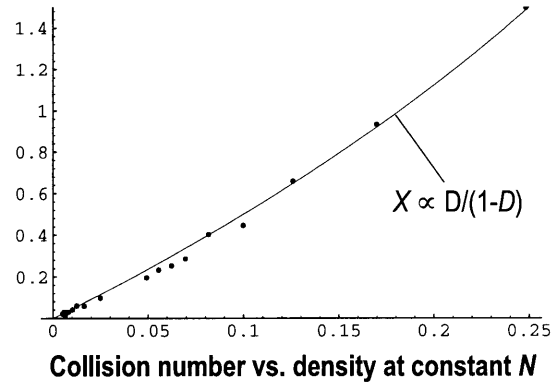
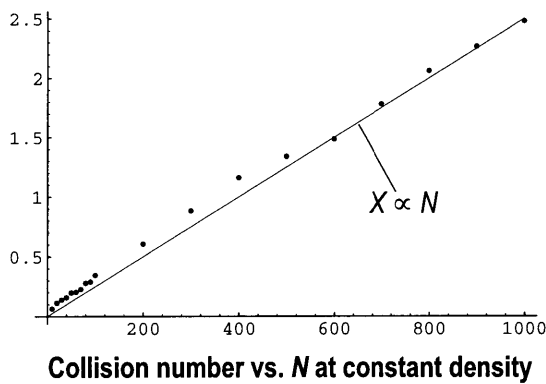
$$D = \frac{4}{3}\pi \frac{r^3 N}{V}$$

We solve this for r and substitute into the above. The V terms cancel leaving

$$C = \frac{3\sqrt{2}}{2}DN \quad (\text{J.1})$$

This relation states that the number of collisions in an environment of a given size is linearly proportional to its density and also to the number of objects present. Note that this equation does not keep the size of the objects constant. If the size of the objects remains constant, the extra factor of N in D will cause the number of collisions to increase as N^2 .

The model from which we have derived this result is used in physical chemistry to model the behavior of molecules in gases. As such, it assumes densities much less than 1. For higher densities, an additional factor of $1/(1-D)$ becomes relevant. This represents the asymptotically increasing number of interactions as space becomes completely filled. However, like the statistical mechanics of gases, sparse dynamics is intended to be used at low densities (e.g. <10%). For these values, the extra term may be safely ignored. Indeed, when the contribution of this term becomes significant, it is a good indication that the situation should be analyzed by other techniques.



To verify the collision counting formula, we tested the algorithm for various N at a constant density ($D=5\%$) and for various densities with constant a N ($N=100$). The first graph shows a direct linear correspondence. The second shows both the expected relationship to $D/(1-D)$ and also the closeness of this function to linearity over our region of interest.

I.2 Cost calculation

We may now calculate the costs associated with each algorithm.

Conventional Method. From Section 2.4 we have that conventional stepped methods require N^2 operations per frame, or if using a hierarchical space representation (as we will assume here), $N \lg N$. The cost of each of these operations is a simple sphere or bounding box comparison. We will treat as inconsequential the costs incurred when bounding volumes actually overlap.

The formula in the previous section calculated the number of meetings occurring as an element

moved the length of its radius. We would like to express the cost for the conventional method in the same terms. The question is then, how many steps are needed to simulate an element moving the length of its radius? The answer depends on the accuracy required. If elements moves a distance d in one step, then the location of any collision will be known only to within d . In other words, if we wish the collision to be accurate to within A digits of the length of r , then we must take 10^A steps. This may be improved by making the step length adaptive, in which case we set the step size to $1/10^{A_1}$ and when a collision is found, we converge on it for A_2 more digits by binary search.

So, the total cost of the conventional algorithm is

$$\text{Cost}_C = 10^{A_1} N \lg N \cdot \text{SphereComp}$$

Note that this is not an calculation of the algorithm's order, but rather a full cost calculation. We do this because the constants are relevant for the regions of N in which we are interested.

Sparse Method. The sparse method does a majority of its work at meetings. At these times two elements will need to check for their next encounter. N objects must be checked for each, so the cost of an event is $2N$. Equation J.1 above gave the number of meetings as $(3\sqrt{2}/2) DN$. There will also be additional stale events (Section 4.2) in half of which one element will need to find its next meeting as well. The cost of each object-object check will be that of the time/space calculation discussed in Appendix F.

$$\text{Cost}_S = \frac{3\sqrt{2}}{2} DN \cdot \left(2N + \text{stale} \frac{N}{2} \right) \cdot \text{PathComp} = 3\sqrt{2} (1 + \text{stale}/4) DN^2 \cdot \text{PathComp}$$

If we are using numerical convergence as in Appendix E, then PathComp in the above will be replaced by a somewhat larger value RootComp. If elements are using error envelopes, we must then converge on solutions as with the conventional method above. This will require an additional factor of A for convergence events and $(1+dr+1/3 dr^2)$ for the extra volume of the cones.

Note that a small amount of work still needs to be spent "per frame" to update object positions for the renderer. Since this computation is required only when making pictures, we consider it part of the rendering cost. In any event, the expense proves negligible when compared to either the rendering itself or the simulation cost.

I.3 Cost comparison

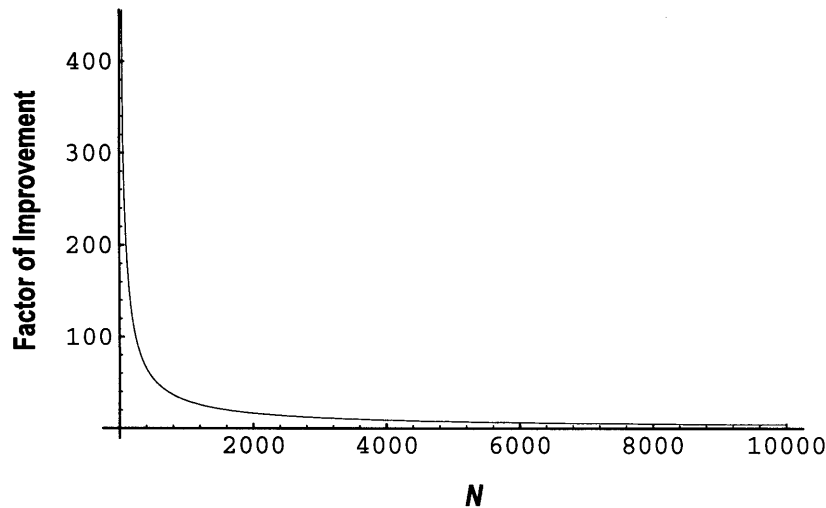
We can see that sparse dynamics is of a higher order in N than the conventional method. This indicates that its asymptotic performance will be worse. However, the constants in this case dominate for those N in which we are interested. To see this, we derive the relative of speed of sparse dynamics to that of the conventional method

$$\frac{\text{Cost}_C}{\text{Cost}_S} = \frac{10^{A_1} \lg N \cdot \text{SphereComp}}{(3\sqrt{2} (1 + \text{stale}/4)) DN \cdot \text{PathComp}}$$

The values for the various comparison operations were obtained empirically on a local machine. Although the actual values will vary according to platform, the relative sizes should be fairly representative

Comparison	Cost
SphereComp	.003 msec
PathComp	.06 msec
RootComp	3 msec

We select a sample density of 10% (rather higher than in practice) and a stale rate typical of that density, 40%. Finally, we choose $A=4$ places of accuracy for the conventional method. Plotting this for various N we get



We see that sparse dynamics is much faster for N of reasonable size. At $N=1000$, for example, the method is better by a factor of 30. In addition, we must recall that the method is returning exact, rather than approximate values, and that no collisions can be missed via aliasing as in the conventional method. It is also important to note that hierarchical methods introduced in Section 8 will reduce one of the factors of N in Cost_S to $\lg N$. We would then expect to see a much more uniform improvement even as N became extremely large.

See Section 7.0 for more comparative timings.

J.0 Lin and Canny Details and Improvements

J.1 Modifications

We have made several alterations to the basic distance finding algorithm to improve performance in our application.

J.1.1 Angle tables for features

When entering a large polygon, each edge must be checked to determine what direction to leave in. We can avoid this iterative step by preprocessing a lookup table for each polygon. The table stores the angular direction of each edge from the polygon's center.

To use the table, we project the vector connecting the two objects onto the polygon's plane. We look up the edge listed for that direction in the table. This results in a simple search problem on an ordered list. We can implement this as a binary search (at a cost of $O(\lg E)$) or a table indexed by angle for nearly constant lookup. In either case, the extra expense associated with moving across large polygons is largely eliminated.

The same method may be used when encountering a vertex surrounded by many faces.

J.1.2 Polyhedron feature tables

When objects first become engaged, they must determine their initial closest features. This can be done by selecting a random point on each object and iterating until the minimum points are found.

To make the start-up faster, we can precalculate an angle-to-feature lookup table for each object. Essentially this is a table indexed by latitude and longitude which indicates what feature is encountered when approaching the object from a given direction. A similar idea is termed the "pierce point" in [Bob89].

Each object picks its starting feature by looking up the other object's centroid in its own table. This gives a good approximation of their nearest approach and typically decreases initialization time by more than half.

This table turns out to be quite useful in other areas and may itself be worth separate study.

J.1.3 Concave extension

As described, the method is limited to convex polyhedra. If objects are concave, there may be more than one local distance minima and thus, hill climbing will be defeated. Lin and Canny suggest a hierarchical decomposition method for concave objects which will result in a larger number of convex ones. Each level is then wrapped in a convex hull which can be utilized by the basic algorithm.

Finding optimal, or even acceptable decompositions for complex objects may prove difficult,

however. Instead, we introduce a method for extending the basic algorithm to concave polyhedra directly.

We do this by introducing *imaginary edges* to the object's representation. An imaginary edge is a link between two vertices of the object which do not share a common polygon. An *imaginary face* is then defined as a polygon formed by the process of placing a new imaginary edge.

The new algorithm is allowed to traverse these edges and faces when searching for the closest feature, but is forbidden to halt on one of them. If an imaginary feature is found to be the closest, we move to the nearest real bounding feature instead. Since vertices can not be imaginary, there will always be such a feature on the current boundary. Imaginary features enable us to make all local regions of an object convex so that the hill-climbing distance algorithm can operate.

Method. During preprocessing, we construct a convex hull around each concave object using imaginary edges and faces to cover cavities. We then divide up any internal spaces this creates until all are convex regions.

During operation, while objects are unengaged, the convex hulls are used to track closest features as in the convex case. At some point, an object may penetrate one of the hull's imaginary surfaces. Any of the faces of the (convex) interior might be struck by the invading feature. We therefore split our attention and track each of them separately. We run a new minimum distance line from the invading point to each of the interior faces and keep a list of all such points for the current engagement. On each step, we use the Lin and Canny algorithm to minimize each of them. If further imaginary faces are broken, we split our attentions again. Obviously, both objects may be simultaneously tracking points in the others' interiors.

As objects become disengaged, the additional search lines can be abandoned. A search line is no longer needed when its attachment to the other object takes it outside its convex region. In this way, minimum distance search lines will be created and destroyed as the objects move. Note that as objects may be contained entirely inside the convex hulls of others, care must be taken as an object *emerges* as well.

There are a number of optimizations possible to reduce the number of points which need to be tracked. Tracking all interior surfaces of invaded regions is highly conservative. The majority of these could likely be obviated with additional analysis. Also, search lines can, in some cases, merge as well as split. In practice the number of points needing to be tracked, will usually be small, due to geometrical considerations. Most objects will likely disengage before becoming deeply tangled. In those cases where the connection is intended to be an enduring one, we can signal this beforehand and so apply other methods.

The cost of this method is likely to be similar to Lin and Canny's concave extension. We prefer this method however, as somewhat more in keeping with the original intent of the algorithm. Also, the method used here for preprocessing concave objects is somewhat more straightforward.

J.2 Cost Analysis

Lin and Canny report constant maintenance times for slow angular rotations on polyhedra with less than 200 vertices. That is, once nearest points have been established, little work is required to update them. To analyze the general case, we apply a constant angle of rotation to objects of varying complexity and count the number of features requiring traversal.

Theoretically, we reason thus: a 3D polyhedra is covered by a two dimensional surface. An arbitrary walk across this surface visits a one dimensional subset of its features. For a surface with N faces, the walk will be proportional to \sqrt{N} of the features. The distance algorithm does a simultaneous walk across the surfaces of two such objects. Its cost should therefore be $O(\sqrt{N})$. This increases quite slowly with N and so explains the algorithm's efficiency. Indeed, as Lin and Canny's results show, small simulations will be fairly insensitive to changes in the value of N .

It is of interest to note how this compares with theoretical results. The best algorithm currently known for 3D minimum distance requires $O(\lg^2 N)$ time [Dob83, Dob90]. Although this is asymptotically better than $O(\sqrt{N})$, \sqrt{N} , is in fact smaller for practical N .

It is known that finding the closest features on 3D polyhedra cannot be cheaper than $O(\lg N)$, as this is the proven lower bound for the analogous 2D problem [Chi83, Edel85]. The three dimensional case is clearly at least as difficult.

There is some hope that an algorithm may be found of this order. With a good hierarchical representation of each object, many types of searches become $O(\lg N)$. Work in this direction may be found in [Bou91].

An imaginary feature extension like the one introduced above might also be useful in this regard. We could construct a tree of such edges *inside* the object, allowing faster traversal than traveling only on the surface. Determining the optimal form of such a tree (a minimum spanning tree perhaps) would be an worthwhile direction of research.

Interestingly, $O(\lg N)$ might possibly be the lower bound for the nearest feature problem regardless of the dimensionality of the objects. All higher dimensions currently share the $O(\lg^2 N)$ bound. The problem requires us to walk only a linear path, regardless of the dimensionality of the surface we traverse.

In any event, \sqrt{N} compares reasonably even to $\lg N$ for $N < 1000$. It also represents a large improvement over commonly reported linear methods [Gilb88, Ocon93]. See Section 7.1 for empirical data.

K.0 Calculation of Zone Events

This derivation is similar to that of analytic paths in Appendix F. Likewise, we can fall back on the same numerical techniques if the analysis becomes too complex.

We construct the equations for a plane rotating around a specified axis and a particle both mov-

ing in space and rotating. The intersection of these two equations is our point of interest. We use this to test the nearest point on each object against the bounding walls and normal plane of the other objects' closest feature. The soonest intersection found along either points' trajectories will be the time of the next event. If the intersection is with a bounding wall, then we queue a new-closest-feature event. If it is with the normal plane, then we queue a collision.

Refer to [Nah93] for a related derivation.

L.0 Hextrees

L.1 Hextree operation

We enter the hyper-cones described into the cells of the hextree. It is easy to see that a hyper-cone element can impinge on more than one hextree cell. We therefore make an entry for a cone in each cell it crosses. When two elements are entered in the same cell it means that they have a potential to collide at that place, *at that time*. In order to determine if they do in fact collide, we must compare the objects' cones together as before. However, this comparison will be done far less often as we need only check against elements contained in the cells we traverse.

When adding an entry to a cell, if it contains over a certain threshold of entries, we split the cell. The current choice for splitting planes is a simple cycle through the dimensions. We divide the cell in half, though more sophisticated choices are possible to keep the tree more equally balanced.

This tree traversal is similar to that described in [Von90].

Method. Move along an object's future path in 4D, marking the cells we go through. In each cell, compare against all objects already there. The first hit found must be the first in time and therefore, we can stop as soon as the next cell along our path has a starting time later than that.

When an object collides, the path it drew into the future becomes invalid. As with the queueing system, we adopt a lazy update method: expired element entries are removed from cells only when they are encountered by something else. This gains us efficiency at the cost of some extra space in the tree.

The use of the tree reduces the algorithm cost to $O(N \lg N)$. This rises sufficiently slowly that, with refinements, we expect to be able to manage the real-time dynamics of objects numbering in the thousands.

L.2 Probability fields

As an object moves into the future, we can think of it as "painting" its presence into units of space/time ([Gill81] terms these *tixels*) in much the same way that a line drawing algorithm fills in pixels as it moves across the screen. When two objects fill the same tixel, we find a collision. When

an object's position in the future is uncertain, we may think of the object as filling in the tixels with a color somewhat lighter than black. I.e., it has some probability of being in that spot at that time. In this way, the object casts a "shadow" into the future, generating a probability field which indicates the likelihood that it will visit a particular region. The chance that two objects will have a meeting at a certain point is merely the product of their probabilities at that point. The total amount of probability energy rendered into a tixel by all objects tells us the likelihood that some event will be occurring there. By evaluating the darker regions of this future map, we can identify those area which require most of our attention.

This terminology is intensionally chosen to evoke analogies to various rendering methods, specifically radiosity [Hanr91]. We believe there is a strong correlation between the two areas. In the case of radiosity, we track the dispersion of energy between various elements of a scene. In dynamics, we track a similar dispersion of energy, differing in that elements carrying the energy *are themselves in motion*. Despite this difference, however, the cases are similar enough that we believe it possible to adapt some of the algorithms and hardware currently intended for rendering to our purposes. This is worthwhile in that rendering has received the lion's share of attention in past years, and so has a large body of sophisticated techniques and devices to support it.

The hyper-cones we use above, are simply equiprobable surface contours drawn within the field generated by each object. Ideally, we would like to enclose inside each cone a fixed proportion of the probability its object generates, based on the importance of the object and the amount of momentum it is carrying. We would have to devise a formula for trading off cone volume with the expense of checking for the object's having escaped that volume. We also may wish to place a cap on the end of each cone so as to limit the number of cells of the hextree into which we must enter it. Roughly, we should cut the cone at the point where the space/time density of its probability falls below the threshold of its importance. We then queue another event for the element to recompute at that time.

M.0 Sparse Dynamics Code

C code implementing this work is available by anonymous FTP from `media.mit.edu:sparse-dyn/`. We have tried to make it of generally utility. The reader is encouraged to obtain and extend it.*

M.1 Sparse Dynamics Collision Detection Package

This package provides fast dynamic collision detection for many body simulations. The package can handle hundreds of complex polyhedra in real-time on reasonably sized machines. There

are a number of limitations discussed below. We think that none of these is insurmountable and would welcome improvements.

To Use. The code should unpack and compile on unix platforms as is. You may need to uncomment the 'ranlib' line in the Makefile. It is written in straight C with no I/O or graphics calls. The programs EXAMPLE.C and EXAMPLE2.C are provided to demonstrate how to call the package.

The package uses the **3D** system to maintain its objects. **3D** is a general simulation package written by the Media Lab graphics group [Chen92]. I have cut out the parts of **3D** that the sparse package needs and included them in the subdirectory *3d/*. **3D** uses the Ohio State file format [Crow82] for polyhedra. Examples are included in the subdirectory *data/*.

To embed the collision detector in a general simulation package, you will need to convert your polyhedra to the **3D** format. If you translate your disk files to the Ohio State format, the **3D** routine *dyn_obj_create* can be called to read them in. The other possibility is to write a conversion routine which will fill the **3D** data structures from your own in memory. See *3d/local/3d.obj.h* for the data structures required and *3d/ph-rddetail.c* for the way in which they are used.

To operate the package, your code needs to use:

```
#include "collide.h"
```

and link with LIBSPARSE.A

Once you have loaded your objects into it, you call *step_time()* to move the simulation forward. You can then call routines such as *dyn_obj_get_current_position()* and *dyn_obj_get_current_velocity()* to follow the resulting motions of the objects. The program EXAMPLE2.C demonstrates these things.

When a collision is detected, the routine *dyn-obj.c/resolve_collision()* is called to compute the recoil forces. This routine is currently quite simple, but may be customized for more complex effects.

M.2 Shortcomings

- Distance bugs - currently about 2% of the cases do not converge. This is due various special cases which haven't yet been chased down. The code detects these cases and gives up gracefully with a best guess. Usually the best guess is close to the right distance.
- Fixed memory allocation - there is a compiled in limit on the number of various sorts of memory objects. This should be made dynamic.

- Convex objects only - The distance algorithm currently operates only on convex polyhedra. Work is underway to extend the method to handle concave shapes. In the mean time, you have to decompose your objects into convex pieces or put hulls around them.
- Simple trajectories - The current code will only provide perfect accuracy for linear motion. Object rotations and accelerations will introduce small errors.
- The hextree is not fully operational.

M.3 Code Overview

The code is written in straight C and contains no graphics or I/O calls (excepting error conditions). The code is divided into 4 modules, each with an associated .h file.

dyn-obj.c. Defines the object data structure DYN_OBJ and the routines to manipulate it. DYN-OBJ contains information used by the sparse phase of the system and pointers to the structures needed for the detailed phase.

queue.c. Defines the queue data structure EVENT, and routines to manipulate it. EVENTS are used to schedule object interactions in both phases of the system.

ph-collide.c, ph-generate.c. Defines the PH_FEATURE and PH_FEATURE_INFO data structures and routines to manipulate them. These hold connectivity info needed for the Lin and Canny algorithm. ph-generate.c does the preprocessing of each ph to fill these structures. ph-collide.c contains the routines needed to implement the Lin and Canny distance minimization algorithm. The actual polyhedral data structures used here are imported from the 3D system mentioned below.

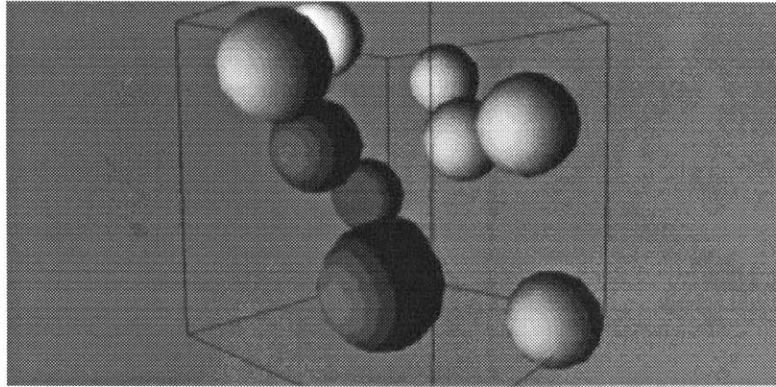
hextree.c. Defines the HEXNODE data structure and routines to manipulate it. These are used to compose the hextree, which contains predicted 4D object motions.

A subsidiary file, collidelib.c, is also present for the purpose of defining global variables and function declarations.

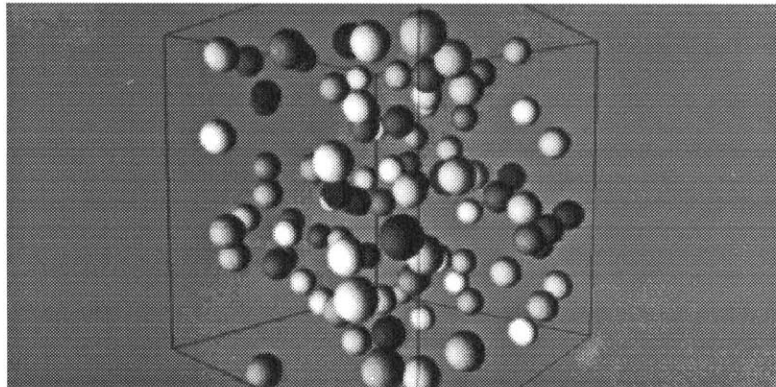
The system uses the general animation package **3D** [Chen92] for polyhedral data structures and general graphics utility routines. The value of having a solid pre-implemented base of routines from which to work can not be overstated.

N.0 Pictures of System Operation

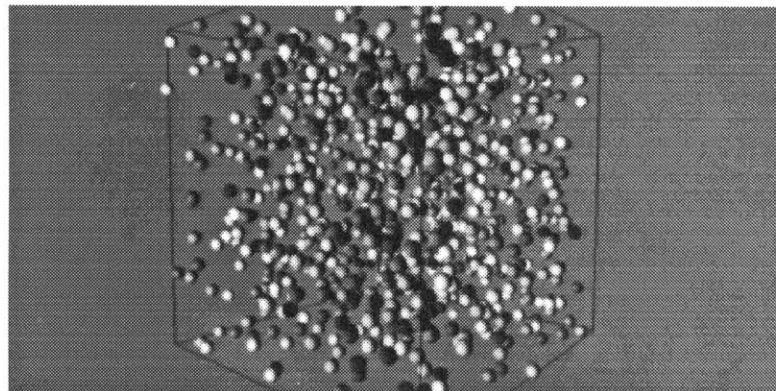
These are pictures of the sparse dynamics system in operation. Each sphere contains a polyhedra with between 24 and 600 faces. Object luminance indicates the speed of motion. Times listed are without rendering.



10 object simulation. fps > 1000



100 object simulation. fps = 85



1000 object simulation. fps = 2

Bibliography

- [Ald59] B. J. Alder and T. E. Wainwright, "Studies in Molecular Dynamics. I. General Method," *Journal of Chemical Physics*, vol 31, no 2, pp. 459-66, 1959
- [App85] Andrew Appel, "An Efficient Program for Many-Body Simulation," *SIAM Journal on Scientific and Statistical Computing*, vol 6, no 1, pp. 85-103, 1985
- [Bai93] P. Baiardi, G. Cannata, G. Casalino, and P. Pagano, "Modeling Contact Phenomena Within the Dynamic Simulation of Advanced Robotic Structures," *Proceedings IEEE International Conference on Robotics and Automation*, vol 3, pp. 196-203, 1993
- [Bara89] David Baraff, "Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies," *Computer Graphics (Proc. SIGGRAPH)*, vol 23, no 3, pp. 223-232, 1989
- [Bara90] David Baraff, "Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation," *Computer Graphics (Proc. SIGGRAPH)*, vol 24, no 4, pp. 19-28, 1990
- [Bara91] David Baraff, "Coping with Friction for Non-penetrating Rigid Body Simulation," *Computer Graphics (Proc. SIGGRAPH)*, vol 25, no 4, pp. 31-40, 1991
- [Bara92] David Baraff, "Dynamic Simulation of Non-penetrating Flexible Bodies," *Computer Graphics (Proc. SIGGRAPH)*, vol 26, no 2, pp. 303-8, 1992
- [Bat85] Klaus-Jürgen Bathe and Anil Chaudhary, "A Solution Method for Planar and Axisymmetric Contact Problems," *International Journal for Numerical Methods in Engineering*, vol 21, pp. 65-88, 1985
- [Ben75] Jon Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM*, vol 18, no 9, pp. 509-17, 1975
- [Bey84] William Beyer, *CRC Standard Mathematical Tables, 27th Edition*, CRC Press, Boca Raton, Florida, p. 12, 1984
- [Bob89] James Bobrow, "A Direct Minimization Approach for Obtaining the Distance between Convex Polyhedra," *International Journal of Robotics Research*, vol 8, no 3, pp. 65-76, 1989
- [Bon89] Susan Bonner and Robert Kelley, "Planning 3-D Collision-Free Paths," *Proceedings IEEE International Symposium on Intelligent Control*, pp. 550-555, 1989
- [Bou91] William Bouma and George Vaneczek Jr., "Collision Detection and Analysis in a Physically Based Simulation," *Proceedings Second Eurographics Workshop on Animation and Simulation*, pp. 191-203, 1991
- [Boy79] John Boyse, "Interference Detection Among Solids and Surfaces," *Communications of the ACM*, vol 22, no 1, pp. 3-9, 1979
- [Bra91] Raymond Brach, *Mechanical Impact Dynamics*, John Wiley and Sons, New York, 1991
- [Cam85] Stephen Cameron, "A Study of the Clash Detection Problem in Robotics," *Proceedings 1985 IEEE International Conference on Robotics and Automation*, pp. 488-93, 1985
- [Cam86] S. A. Cameron and R. K. Culley, "Determining the Minimum Translational Distance Between Two Convex Polyhedra," *Proceedings 1986 IEEE International Conference on Robotics and Automation*, vol 1, pp. 591-596, 1986

- [Cam90] Stephen Cameron, "Collision Detection by Four-Dimensional Intersection Testing," *IEEE Transactions on Robotics and Automation*, vol 6, no 3, pp. 291-302, 1990
- [Can86] John Canny, "Collision Detection for Moving Polyhedra," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 8 no 2, pp. 200-209, 1986
- [Cas83] Gilbert Castellan, *Physical Chemistry*, Addison-Wesley Publishing, Massachusetts, pp. 750-1, 1983
- [Chen92] David Chen and David Zeltzer, "The 3d Virtual Environment / Dynamic Simulation Language," *Computer Graphics and Animation Group Tech Report*, Media Laboratory, Massachusetts Institute of Technology, 1992
- [Chi83] Francis Chin and Cao An Wang, "Optimal Algorithms for the Intersection and the Minimum Distance Problems Between Planar Polygons," *IEEE Transactions on Computers*, vol C-32, no 12, pp. 1203-1207, 1983
- [Cook86] Peter Cook, *Nonlinear Dynamical Systems*, Prentice-Hall International, Englewood Cliffs, New Jersey, 1986
- [Crow82] F. C. Crow, "A More Flexible Image Generation Environment," *Computer Graphics (Proc. SIGGRAPH)*, vol 16, no 3, pp. 9-18, 1982
- [Cul86] R. K. Cully and K. G. Kempf, "A Collision Detection Algorithm Based on Velocity and Distance Bounds," *Proceedings 1986 IEEE International Conference on Robotics and Automation*, vol 2, pp. 1064-1069, 1986
- [Dob83] David Dobkin and David Kirkpatrick, "Fast Detection of Polyhedral Intersection," *Theoretical Computer Science*, vol 27, pp. 241-253, 1983
- [Dob85] David Dobkin and David Kirkpatrick, "A Linear Algorithm for Determining the Separation of Convex Polyhedra," *Journal of Algorithms*, vol 6, pp. 381-92, 1985
- [Dob90] David Dobkin and David Kirkpatrick, "Determining the Separation of Preprocessed Polyhedra-- A Unified Approach," *Automata, Languages and Programming*, vol 17, pp. 400-412, 1990
- [Duf92] Tom Duff, "Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry," *Computer Graphics (Proc. SIGGRAPH)*, vol 26, no 2, pp. 131-138, 1992
- [Dwor93] Paul Dworkin and David Zeltzer, "A New Model for Efficient Dynamic Simulation," *Proceedings Fourth Eurographics Workshop on Animation and Simulation*, pp. 135-147, 1993
- [Edel85] H. Edelsbrunner, "Computing the Extreme Distances between Two Convex Polygons," *Journal of Algorithms*, vol 6, pp. 213-224, 1985
- [Foi90] André Foisy, Vincent Hayward, and Stéphane Aubry, "The Use of Awareness in Collision Prediction," *Proceedings 1990 IEEE International Conference on Robotics and Automation*, vol 1, pp. 338-343, 1990
- [Gas93] Marie-Paule Gascuel, "An Implicit Formulation for Precise Contact Modeling between Flexible Solids," *Computer Graphics (Proc. SIGGRAPH)*, pp. 313-320, 1993

- [Gilb90] Elmer Gilbert and Chek-Peng Foo, "Computing the Distance Between General Convex Objects in Three-Dimensional Space," *IEEE Transactions on Robotics and Automation*, vol 6, no 1, pp. 53-61, 1990
- [Gilb88] Elmer Gilbert, D. Johnson, and S. Keerthi, "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," *IEEE Journal of Robotics and Animation*, vol 4, no 2, pp. 193-203, 1988
- [Gill81] R. Gillespie and W. A. Davis, "Tree Data Structures for Graphics and Image Processing," *Proceedings 7th Canadian Man-Computer Communications Conference*, vol 7, pp. 155-162, 1981
- [Gold60] Werner Goldsmith, *Impact: The Theory and Physical Behaviour of Colliding Solids*, Edward Arnold Ltd., London, 1960
- [Gon90] Larry Gonick, *The Cartoon History of the Universe*, Doubleday, New York, 1990
- [Gon91] Larry Gonick and Mark Wheelis, *The Cartoon Guide to Genetics*, Harper Perennial, New York, 1991
- [Gut1455] Johann Gutenberg, *Biblio latina*, Fust and Schöffer, Mainz, Germany, 1455
- [Hahn88] James Hahn, "Realistic Animation of Rigid Bodies," *Computer Graphics (Proc. SIGGRAPH)*, vol 22, no 4, pp. 299-308, 1988
- [Hanr91] Pat Hanrahan, David Salzman, and Larry Aupperle, "A rapid Hierarchical Radiosity Algorithm," *Computer Graphics (Proc. SIGGRAPH)*, vol 25, no 4, pp. 197-206, 1991
- [Hop83] John Hopcroft, J. Schwartz and M. Sharir, "Efficient Detection of Intersections among Spheres," *The International Journal of Robotic Research*, vol 2, no 4, pp. 77-80, 1983
- [Hub93] Philip M. Hubbard, "Interactive Collision Detection," *Proceedings 1993 IEEE Symposium on Research Frontiers in Virtual Reality*, San Jose, CA, 1993
- [Jos91] Leo Joskowicz and Elisha Sacks, "Computational Kinematics," *Tech Report CS-TR-300-90*, Department of Computer Science, Princeton University, 1991
- [Lin91] Ming C. Lin and John Canny, "A Fast Algorithm for Incremental Distance Calculation," *Proceedings 1991 International Conference on Robotics and Automation*, pp. 1008-1014, 1991
- [Lin92] Ming C. Lin and John Canny, "Efficient Collision Detection for Animation," *Proceedings Third Eurographics Workshop on Animation and Simulation*, 1992
- [Lin93] Ming C. Lin, Dinesh Manocha, and John Canny, "Fast Collision Detection between Geometric Models," *Tech Report TR93-004*, Department of Computer Science, University of North Carolina, 1993
- [Loz83] Tomás Lozano-Pérez, "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, vol C-32, no 2, pp. 108-20, 1983
- [Luc91] Annie Luciani, Stéphane Jimenez, Jean Loup Florens, Claude Cadoz, and Olivier Raoult, "Computational Physics: A Modeler-Simulator for Animated Physical Objects," *Eurographics Proceedings '91*, pp. 425-37, 1991

- [Ma92] Ou Ma and Meyer Nahon, "A General Method for Computing the Distance Between Two Moving Objects Using Optimization Techniques," *Advances in Design Automation (Proc. ASME Design Technical Conferences -- 18th Design Automation Conference)*, DE-vol 44-1, pp. 109-17, 1992
- [Mac60] William MacMillan, *Dynamics of Rigid Bodies*, Dover Publications, New York, 1960
- [Min86] Marvin Minsky, *Society of Mind*, Simon and Schuster, New York, 1986
- [Moor81] John Moore and Ralph Pearson, *Kinetics and Mechanism*, John Wiley & Sons, New York, pp. 86-90, 1981
- [Moor88] Matthew Moore and Jane Wilhelms, "Collision Detection and Response for Computer Animation," *Computer Graphics (Proc. SIGGRAPH)*, vol 22, no 4, pp. 289-298, 1988
- [Nah93] Meyer Nahon, "Determination of the Minimum Distance Between Moving Objects Including Velocity Information," *Advances in Design Automation (Proc. ASME Design Technical Conferences -- 19th Design Automation Conference)*, DE-vol 65-1, pp. 693-98, 1993
- [Ocon93] Ruaidhri O'Connor, John Gill, and John Williams, "A Linear Complexity Contact Detection Algorithm for Multi-Body Simulation," *Proceedings 2nd International Conference on Discrete Element Methods*, 1993
- [Pen90] Alex Pentland, "Computational Complexity Versus Simulated Environment," *Computer Graphics*, vol 24, no 2, pp. 185-192, 1990
- [Pob92] Angel del Pobil, Miguel Serna, and Juan Llovet, "A New Representation for Collision Avoidance and Detection," *Proceedings IEEE International Conference on Robotics and Automation*, vol 1, pp. 246-51, 1992
- [Pre88] William Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes in C*, Cambridge University Press, New York, 1988
- [Sch91] Achim Schweikard, "Polynomial Time Collision Detection for Manipulator Paths Specified by Joint Motions," *IEEE Transactions on Robotics and Automation*, vol 7, no 6, pp. 865-870, 1991
- [Scla91] Stan Sclaroff and Alex Pentland, "Generalized Implicit Functions for Computer Graphics," *Computer Graphics (Proc. SIGGRAPH)*, vol 25, no 4, pp. 247-250, 1991
- [Sed83] Robert Sedgewick, *Algorithms*, Addison Wesley Publishing Co., Reading, Massachusetts, 1983
- [Sny92] John Snyder, "Interval Analysis for Computer Graphics," *Computer Graphics*, vol 26 no 2, pp. 121-30, 1992
- [Sny93] John Snyder, A. Woodbury, K. Fleischer, B. Currin, and A. Barr, "Interval Methods for Multi-Point Collisions between Time-Dependent Curved Surfaces," *Computer Graphics (Proc. SIGGRAPH)*, pp. 321-334, 1993
- [Str79] William Strunk Jr. and E. B. White, *The Elements of Style*, MacMillan Publishing Co., New York, 1979
- [Thi87] William Thibault and Bruce Naylor, "Set Operations on Polyhedra Using Binary Space Partitioning Trees," *Computer Graphics (Proc. SIGGRAPH)*, vol 21, no 4, pp. 153-62, 1987

- [Tuf83] Edward Tufte, *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, Connecticut, 1983
- [Tuf90] Edward Tufte, *Envisioning Information*, Graphics Press, Cheshire, Connecticut, 1990
- [Tur90] Greg Turk, "Interactive Collision Detection for Molecular Graphics," *Tech Report TR90-014*, Department of Computer Science, University of North Carolina at Chapel Hill, 1990
- [Van91] George Vanecek Jr., "Brep-Index: A Multidimensional Space Partitioning Tree," *First ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAD Applications*, pp. 35-44, 1991
- [Von90] Brian Von Herzen, Alan Barr, and Harold Zaltz, "Geometric Collisions for Time-Dependent Parametric Surfaces," *Computer Graphics (Proc. SIGGRAPH)*, vol 24, no 4, pp. 39-48, 1990
- [Wilh87] Jane Wilhelms, "Dynamics for Everyone," *Tech Report UCSC-CRL-87-6*, Department of Computer Science, University of California, Santa Cruz, pp. 49-71, 1987
- [Will87] John Williams, "A Method for Three Dimensional Analysis of Numerous Deformable Discrete Bodies Including Automatic Fracturing," *International Conference on Computational Plasticity*, 1987
- [Will88] John Williams, "Contact Analysis of Large Numbers of Interacting Bodies Using Modal Methods for Microscopic Failure Analysis," *International Journal of Computer Aided Methods in Engineering -- Engineering Computations*, vol 5, no 3, 1988
- [Will92] John Williams and Alex Pentland, "Superquadrics and Modal Dynamics for Discrete Elements in Interactive Design," *International Journal Computer Aided Engineering Computation*, vol 9, no 2, 1992
- [Yang] Zijiang Yang and Brian Barsky, "Realtime Dynamic Motion Animation with Collision Detection, Response, and Frame Scheduling," Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, In preparation
- [Zeg91] S. Zegloul, P. Rambeaud, and J. P. Lallemand, "On Fast Computation of Minimum Distance Between Convex Polyhedrons: Application to Collision Detection in a Robotic Simulation System," *Proceedings ASME Computers in Engineering Conference*, vol 2, pp. 425-32, 1991
- [Zel89] David Zeltzer, Steve Pieper, and David Sturman, "An Integrated Graphical Simulation Platform," *Proceedings Graphics Interface '89*, pp. 266-274, 1989