Graceful Degradation of Collision Handling in Physically Based Animation

John Dingliana Carol O'Sullivan

Image Synthesis Group, Computer Science Department,
Trinity College Dublin, Ireland.
http://isg.cs.tcd.ie
John.Dingliana | Carol.OSullivan @ cs.tcd.ie

Abstract

Interactive simulation is made possible in many applications by simplifying or culling the finer details that would make real-time performance impossible. This paper examines detail simplification in the specific problem of collision handling for rigid body animation. We present an automated method for calculating consistent collision response at different levels of detail. The mechanism works closely with a system which uses a pre-computed hierarchical volume model for collision detection.

1. Introduction

Despite the levels of computational power available today, even to the modestly funded home user, interactive physically based animation remains a challenge. The source of the problem lies in the fact that the physical world we are attempting to model is infinitely complex. Thus, no matter how precise our model, it will only be an approximation of the real world. This is not always disastrous as many simplifications in animation go unnoticed by the human viewer. This uncertainty could in fact be exploited to simplify less obvious parts of the scene in order to reduce computational complexity[3,7,17]. Time critical approaches trade accuracy for speed, simplifying where necessary, to meet the demands of real-time rates throughout the simulation.

Collision handling is a particular area of physically based animation where there is great potential to save on computation time by optimising the speed-accuracy trade-off. Collision detection has long been a major bottleneck in physically based animation and it would be highly desirable to have a system capable of managing this difficult problem at different levels of detail. One such system uses the concept of an interruptible collision detection mechanism which "adapts its workload to fit a time budget"[10]. However, such a system is not complete unless the data returned by it can immediately be of use to a collision response mechanism, which evolves the state of colliding objects based on how they have collided.

This paper describes a comprehensive approach to level of detail management in the collision-handling problem. We present a method in which the approximate data returned by a progressively refinable collision detection mechanism is interpreted and used by its sister process, the collision response mechanism. Section 2 presents the background to this work. In section 3, we describe a mechanism which performs collision detection based on hierarchical sphere tree representations of objects. We focus on the selective interpretation of data by the collision response process in section 4. Section 5 discusses the details of our implementation and we conclude in section 6 with a discussion and plans for future work.

2. Background

The problem of simplifying the real world model before it is stored and manipulated on a machine is not a new one, nor is it limited to the domain of physically based animation. However, it does have particular relevance in real-time applications, where the demand for consistently high frame rates frequently calls for further simplifications of the model.

Simplifying the entire scene might be one way of ensuring real-time rates, but it is not always possible to predict, in advance, how the simulation complexity will evolve with time. Such an approach faces the risk of simplifying the entire simulation for the sake of a few rare snapshots of high complexity. What is required is a mechanism capable of dynamically managing the detail level of a scene, trading accuracy with speed, so that optimal use is always made of the time available for each frame. Thus can it be guaranteed that the demands of real-time animation will be met throughout the lifetime of the simulation without excessively sacrificing the detail levels.

Many physically based animation systems achieve real-time rates by culling computations for parts of the scene based on visibility[5]. However we must take care, in simulation, not to neglect even the non-visible parts of the scene, for as the simulation evolves or the view changes, dependencies often arise between the objects in the visible and non-visible parts of the scene. Furthermore, culling is not an option in cases such as when the visible part of the scene alone is still too computationally complex. Simplification within the visible area itself is required, and a scheduling mechanism, which employs perceptual heuristics, may be used to determine which objects or events in the visible scene deserve more processing time[4,15,17]. Once the scene has been partitioned into areas of varying priority, there are various ways in which we can trade accuracy for speed. One approach is to use simulation models, of varying levels of complexity, to evolve different parts of the scene[4,6]. Even more desirable would be to have a single consistent mechanism capable of updating the scene at different levels of behavioural detail. In this paper, we discuss such a mechanism for dealing with the problem of collision handling.

The process of collision handling, in most current approaches, can be broken down into several distinct phases. A broad phase collision test rapidly eliminates objects which are clearly not colliding. This is usually achieved using coarse bounding volume intersection tests[9]. The narrow phase of collision detection handles possible candidates that the broad phase is unable to eliminate. In most cases the narrow phase consists of polygonal intersection tests which are as accurate as the surface representation of the object will allow. Some approaches have an intermediate phase which employs progressive refinement of data to narrow down the areas of possible collision[8,16]. Once it has been conclusively determined that a collision has taken place, a contact modelling phase extracts the relevant details of the collision. This is required by the final mechanism responsible for updating the states of the colliding objects, based on the laws of dynamics.

To ensure a completely time-critical system, we require that all processes of indeterminate complexity be packaged into the interruptible part of the system. Any calculations which have to be performed after that must take a constant or predictable time to complete. Alternatively, we can have several different mechanisms handling separate parts of the problem, each packaged in their own interruptible processes. Our time-critical approach to the collision-handling problem uses interruption in the progressively refinable narrow phase to obtain approximate contact data. This must be interpreted by the contact modelling process before the collision can be resolved.

Up to now, contact modelling has been problematic, due to the inexact nature of even the most accurate of techniques available[12]. In interruptible systems, the problem is further exacerbated due to the reduced accuracy of results. A system which reduces computational complexity is ineffective if the resulting response is unbelievable. In an approach based on graceful degradation our first requirement is that the system delivers acceptably consistent results even with reduced input data from collision detection. Using perceptually based sorting it is possible to strategically simplify the scene and work towards the goal of plausible simulation[15]. Thus, we exploit uncertainty to deliver real time animation.

3. Collision Detection

We use a collision detection mechanism, based on the approach described by Hubbard, which checks for intersections between the nodes of a sphere tree data structure[9,10]. The key feature we wish to inherit from this approach is the hierarchical layout of the volume representation, which provides us with the basis for graceful degradation. We exploit the nature of the sphere-node, which provides us with a quick means of obtaining useful data for the collision response mechanism described later. Other useful advantages of this approach include an inbuilt method of handling collisions between concave objects, which other methods approach by using unions of convex objects; the sphere tree by its very nature is already a union of spheres.

For collision detection, we follow Hubbard's general approach, and extend it with our own mechanisms for scheduling, contact modelling and collision response. These issues constitute the main focus of this paper, but

it is impossible to discuss full collision handling without first explaining the features of the collision detection mechanism being used. This section gives such an outline, and details the features that are most relevant to our implementation of the full collision handler.

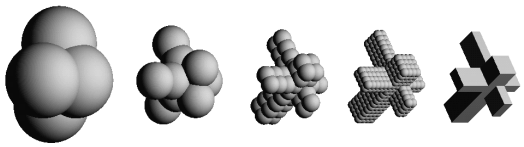


Figure 1: *Sample Sphere Tree Hierarchy*

Each object in the world is associated with its own sphere-tree, which is a spatial occupancy representation of the object's volume based on sphere primitives (see Figure 1). The sphere primitives are useful because collision detection between nodes effectively amounts to a distance test between the centres of the relevant spheres.

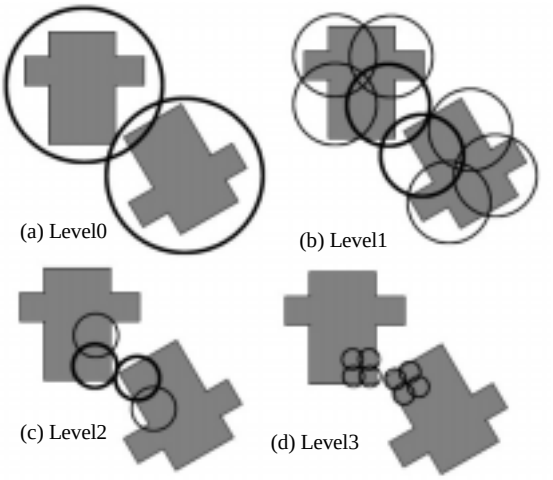


Figure 2 *Collision Detection at different levels of detail*

The difficult part of such an approach, as with all hierarchical representations, is updating the hierarchy to reflect its change of state. We must update the position of each node to reflect the object's orientation before any intersection tests can be done. This can become computationally taxing as we go deeper down the hierarchy, but the workload is greatly reduced by updating only the relevant sub-trees i.e. the nodes for which the parent has been found to be intersecting.

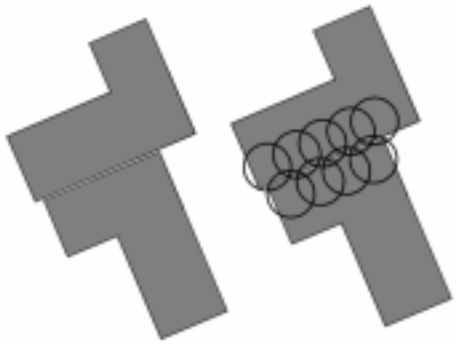
The collision detection mechanism is outlined in Figure 2. The process involves performing intersection tests for increasingly finer levels of sphere-tree detail. A **possible collision** is flagged when an intersection is found between sphere nodes on two different objects. Nodes requiring further processing are marked in bold in Figure 2 (a), (b) and (c). If the intersecting nodes are not leaves on the sphere tree, then all of their children must in turn be checked for intersection. Only when we reach the leaf nodes can we return a conclusive result as to whether the objects are colliding (in which case, we have an instance of a **true collision)** or if, in fact, the objects are not colliding at all (Figure 2(d)). Each true collision must definitely be resolved by the response mechanism. A time critical approach to sphere-tree collision detection consists of a scheduling mechanism, which interrupts the detection process when it exhausts the time allocated to it (see section 5.1.). When the process is interrupted at some stage in the traversal of the sphere tree, all possible collisions must then be processed by the resolution mechanism as if they were true collisions.

One approach to collision detection uses the progressive traversal of the sphere tree as a middle phase and follows it up with a more detailed narrow phase consisting of polygonal intersection tests[16]. Sphere tree traversal in such an approach is used to localize the region to be checked later for polygon intersection. This not only requires more pre-processing time and data storage in determining the polygonal surface regions encompassed by each sphere, but increases the time spent by collision detection in the more intensive polygon intersection test. It also adds to the burden of collision response in having to deal with several special cases of intersections, depending on whether the intersection involves combinations of faces, edges or vertices.

Our coarser intersection test will only yield a generic type of interference, that between two spheres. As we traverse deeper down our sphere tree hierarchy, this approximation becomes increasingly more accurate. The radii of the sphere-tree nodes become smaller and their volumes more closely approximate the true surface. Progressive refinement is introduced into our system at the sphere-tree traversal stage. Thus, it constitutes the main part of the narrow phase in our system. We compensate for the lack of polygonal detail by having deeper trees in our representation of the object than would be required of applications that depended wholly on narrow phase polygon testing.

It must be remembered that our detection process is interrupted at some stage during the traversal of the sphere tree. To do further polygonal intersection tests after the collision detection has exceeded the time allocated to it, defeats the purpose of our entire approach, as we cannot predict how long these will take. Instead, polygon tests are reserved as the finest level of detail after the leaf nodes in the tree have been found to be intersecting. We must allow for the possibility that

collision detection will be interrupted before our system has had the opportunity to do accurate polygon testing and that we will be required to compute response based on whatever data that we have gathered in the coarser intersection tests of the intermediate phase. Many others have dealt with polygon level collision detection[12]. Our principle concern in this paper will be the collision handling at coarser levels of detail.



(a) This might be considered a single collision between two faces

(b) An increasing number of sphere collisions will be detected at each level of detail

Figure 3: *Multiple Sphere Intersections*

One side effect of having only the generic sphere-node collision is that edge and face collisions are detected as multiple sphere collisions. One special case in a polygon-based approach might be detected as multiple instances of sphere intersections (see Figure 3). This increases the number of cases that collision response has to deal with and might seem to increase the workload of collision handling. However, we are simply trading multiple instances of a much simpler problem for fewer instances of a more difficult one. This is a necessary tradeoff in our chosen approach as it allows us to support the interruptible sphere-tree collision detection which we use in our implementation.

4. Collision Response

The primary goal of collision response is to deal with objects which the detection mechanism has identified as colliding[14]. The response may be something as simple as changing the colours of the colliding objects, destroying one or all of them, or moving them apart so they are no longer colliding after the collision event. More is required in physically based animation, as the collision response mechanism has to calculate a change of state for the colliding objects based on laws of dynamics.

Solving the collision response problem can become a computationally intensive task if we take into account all the variables, such as elasticity, friction and energy, which contribute to the behaviour of colliding objects in a scene. There are countless mathematical laws in dynamics, which might be applicable to the problem of collision resolution. However, there is no general solution encompassing all these details that would be of use to real-time animation. Once again we must give up accuracy for speed, and make some simplifications to the model in order to provide a universally applicable solution for interactive animation.

As explained in the previous section, our collision response mechanism has the added burden of dealing with approximated data, which must be selectively processed before it can be passed over to the part of the system that is responsible for the dynamics. In this paper we are chiefly concerned with the problem of using the refinable approximations returned by our collision detection system to obtain useful results for a real-time system. We will focus on a simple dynamics solver which deals with the problem of collision handling for rigid bodies[1]. We model all of our colliding entities as perfectly rigid. No kinetic energy is lost during collisions, and change of state is calculated based on instantaneous impulses[2,11,13]. We concentrate on the problem of contact forces, acting in directions normal to the surfaces of colliding objects, and do not handle the problem of friction forces during collisions. Mirtich showed how an impulse model can be applied to cases of resting and sliding contact as well as colliding contact[11].

4.1. The Dynamics Model

Working with an impulse model allows us to deal with the instantaneous values of the colliding objects' state variables. As we are dealing with approximations which change in their degree of accuracy from frame to frame it is desirable that we are able to deal with the instantaneous values of the relevant parameters.

For rigid body collisions, a scalar j representing the magnitude of the impulse along the collision normal, $\hat{\mathbf{n}}$, is calculated using the following equation[2]:

$$j = \frac{-(1+\epsilon)\,\mathbf{v}_C}{\frac{1}{m_a}+\frac{1}{m_b}+\hat{\mathbf{n}}\bullet\left(\mathbf{I}_a^{-1}(\mathbf{r}_a\times\hat{\mathbf{n}})\right)\times\mathbf{r}_a+\hat{\mathbf{n}}\bullet\left(\mathbf{I}_b^{-1}(\mathbf{r}_b\times\hat{\mathbf{n}})\right)\times\mathbf{r}_b} \quad (1)$$

The variables are

ϵ : the coefficient of restitution which will be 1 for perfectly elastic collisions

m_i: mass of body *i*

$\mathbf{I}_i$: the inertial tensor matrix of body *i*

$\mathbf{r}_i$: the displacement vector representing the

displacement of the collision point **p** from the respective centres of mass $\mathbf{x}_i$ of the colliding objects as follows:

$$\mathbf{r}_i = \mathbf{p} - \mathbf{x}_i \tag{2}$$

$\mathbf{v}_C$: the collision velocity which represents the relative velocities of the points of collision along the collision normal. This is calculated from the linear velocity $\mathbf{v}_i$, rotational velocity $\boldsymbol{\omega}_i$, and collision point displacement $\mathbf{r}_i$ of the colliding objects *a* and *b* as follows:

$$\mathbf{v}_C = \hat{\mathbf{n}} \bullet \left(\left(\mathbf{v}_a + (\boldsymbol{\omega}_a \times \mathbf{r}_a) \right) - \left(\mathbf{v}_b + (\boldsymbol{\omega}_b \times \mathbf{r}_b) \right) \right) \tag{3}$$

4.2. Contact Modelling based on Approximate Data

We discussed in Section 3 how the narrow phase of our collision detection system selectively traverses the sphere tree to localize the region of interference between two objects. In the previous section, we identified the variables that we would require even in a minimal impulse based solution. Apart from the state variables of the individual objects, which should be directly available through their associated data structures, we need to know for each collision, a collision point and a vector representing the direction in which the required impulse is to be applied. Further processing is thus required before we have data of a form with which the dynamics solver is able to deal.

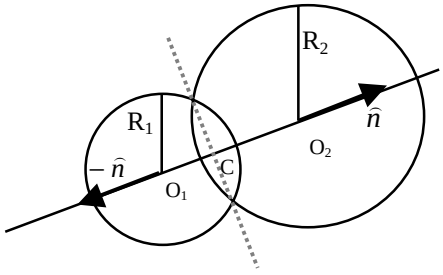


Figure 4: *Determining collision point and normal*

Exploiting once again the simple nature of our sphere-tree nodes, a quick approximation of the collision vector is the common normal to the two spheres which have been flagged as colliding. We can obtain this from the line that runs across their two centres.

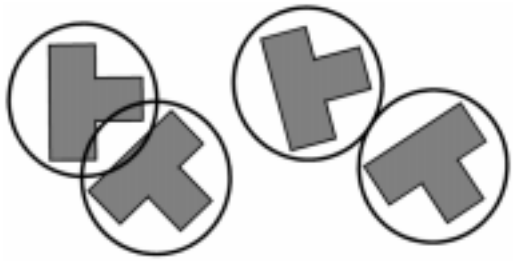
$$\hat{\mathbf{n}} = \frac{\mathbf{o}_2 - \mathbf{o}_1}{\left| \mathbf{o}_2 - \mathbf{o}_1 \right|} \tag{4}$$

An approximation of a collision point is where this line crosses the plane of intersection between the two spheres (see Figure 4). This plane subdivides the line segment between the two centres into lengths, proportional to their respective radii, and the point of intersection is determined by a simple proportionality calculation.

$$\frac{\left| \mathbf{c} - \mathbf{o}_1 \right|}{\left| \mathbf{c} - \mathbf{o}_2 \right|} = \frac{R_1}{R_2} \tag{5}$$

This result can easily be calculated, even in the case where we are resolving intersections between spheres of different levels in the respective hierarchies of the two objects.

Now that we have decided on a collision point and a vector along which to apply the collision impulse, we can apply the equations of section 4.1 to generate the appropriate response. It should be noted that so far we are only dealing with the objects at a stage after they have interpenetrated. Traditional collision detection systems would require, at this stage, that we backtrack in simulation time to determine the instant in time when the actual collision took place. This is required to obtain data for calculating an accurate response, as well as to enforce the non-interpenetration constraint. In the coarse phase, we will always be dealing with approximations of object volume. Thus, when an intersection occurs between sphere nodes of an object, we cannot imply that the actual objects themselves are even touching (see Figure 5). It would be highly infeasible to perform backtracking at every level of detail, as we can be certain that a case of interpenetration at one coarse level will frequently resolve to non-interpenetration at a finer resolution.



(a) intersection at time t_0 (b) after backtracking to "collision time"

Figure 5: *Inappropriate backtracking*

At present we leave backtracking, as we do polygonal intersection tests, as a possible final level of detail to be applied to the highest-priority collisions only.

4.3. Processing Collision Data

The final phase in collision handling is to call the dynamics solver mechanism which applies and resolves all the impulses to generate a mathematically consistent change of state in all colliding objects in the scene. At this stage, the contact modelling mechanism has provided us with all the variables we require, so computing and applying the required impulses for each instance of a collision is a fairly straightforward process of applying the variables to the equations of section 4.1.

All that is required now is to go through the dynamically generated collision table and to retrieve the relevant data. Several options present themselves as to how collision processing is to be organized. One option is to store collision data in a hierarchical tree structure similar to, but separate from, the way we have organized our volume data. A time-critical response resolution mechanism does a prioritised traversal of the collision tree applying impulses at increasing levels of detail. In such an approach, we have the option of performing the contact modelling within the schedule of either the response or the detection mechanism. This approach conforms to the requirements we laid out for a fully time-critical system with all processes of indeterminate complexity packaged in their own interruptible mechanisms.

However, it is often the case that data used in the collision detection phase is immediately of use to the response calculations. The two processes have many common requirements, such as the distance calculations and the data they return. Furthermore, in a system that schedules collision handling based on the visibility of the different parts of the scene, it is likely that we will wish to follow the same prioritisation scheme for both processes of collision handling. It would therefore be advantageous to have a single interleaved collision handling mechanism, with both the processes of detection and response working together and sharing data within the same schedule.

This is the approach that we have favoured in our implementation, which we discuss in section 5. In such a system, there is no need to store the various different levels of contact detail. We are concerned only with the level of detail that is currently being processed. Data gathering continues to run with the interruptible detection part of the mechanism. When this is interrupted, the impulse calculations are immediately applied with collision data at the highest available level of detail.

5. Implementation

In this section we describe an implementation of the ideas presented in earlier sections and illustrate some of the results we have obtained.

5.1. The Application

A summary of the system design is shown in Figure 6. A world class encapsulates all the simulation concerns of the application, including collision handling and state propagation when there are no objects colliding. User interaction and all other interfaces are maintained by the application class, which contains the world.

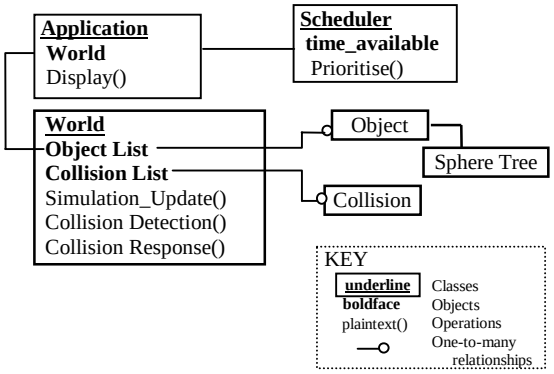


Figure 6: *Main control and data structures*

The key data structures in the application are the object class, which stores the state of each body in our system; the sphere tree class; and a dynamically generated collision list, through which the collision detection and response mechanisms communicate data.

The **Object** class needs to contain at least the state variables and constants shown in Figure 7.

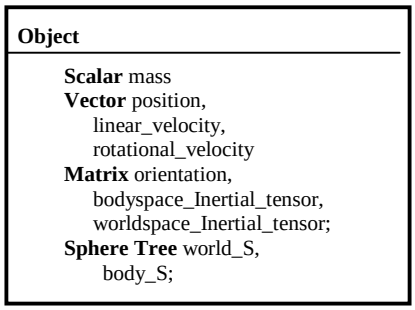


Figure 7: *The Generic Object Class*

A sphere-tree associated with each object is laid out in a hierarchical manner as already discussed in Section 2. Two instances of a sphere tree are stored: the first (body_S) holds the untransformed body-space coordinates of the sphere nodes and is constant throughout the simulation, while the second (world_S) represents the object in world coordinates after it has been updated with respect to the orientation and position

of the body. Only the relevant nodes, i.e. the ones being processed by collision detection, are updated.

The dynamically generated **collision list** stores collision data in the form of **collision entries**. Each collision entry models the contact between a pair of objects in the scene and stores more detailed data in the form of a list of **sphere collision** data structures (Figure 8). The sphere collisions represent the specific intersections between nodes on the pair of colliding objects. This is the data that is required by the collision response mechanism.

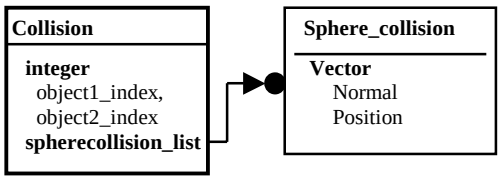


Figure 8: *The collision data structures*

The world class carries out collision detection after every update of the simulation scene, generating the collision list of intersections at various levels of detail. Potential candidates for collision, in other words those objects not eliminated by the broad phase, are placed in a list of collisions for further processing. Starting at the root nodes we traverse the sphere trees of the objects checking for intersection. When an intersection is found, the contact variables are calculated and an entry is stored in the **sphere collision list** of the respective collision object. Whenever we finish processing a complete level of a sphere tree, we have the required data to refine the resulting response by one level of accuracy.

After solving the equations of section 4.1 to determine the impulse that needs to be applied at each intersecting node, a final change of state for each object is determined by resolving multiple impulses to determine their net effect on the object. Two things are worth noting here:

(i) the process of calculating a single impulse always involves a constant amount of processing time.

(ii) the time spent on collision response is directly dependent on the number of intersections being processed.

The number of intersecting nodes is likely to be different for each collision and for each level of detail so processing time for impulse resolution varies for different collisions. However it is at the contact-modelling phase that we decide on the number of nodes that will be processed so, given the amount of processing time for a single impulse computation, it is possible to forecast the amount of time that will be spent on the process of calculating collision response. A scheduling mechanism can then make use of this information to fulfil the requirements of a time critical system.

The time-critical scheduler for the full application monitors the time spent in generating each frame of the animation. Taking into account the time required for other processes, such as rendering, it needs to decide how much time will be made available for the combined processes of collision detection, contact modelling and collision response. The scheduler initially allocates a time quota at each frame for all collision handling processes and decides on the order of collision processing. At each stage in the collision handling/contact modelling phase it not only checks if the quota has been exceeded but also reserves the time that will later be spent on the response computations. When it has determined that the detection process has exhausted its allocated time quota, it interrupts the collision detection process and passes control to the collision response mechanism. The response mechanism in turn updates the scene based on the most accurate level of detail available (in other words the most recently completed sphere collision lists for each collision).

5.2. Results

Figures 9(a) – (c) show sequences of frames from an animation of a single collision interrupted at three levels of sphere tree detail. We look at the two dimensional projection of a very simple collision case for ease of visualization. The first two frames in each strip show, respectively, the starting positions of the two objects and the instant in which an intersection between sphere nodes is detected. After the collision has been processed the three different animation levels of detail show increasing accuracy as can be seen from the trajectories and the rotational velocities of the two objects.

This approach allows us to manage collision handling in massively populated scenes. The only requirement is that we have time to do collision detection and response at the coarsest level of detail for all objects in the scene.

6. Conclusion and Comments

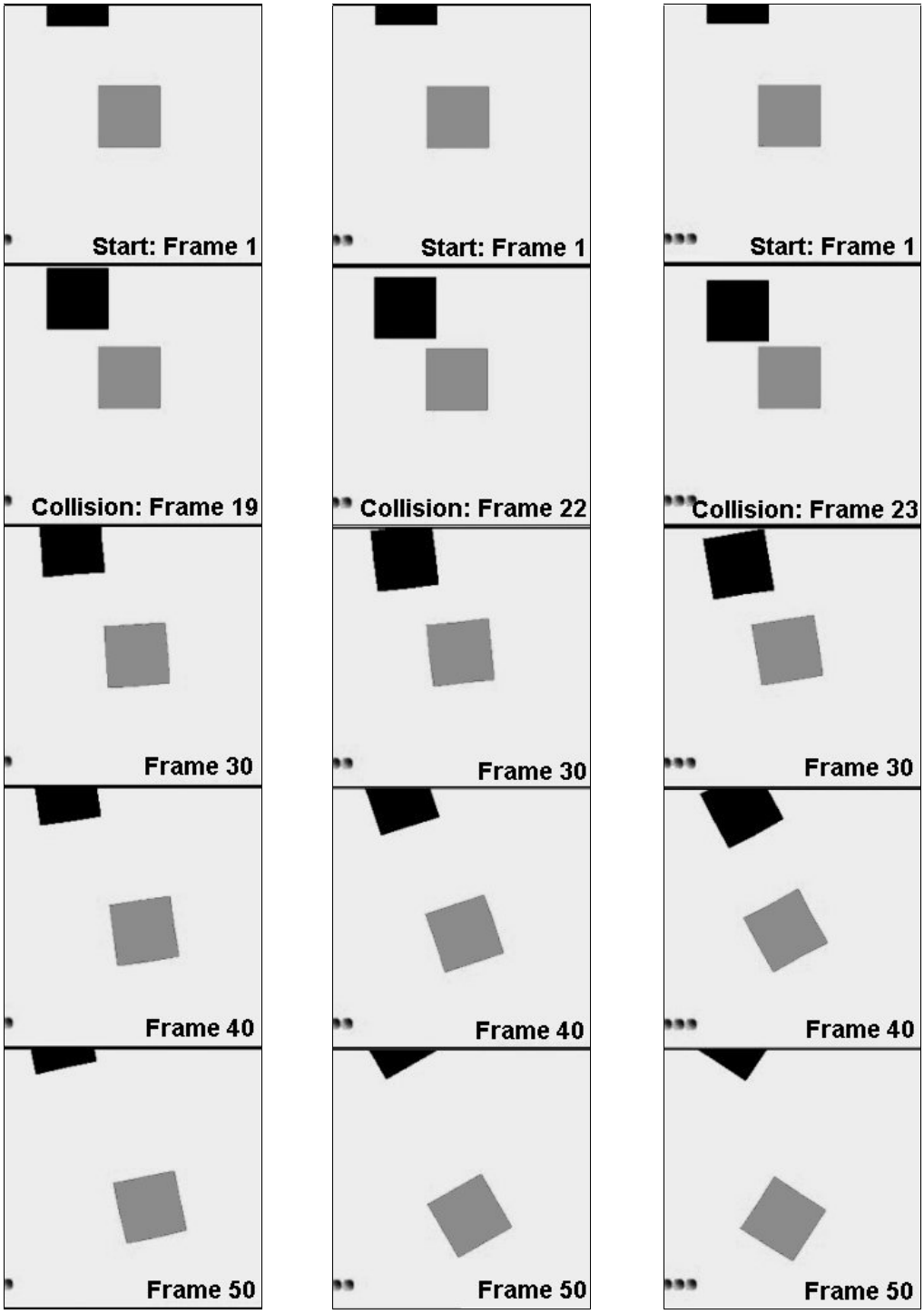
In this paper we introduced an approach to what we call level of detail collision handling, and described an application which implemented this idea. Previous approaches have dealt with refinable detection but have failed to describe how imprecise data returned by a coarse level process might be used in anything but the simplest response. We go one step further in this particular direction and propose a method in which we actually use the approximate data returned by an interruptible collision detection to achieve useful results

for interactive animation. By controlling the level of detail of individual interactions in a complex scene, it becomes possible to reduce computational workload and meet the demand for real-time frame rates. The harmful impacts of simplification on the believability of the animation can be reduced by careful management of levels of detail.

Evaluating the effectiveness of our new collision response methods is a difficult task. Hubbard[9,10] showed that, using interruption, almost constant-time collision detection was possible. In previous work[15], a metric was presented for evaluating the geometric and perceived collision inaccuracy present in a frame of an animation where interruptible collision detection was used. This metric was based on a set of psychophysical experiments which determined subjects' sensitivity to inappropriate collisions, such as objects repulsing each other at a distance. It was shown that using a scheduling mechanism based on perceptual heuristics, perceived inaccuracy, as measured by this metric, was significantly reduced. However, the effect of unrealistic response on this reaction was not considered. A fully comprehensive evaluation of our techniques would also involve such experiments, which deserve a study in their own right, and will reveal insights into how the full process of collision handling should be prioritised by the scheduler.

References

1. Baraff D. – Non-penetrating Rigid Body Simulation. Eurographics '93 State of the Art Reports. 1993
2. Baraff D. and Witkin A. - Physically Based Modelling. Siggraph '98 Course Notes (1998)
3. Barzel R., Hughes J.F. and Wood D.N. -Plausible Simulation for Computer Graphics. Animation and Simulation '96. pp. 183-197. (1996)
4. Carlson D.A. and Hodgins J.K. – Simulation Levels of Detail for Real-Time Animation. Proc. of Graphics Interface '97. pp. 1-8. (1997)
5. Chenny S. – Culling Dynamical Systems in Virtual Environments. Proc. 1997 Symposium on Interactive Graphics (1997)
6. Chenny S. and Forsyth D. – View Dependant Culling of Dynamic Systems in Virtual Environments. Proc. 1997 Symposium on Interactive 3D Graphics. (1997)
7. Funkhouser T.A. and Sequin C.H. - Adaptive Display Algorithm for Interactive Frame Rates During Visualization of complex Environments. SIGGRAPH '93. pp. 247-254. (1993)
8. Gottschalk S, Lin M.C. and Manocha D – OBBTree: A Hierarchical Structure for Rapid Interference Detection. SIGGRAPH '96. (1996)
9. Hubbard P.M. - Collision Detection for Interactive Graphics Applications. IEEE Transactions on Visualization and Computer Graphics. 1(3) pp. 218-230. (1995)
10. Hubbard P.M. – Approximating Polyhedra with Spheres for Time-Critical Collision Detection. ACM Transactions on Graphics, 15(3) pp. 179-210 (1996)
11. Mirtich B. – Impulse Based Dynamic Simulation of Rigid Body Systems. PhD Thesis, University of California, Berkeley, 1996.
12. Mirtich B. – V-CLIP: Fast and Robust Polyhedral Collision Detection. ACM Transactions on Computer Graphics 1998.
13. Mirtich B., and Canny J. – Impulse Based Dynamic Simulation. Proc. 1995 symposium on Interactive 3D Graphics. 181-188 (1995)
14. Moore M. and Wilhelms J. – Collision Detection and Response for Computer Animation. Computer Graphics Vol 22(4). pp. 289 – 297. (1988)
15. O'Sullivan C.A., Radach R. and Collins S. – A Model of Collision Perception for Real-Time Animation. Computer Animation and Simulation '99. pp. 67-76 (1999)
16. Palmer I.J. Grimsdale R.L. - Collision Detection for Animation using Sphere-Trees. Computer Graphics Forum, 14(2) pp. 105-116. (1995)
17. Reddy M. - Perceptually Modulated Level of Detail for Virtual Environments PhD Thesis University of Edinburgh. (1997)



(a) Level 1 Collisions (b) Level 2 Collisions (c) Level 3 Collisions

Figure 9: *Animation strips*: Shown above are selected frames from collision simulations interrupted at three different levels of sphere-tree detail. Note the varying gap at "collision time" and the differences in the calculated final linear and rotational velocities.