Collision Handling for Virtual Environments

Carol O'Sullivan, John Dingliana, Fabio Ganovelli, Gareth Bradshaw

Image Synthesis Group, Trinity College Dublin
{Carol.OSullivan, John.Dingliana, Fabio.Ganovelli, Gareth.Bradshaw}@cs.tcd.ie

Abstract

An efficient and realistic collision handling mechanism is fundamental to any physically plausible Virtual Environment. In this tutorial, we will first examine the applications of collision handling, and introduce the open problems in this area. We will then provide a detailed introduction to the many different approaches, past and current, to the problems of collision detection and contact modelling. The construction and evaluation of the wide range of bounding volume hierarchies used for collision detection will be discussed, as will the particular problems associated with deformable object animation. The next issue to be addressed is the problem of collision response, and finally we will discuss perceptual issues relating to this topic.

Tutorial Outline

i. Introduction. Collision Handling: Applications and Challenges;

ii. Collision Detection for rigid bodies. Hierarchical techniques and progressive refinement; Interruptible collision handling.

iii. Bounding volumes: construction and evaluation;

iv. Deformable object animation.

v. Collision Response: Physically-accurate response techniques for rigid-body animation; Problems with contact modelling; Approximate/plausible response; Graceful degradation of response.

vi. Perceptual issues in collision handling; Heuristics for perceptually-adaptive collision processing; Perception of collision physics;

An overview of these issues is presented in the following notes and in the sets of slides found in Appendix A.

1. Introduction

Everyone knows that one solid object cannot occupy the space of another. To express solidity in a simulated virtual environment, objects need to respond at the right time and in the right manner to collisions with other objects in the scene. Therefore, any physically plausible animation system, such as one that drives a Virtual Environment, needs efficient and realistic collision handling. The problems of detecting when objects collide, reporting the points of contact between them, and subsequently

determining an appropriate physical (or behavioural) response, all require a significant amount of processing. As the numbers of objects in the scene increases and/or the models become more complex, collision handling quickly becomes a major bottleneck in the system, sometimes accounting for over 95% of the overall computation time per frame [Mirtich 1996].

Many of the earlier approaches to collision detection were based on expensive object-object intersection tests. Introducing a multi-pass approach to the problem allows the elimination of expensive tests between distant objects. In addition, many techniques only provide a yes/no answer to the question of whether a collision has occurred, without providing detailed contact information. Performing accurate collision response under such circumstances is often difficult. Because of the inherent complexity of the problem, further improvements in efficiency may be achieved by using pre-computation processes. These usually involve the calculation of hierarchical approximations of the objects, thus enabling quick rejection tests. Many collision handling methods make assumptions about the rigidity and topology of the objects, but what happens when such assumptions are not valid? Collision handling between deformable objects is very computationally expensive, and most solutions are far from real-time, although recently some attempts have been made address this situation.

For dynamic interactive environments, the kinematics of objects (and hence their collisions) cannot be determined a priori. In such situations, where a high and constant frame-rate is vital, significant levels of simplification are required in order to overcome the collision handling bottleneck. Interruptible techniques attempt to optimise this speed-accuracy trade-off by adaptively managing the complexity level of the simulation. Approximate contact modelling and plausible response in such circumstances is a particularly challenging issue. In particular, knowledge of the visual perception of collision events and physics can be very useful in optimising the quality of such real-time simulations.

1.1. Collision Handling: Applications and Challenges;

The original motivation for Collision Detection algorithms arose in areas such as CAD and Robotics, where the desire was to do more work on the computer, and to off-load work from the human designer or planner. Problems such as handling many different CAD shape descriptions, VLSI layout, robot path planning, bin packing, and assembly planning gave rise to off-line algorithms, where objects positions and motions were fully predictable over time, and "what if" analysis was done to generate an optimum solution. Real-time performance was not an issue in such systems, and fully accurate mathematical intersection tests could be utilized. As time progressed, the desire for realtime and interactive systems increased. For example, CAD designers want to see the results of their new layout immediately, not several minutes later. As they reposition an object, they want to try it out in several locations, and the collision tests with other objects must be performed while they are doing this, in real-time. In such a case, it is not possible to predict in advance what will happen.

Robotics applications in the past were typically characterised by static scenes, where one or more robotic devices performed pre-specified tasks. Therefore, pre-processing could be used to predict where collisions would happen. However, with the

development of more autonomous robots, who can operate in unknown environments, and whose behaviour is impossible to predict in advance, this situation has also changed. Much cross-over research has occurred between the fields. For example, the collision detection algorithm proposed in [Lin and Canny 1991] and [Lin 1993] has been used in both robotics and animation.

The range of applications that require collision detection is extensive. Vehicle simulators are one case in point, where the users manipulate a steering device, and attempt to avoid obstacles in their way. In molecular modeling, simulations allow interactive testing of new drugs to examine how molecules interact and collide with each other. Training and education systems that realistically model the movement of objects within the geometric constraints of their layout, allow designers to experiment interactively with different strategies, e.g. to assemble or disassemble equipment, to perform a virtual surgery, or to test different paths that a robot could take. Such simulations are a safe and cheap way to teach. Human character animation is one of the most challenging topics in computer animation, and collision detection is an important issue here also. As a figure moves, collisions must be detected between the virtual person and its environment, its clothes and hair, and self-collisions between limbs and digits. Haptic interfaces are devices that allow humans to interact manually with virtual environments, and to actually feel a sense of touch via these devices as if they were really touching the virtual objects.

Virtual Environment applications allow users to enter a computer-generated virtual world and interact with graphical objects and virtual agents with a sense of reality. Such systems may be either immersive, or desktop based. One thing they have in common is a requirement for extremely high and constant frame-rates. Physical interactions such as touching, hitting and throwing are usually triggered by collision. The more objects in the environment, and the more complex these objects are, the higher the burden on the engine that powers the animation, and hence the greater the need for extremely rapid collision detection.

We have seen that there is a requirement for collision detection in almost all systems that animate graphical objects, or which need to determine potential collisions with real objects in advance. This poses a significant challenge in the area of real-time systems, and as the demand for more realism and interaction in such systems is constantly increasing, it is also likely to remain a challenge for quite some time into the future.

2. Collision Detection for Rigid Bodies

When animating more than two objects, the most obvious problem which arises is the $O(N^2)$ problem of detecting collisions between all N objects. This is known as the all-pairs problem. It is obvious that this is an undesirable property of any collision detection algorithm, and several techniques have been proposed to deal with it. Hybrid collision detection refers to any collision detection method which first performs one or more iterations of approximate tests to identify interfering objects in the entire workspace and then performs a more accurate test to identify the object parts causing interference. [Hubbard 1995a,b] and [Cohen 1995] both propose hybrid algorithms for collision detection. The former refers to the two levels of the algorithm

as the broad phase, where approximate intersections are detected, and the narrow phase, where exact collision detection is performed. Such an approach is essential for acceptable collision detection performance. The narrow phase itself may also consist of several levels of intersection testing between two objects at increasing levels of accuracy, the last of which may be fully accurate. We shall refer to these as the progressive refinement levels and the exact level respectively.

2.1. Narrow Phase: The Exact Level

Any collision detection algorithm depends on the technique used to model the objects, and the data structure used to represent that model. The narrow phase, where exact collision detection is performed, depends greatly on the object representation scheme used. Polygonal representations of surfaces are widely used to represent surfaces in 3D graphics. Surfaces, which are intrinsically planar, such as those on building exteriors and cabinets, can be easily and naturally represented in this way. However, virtually any surface can be represented by polygons if the number of polygons is sufficiently high. This leads, however, to an approximate representation only. In order to reduce the faceted effect of these surfaces, the number of polygons has to be increased to such an extent that affects space requirements and computation time of algorithms processing the surface. Many different special cases must be handled, and cases may occur where polygon edges "tunnel" through each other, or smaller surfaces pass through an entire polygon. A major advantage to using polygonal representations of objects is the fact that many manufacturers provide specialized acceleration hardware to implement common operations on polygons, such as clipping and shading.

Polyhedral methods are not well suited to surfaces that deform in time, and which roll or slide against each other. In such cases High-level surface representations are more desirable. One such representation is a collection of Parametric Patches, which are regions on a curved surface bounded by parametric curves. The number of parametric patches needed to approximate a curved surface to a reasonable degree of accuracy is many fewer than the number of polygonal patches that would approximate it to the same level. Implicit Surfaces are defined using implicit functions, and also have some desirable properties, such as being closed manifolds. This means that they define a complete solid model, not just the surface as in the case of parametric surfaces and polygonal models. Collision detection algorithms have been developed to process collisions between objects modeled via these techniques [Von-Herzen et al. 1990] [Snyder et al. 1993][Shene and Johnstone 1991]. For a more extensive survey of collision detection techniques between a variety of geometric models, see [Lin and Gottschalk 1998].

Polygonal models can be either convex, or non-convex. If a polygon is convex, that means that the line between any two points inside the polygon must also lie completely inside. The concept extends to three dimensions, where a surface composed of polygons is called a polytope. If the polytope is convex, then any line between two points inside this area must lie completely within the surface defined by such a polytope. Most of the work on collision detection techniques has concentrated on detecting collisions between convex polytopes. Such approaches fall into two broad categories: Feature-based methods, and Simplex-based methods.

Feature-based methods concentrate on the inter-relations between the vertices, edges and faces of two polytopes, i.e. their features. The main goal of such algorithms is to detect whether two polytopes are touching or not. All feature-based schemes are broadly derived from the Lin-Canny closest features algorithm [Lin and Canny 1991] [Lin 1993]. They are based on partitioning each polytope into features and constructing a Voronoi Region for each feature i.e. the set of points closer to that feature than any other. See Figure 1 for a representation of the Voronoi regions associated with a face, edge and vertex.

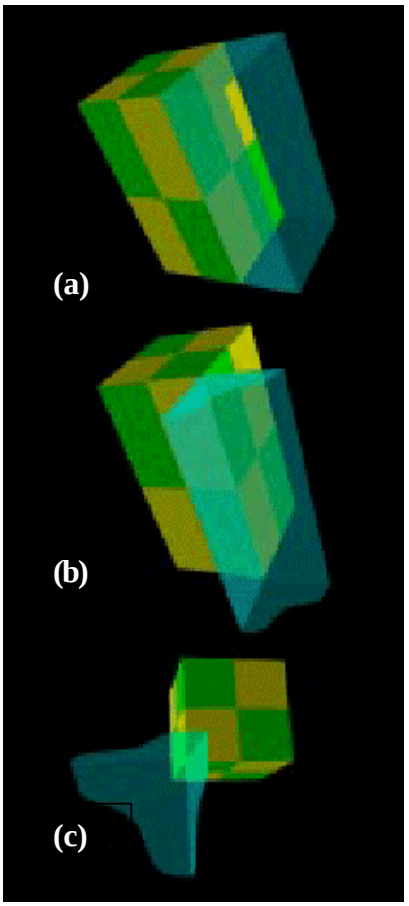


Figure 1: Voronoi regions for (a) a face, (b) an edge and (c) a vertex.

The Lin-Canny algorithm determines whether two objects are disjoint or not, by computing the distance between their closest features. It tracks these features, and caches them between subsequent calls to the algorithm. In this way it exploits coherence, because the closest features will not change significantly between successive frames, and "feature-stepping" is used to keep the closest features up to date, i.e. if the closest features have changed, they are going to be adjacent to those cached, and hence finding them is quite efficient. The algorithm to track these features runs in expected constant time if the collision detection time-step (i.e. the steps which the animation takes before each iteration of the collision detection algorithm) is small relative to the speed at which the objects are moving. The algorithm has been built into a general collision detection package I-Collide, described in [Cohen et al. 1995] and [Ponamgi et al. 1997] which is freely available on the World Wide Web.

The Lin-Canny algorithm does not handle penetrating polytopes, however, and if such a condition arises the algorithm enters an infinite loop. A possible solution to this problem is to force termination after a maximum iteration, and return a simple result stating that the objects have collided. However, this solution is quite slow, and no measure of inter-penetration is provided. Inter-penetrating objects will occur very frequently unless they are moving quite slowly, and/or if the detection time-step is quite small. This is unlikely to be the case in real-time applications such as games and Virtual Environments. If inter-penetration occurs, and more information is needed about the exact time of contact, backtracking is necessary to pinpoint the exact instant in time when collision occurred, a slow and cumbersome process. Pseudo internal Voronoi regions for convex polyhedra were introduced in [Ponamgi et al. 1997] to try to circumvent this problem. Another problem is the need to handle many special cases separately (e.g. parallel features), and the difficulty of configuration, with several numerical tolerances that need to be adjusted to achieve the desired performance.

The V-Clip (Voronoi-Clip) feature-based algorithm presented in [Mirtich 1998] has been inspired by the Lin-Canny algorithm, but claims to overcome the chief limitations of that algorithm. It handles the penetration case, needs no tolerances to be adjusted, exhibits no cycling behaviour, and is simpler to implement due to fewer special-case considerations. This is commonly held to be the fastest available published scheme for collision detection between rigid convex bodies. The main advantage to the feature-based algorithms is efficiency and fast yes/no answers to collision detection. However, for contact modelling they are not ideal.

Simplex-based algorithms are an alternative to feature-based solutions. A simplex is the generalisation of a triangle to arbitrary dimensions. The approach in these cases is to treat a polytope as the convex hull of a point set. Operations are then performed on simplices defined by subsets of these points. The first of such algorithms was presented in [Gilbert et al. 1988] and is commonly referred to as GJK. The main strength of this algorithm is that, in addition to detecting whether two objects have collided or not, it can also return a measure of interpenetration. [Rabbitz 1994] improved upon GJK by exploiting coherence, and [Cameron 1997] developed it further to produce the algorithm that is known as Enhanced GJK. This algorithm achieves the same almost-constant time complexity as Lin-Canny, while eliminating most of its main weaknesses. Mirtich claims that the V-Clip algorithm requires fewer floating-point operations than Enhanced GJK, and is hence more efficient, but it is also admitted that the GJK algorithms return the best measures of penetration.

It is often claimed that extending these “exact” techniques to handle non-convex polytopes is simple, as such polytopes can be represented by hierarchies of convex components. A “pass the parcel” approach is recommended, with collision checking being performed between the convex hulls of successive subsets of convex components, which are then unwrapped when a collision is detected. Results are rarely presented for such operations, and the focus of the validation performed is mainly on the efficiency of the intersection tests between two convex objects. Mirtich admits that although this technique works well for slightly non-convex objects, it becomes very inefficient as the level of non-convexity increases. Therefore, these techniques are very useful for situations where a small number of convex, or slightly non-convex objects are interacting in real-time, but in other situations techniques based on hierarchical representations are much more suitable.

2.2. Narrow Phase: Progressive Refinement Levels

The progressive refinement levels of the narrow phase of a collision detection algorithm are often based on using bounding volumes and spatial decomposition techniques in a hierarchical manner. Hierarchical methods have the advantage that as a result of simple tests at a given point in the object hierarchies, branches below a particular node can be identified as irrelevant to the current search and so pruned from the search.

Trees of bounding volumes are used, each level approximating the object. This is a form of Level Of Detail (LOD) representation of the object. This differs from the polygonal levels of detail used in multiresolution methods for faster rendering of complex objects, or surfaces such as mountainous terrain [Hoppe 1998][Rossignac and Borrel 1993]. In such techniques, the aim is to render an approximation that is as visually similar to the original model as possible. LODs for collision detection are always conservative approximations to the object, and the choice of volume is usually based on the speed of their intersection tests. More recently emphasis has been placed on their ability to approximate the geometry of the bounded object. Some of the hierarchies that have been used are presented in Section 3.

2.3. Broad Phase Collision Detection

[Hubbard 1995a,b] highlights three potential weaknesses of collision detection algorithms. The most serious of these is the all-pairs weakness discussed above, where every object in the scene must be compared with every other one at every collision timestep of the animation. Most research has concentrated on alleviating this problem. A second problem is what he calls the fixed-timestep weakness. Allowing the objects to move larger distances before checking whether they intersect leads to a more efficient algorithm, but it is possible that some collisions will be missed, and objects will tunnel through each other. Decreasing the size of the time-step would reduce the chances of this happening, but would cause a lot of extra unnecessary intersection tests. The third weakness refers to the narrow phase, and is the pair-processing weakness. This refers to the non-robust properties of algorithms such as the original Lin-Canny discussed above, which must handle many special cases and can exhibit strange behaviour such as cycling.

Hubbard recommends the use of an adaptive timestep, which becomes small when collisions are likely and large when they are not. For the broad phase of his algorithm, 4-dimensional structures called space-time bounds are used, which provide a conservative estimate of where an object may be in the future. The fourth dimension represents time. Overlaps of these bounds trigger the narrow phase, which is based on hierarchies of spheres. Collision detection between sphere trees is robust, thus solving the pair-processing problem. Using the space-time bounds, attention is focused on the objects that are likely to collide, and those far away can be ignored, thus alleviating both the all-pairs weakness and the fixed-timestep weakness. [Cameron 1990] also addresses the fixed-timestep weakness through the use of four-dimensional bounding structures.

In [Cohen et al. 1995] multiple object pairs are "pruned" using bounding boxes. Overlapping bounding boxes then trigger the narrow phase of the algorithm. Their "Sweep and Prune" algorithm orthogonally projects axis-aligned bounding boxes of all objects onto the x, y and z-axes. This results in intervals, of which overlaps in all three dimensions indicate overlaps of the corresponding bounding boxes. Because of coherence, the relative positions of objects will not change significantly between frames, so insertion sort is used to keep the interval lists sorted, which runs in almost linear time for almost-sorted lists.

This O(N) algorithm is extremely desirable, because it handles the all-pairs weakness, and has small computational overheads. It does not tackle the fixed-timestep weakness, but runs in almost constant time for a given number of objects. This makes it preferential in situations where a constant frame rate is required in the presence of large numbers of interacting objects. The use of an adaptive timestep could be undesirable in these circumstances because processing would slow down when many objects were close to each other, due to the smaller size of the time-steps, and then speed up as they became more evenly distributed. This would give a non-constant frame-rate, and hence lead to a jerky animation that can cause simulator sickness. However, it may be feasible to use an adaptive time-step in conjunction with adaptive techniques for the narrow phase testing (and ideally for other operations such as rendering). Efficient load-balancing mechanisms would be important in this case.

2.4. Interruptible Collision Detection

In Virtual Reality applications, in order to create an illusion of real-time exploration of a virtual world, high and near-constant frame-rates must be achieved. If the frame rate is too slow, or too jerky, the interactive feel of the system is destroyed. In a very complex environment, the objects may be modeled using thousands or even millions of polygons. To render these polygons completely accurately, with full hidden surface removal, shading, and collision detection is usually beyond the capabilities of a typical desktop computer. Even with higher-powered graphics workstations, highly variable frame rate would be the result. One solution to this problem is to adjust image quality adaptively in order to maintain a uniform frame rate. Now consider the problem of a simulation with large numbers of colliding objects. In [Mirtich 2000] an avalanche of rocks falling down a mountain was simulated, generating a large number of collisions to be handled. Each frame took on average 97 seconds to compute, because the high number and complexity of contact groups formed generated a significant bottleneck in the collision detection routines. He states, however, that when robustness is more important than efficiency, it may be necessary to accept these computation times. However, collisions in a Virtual Environment must be handled in real-time, so it is obvious that a trade-off between detection accuracy and speed is needed to achieve the required high and constant frame-rate, thus maintaining the immersiveness and plausability of the environment.

We have seen in the previous section that the use of an adaptive timestep is recommended in [Hubbard 1995a,b]. However, he goes further than this and recommends adaptive refinement at the narrow-phase level also. The idea of interruptible collision detection is presented, which allows the accuracy of intersection tests to be progressively refined until a target time has elapsed. Collisions

are detected between the sphere-trees of objects in round-robin order at increasing levels of detail, descending one level of all sphere trees at each iteration of the algorithm, until interruption occurs. This enables a fast, albeit approximate, answer when required.

The advantage of an interruptible algorithm is that the application has full control over the length of time that collision processing may take. However, Hubbard's algorithm simply returns a yes/no answer to the question of whether two objects have approximately collided or not. There is no facility to improve response based on the increasing accuracy of the tests when time allows. In addition, inaccuracies in the handling of collisions may cause the viewer to perceive unrealistic behaviour of colliding entities, such as them bouncing off at a distance or not rotating appropriately. In [Dingliana and O'Sullivan 2000], [Dingliana et al. 2001] and [O'Sullivan et al. 1999], an interruptible collision handling scheme is presented which attempts to alleviate both these problems. Firstly, perceptually-guided scheduling of collision processing is used, which allows the application to prioritise collisions and assign more processing time to those that are more important (see section 6 for further details). Secondly, graceful degradation of response is achieved by using approximate contact information to provide an optimal physical response with the most accurate data available (see section 5).

3. Bounding volumes: construction and evaluation

As mentioned in Section 2.2 the narrow phase processing often uses a tree of bounding volumes to cull out areas of the objects that cannot be in contact. Here is a list of some hierarchies that are used:

- **Octrees** [Sammet and Webber 1988] Octrees are built by recursively sub-dividing the volume containing an object into eight octants, and retaining only those octants that contain some part of the original object as nodes in the tree. Such a data structure is simple to produce automatically, and lends itself to efficient and elegant recursive algorithms. The disadvantage of this approach is that each level of the hierarchy does not fit the underlying object very tightly.
- **Sphere Trees** [Hubbard 1995a, 1996][Palmer and Grimsdale 1995][Quinlan 1994]. The main advantages of using spheres are that they are rotationally invariant, making them very fast to update, and it is very simple to test for distances between them, and test for overlaps. The disadvantage is that spheres do not approximate certain types of objects very efficiently. Hubbard attempts to improve upon this by building first a medial axis surface, which is like a skeleton representation of an object, and then placing the spheres upon this to provide a tighter-fitting approximation to the object. [O'Rourke and Badler 1979] also developed a method of tightly fitting spheres to an object.
- **C-trees** consist of a mixture of convex polyhedra and spheres [Youn and Wohn 1993]. This has the advantage of choosing primitives that best approximate the enclosed object, but a major drawback is that the hierarchy must be created by hand, and cannot be produced automatically. A similar approach is taken in [Rohlf and Helman 1994].

- **OBB-trees** [Gottschalk et al. 1996]. These hierarchies consist of tightly fitting Oriented Bounding Boxes. It is claimed that using an algorithm based on a separating axis, all the contacts between large complex geometries can be detected at interactive rates. However, it is admitted that other methods are very good at performing fast rejection tests, and a disadvantage of OBB-trees over Sphere trees is that they are slower to update.
- **AABB-trees** [Van Den Bergen 1997]. Axis Aligned Bounding Boxes are used, the advantage of these being their ease of computation and overlap testing. The disadvantage is, however, that the bounding boxes must be re-calculated whenever the objects rotate, or a separating axis test must be performed, as for OBB-trees.
- **K-DOPs** [Klosowski et al. 1997]. This approach uses hierarchies of k-DOPs, or discrete orientation polytopes, which are convex polytopes whose facets are determined by half spaces whose outward normals come from a small fixed set of k orientations. Again, they implement it with a small number of highly complex objects, for the purposes of haptic force-feedback. If there are a large number of objects between which fast rejection or acceptance is needed, the update time needed for these approximations is likely to add an unacceptable additional burden. This approach is a generalisation of AABBs (which are actually 6-dops), and therefore also suffers from the need for dynamic updating of the nodes.
- **ShellTrees** [Krishnan et al 1998a,b]. These trees consist of oriented bounding boxes and spherical shells, which enclose curved surfaces such as Bezier patches and NURBS. They are particularly suited to collision detection between the higher-order surface representations discussed in the previous section.
- **Swept Sphere Volumes** [Larsen et a. 1999]. A swept sphere volume is a sphere that is swept out along a geometric primitive, such as a point (a sphere) line (a cylinder with rounded ends) or rectangle (a cuboid with rounded edges and corners). These volumes provide a means varying the shape of the bounding primitive to achieve a tighter fit to the underlying geometry, without the disadvantage of having to compute them by hand. Similar performance results to OBB-trees are reported, although the potentially better fit provided by the swept voumes should provide more accurate collision tests, especially if no exact testing is performed at the end of the narrow phase, e.g. in the case of interruptible collision detection.

While many of the algorithms that use these Bounding Volume Hierarchies do so simply as an acceleration technique, interruptible collision detection [Hubbard 1995a] uses the hierarchies to produce a definite answer. Thus it is not only necessary to have tight fitting hierarchies – but it is also necessary to be able to compute the points of contact that are needed for contact modelling and collision response. For example the Separating Axis Test, used for OBBs and k-DOPs, simply provides a yes/no.

A common way to evaluate such hierarchies is based on the time taken to perform the narrow phase traversal. As the objects are in motion it is necessary to update each of the primitives as the traversal occurs. Having updated a pair of nodes, so that they are both in the same co-ordinate frame, it is necessary to determine if they overlap. If the two nodes don't overlap then their children nodes need not be considered. The following equation expresses the cost of performing collision detection between two objects:

$$T_C = N_u * C_u + N_v * C_v$$

where C_u and C_v are the update and overlap costs respectively, and N_u and N_v are the number of nodes to be updated and tested. The number of bounding volumes needed to represent an object generally depends on the primitive used. Generally, the more degrees of freedom the primitive has the better it can approximate an object. Spheres and AABBs generally converge quite slowly to the object's geometry, whereas OBBs and Spherical Shells converge much faster. It is also generally the case that the more degrees of freedom the primitive has – the higher the cost of updating (C_u) and of overlap testing (C_v) will be.

This section of the tutorial will look at some of the bounding volume hierarchies mentioned and will use the above metric to evaluate their suitability for use in interruptible collision detection. More details may be found in the slide set in Appendix A.

4. Deformable object animation

This section reviews the approaches to collision detection that take deformations into account. The ability to detect and handle a collision between deformable objects is commonly seen as a "special feature" rather than a different problem. After all, the introduction of deformable objects in real-time environments is still a recent issue, so it is natural to try to extend the known algorithms, i.e. those for rigid bodies, to the case of objects that deform, before developing ad hoc algorithms.

Unfortunately it turns out that most of the known approaches, and especially the most efficient ones, are not really applicable to this new context. The reason is that they either use structures that depend on the shape of the objects (OBBTree, k-DOPs, sphereTrees, shellTrees) or assume that the objects are convex (feature based, V-clip, GJK). Furthermore, taking deformation into account leads to two new problems:

- **self-intersection**: this term refers to the fact that a deformation can possibly lead to a contact between different parts of the same surface, especially when it is experiencing a high degree of deformation as, for example, a piece of cloth. This problem introduces a factor of $O(m^2)$ to the complexity of the brute force solution, where m is the number of primitives composing the surface;
- **cuts**: the ability to cut the objects is a must for many real applications, especially the ones related to virtual surgery. This problem has a deep impact in both modelling the physics of the object and in detecting the possible collisions, because it requires discarding the assumption that at least the description of the surface never changes.

This part of the tutorial presents a set of techniques specifically designed for collision detection between deformable objects. It identifies the situations that each approach is suitable for, i.e. which assumptions it makes. The first part relates to collision between surfaces defined by parametric functions, emphasising the advantages of having such a compact description in computing hierarchies of bounding boxes and in testing for self-intersection [Von-Herzen et al. 1990, Hughes et al. 1996]. The second part is devoted to collision detection between polyhedral surfaces. Under the

assumption that the connectivity of polyhedra does not change, [Van Den Bergen 1997] showed that a hierarchy of Axis Aligned Bounding Boxes can be kept updated on the fly and [Volino et al. 1995] devised an approach to detect self-intersection through the use of triangles' normals.

If the assumption on connectivity is also discarded, a surface may simply be considered as a soup of polygons. The no-assumption approaches, which can handle cuts, are generally based on indexing the space occupied by the bounding box of the object. In [Kitamura et al. 1998], when the bounding boxes of two different objects overlap, a hierarchy of Axis Aligned Bounding Boxes is built over the polygons intersecting the overlap region, while in [Ganovelli et al 2000] such a hierarchy is kept updated for each object, exploiting frame-to-frame coherency.

The special case of collision detection between a rigid object with a simple shape and a deformable object has been investigated in the context of models applied to virtual surgery, where ad hoc techniques, not applicable to the general case, have been developed [Cotin et. al 1998, Lombardo et. al 1999].

Further details about these techniques may be found in the slide set in Appendix A. It will emerge how challenging it is to define an efficient algorithm for the general case and therefore how many opportunities for research collision detection between deformable models still offers.

5. Contact Modelling and Collision Response

The primary goal of collision response is to deal in some consistent and expected manner with objects, which the detection mechanism has identified as colliding [Moore and Wilhelms 88]. The actual response will be specific to the application domain and may range from something as simple destroying one of the objects in a game, choosing a different path for one of the objects in a collision avoidance system or moving two objects apart so they are no longer colliding after the collision event. In self-regulating physically based animations we are primarily interested in rules that create behaviours in objects that are representative of their real-world counterparts. Solid objects should therefore not interpenetrate and if they should collide due to their movements across the scene, we expect that they will bounce away in directions and velocities that depend on their initial states and properties e.g. mass-distribution, relative sizes and densities.

Collisions are a primary source of simulation discontinuity, where the states of objects or the laws that drive the evolution of states change instantaneously. Collisions are therefore a source of much additional computational workload. As we deal with a simulation on a discrete timestep by timestep scheme we often only detect a collision when two objects disjoint at time t_0 are found to be interpenetrating at time $t_0+\Delta t$. For accurate results a rollback to the precise moment of contact is required and simulation should ideally restart from that point [Baraff and Witkin 97]. However this is expensive as it entails not only determining (or reasonably approximating) the exact contact time but discarding all the updates to the world that have been applied for the interpenetration frame. Techniques exist to improve this necessary rollback [Mirtich 00] but this remains a major problem for speed dependant interactive environments.

The requirement for fast and consistent frame rates in interactive animations can only be met through simplification and culling; by trading accuracy for speed. Many approaches try to optimise this trade-off by means of interactive techniques [Funkhouser and Sequin 93][Chenney and Forsyth 97][Dingliana and O'Sullivan 00]. Where accuracy is sacrificed, visually plausible results can still be achieved cheaply by the introduction of non-determinism (whether it be by design or implicitly due to simplifying approximations)[Barzel et al 96].

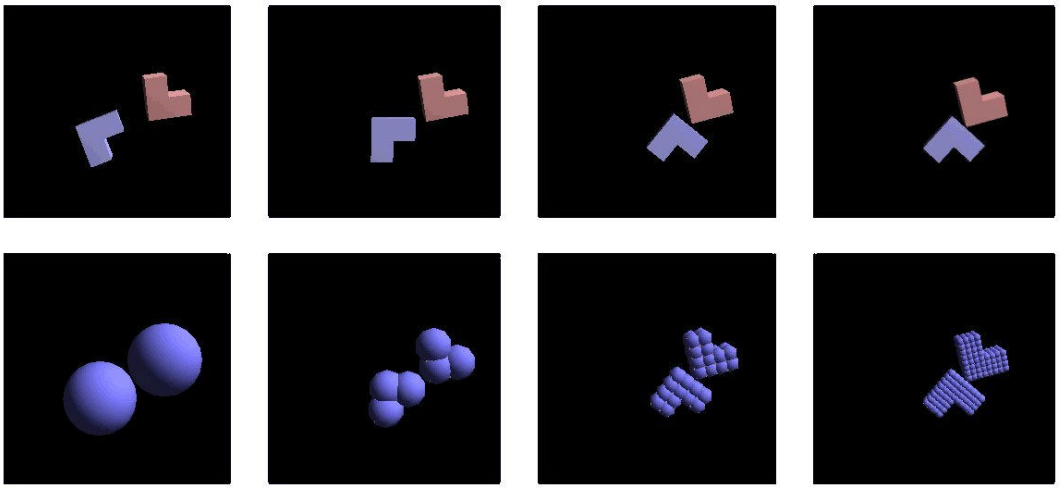
Traditional techniques for calculating contact responses can be roughly classified into penalty methods, impulse based methods and constraint based methods. The impulse-based methods allow for local calculation of state changes and are most suited for interactive simulation. All methods require fairly detailed input about the object to object contact points (or manifolds) and an intermediate contact modelling phase is necessary particularly in the case of refinable collision detection methods which have been traditionally used to only return a "yes, no or maybe" result about a collision. Ideally the narrow phase implements fine polygon-level intersection tests that produces accurate response data [Palmer and Grimsdale 95]. However, in time-critical implementations we are required to make an approximation of the contact from the results of an imprecise detection mechanism. As the collision detection is refined so too is the contact approximation and this is the convenient basis for refining the collision response (an otherwise "non-negotiable" constant time process). In hierarchical collision detection contact data becomes more precise as we traverse deeper down the hierarchy. The number of intersecting nodes increases and so does the collision response workload. A scheduler for such a system needs to project time for collision response and pre-allocate this from the full time-quota. These issues are presented in greater detail in the slide sets in Appendix A.

6. Perceptual Issues in Collision Handling

As stated in Section 2.4, Dingliana and O'Sullivan extended Hubbard's interruptible collision detection routine to return collision data, which can be used in computing physically based responses to object collisions. As with the collision detection process the accuracy of the data (such as details of contact points) is improved as the mechanism traverses deeper into the sphere tree. As we deal with higher resolutions of the volume model, the approximated contact points become increasingly accurate (see Figure 2). If the mechanism is interrupted, e.g. when it has used up its allocated time quota, then it immediately returns the most accurate approximation it has so far computed.

We must remember that we are dealing with a viewer-centric model and that what we are trying to optimise is the plausability of the animation rather than it's mathematical accuracy [Barzel et al. 1996][Chenney and Forsyth 2000]. In the viewer-centric model, the first thing we should note is that objects (and collisions between objects) further away from the user, due to occlusion or perspective fore-shortening become more difficult to judge and, as a result, errors and approximations in their states and behaviours become more difficult to notice. Furthermore, even a casual study will show that users' primary awareness of events in a scene focuses in a small radius around the point of fixation (see Figure 3).

Other factors, which might influence the user's ability to judge an event on the scene, are properties of the objects involved e.g. size, shape, velocity, separation of the colliding entities; or properties of the scene and surroundings e.g. crowdedness, lighting, etc. Since the user's ability to notice error is non-uniformly spread across the scene, this suggests that processing time (and as a result simulation accuracy) for different events across the scene should also not be uniform. Instead, events should be prioritised across the scene, allowing the scheduling of processing time to be based on some measure of the importance of a collision or other event. A similar approach to the use of heuristics to select levels of detail for rendering is described in [Funkhouser and Sequin 1993] and [Reddy 1998]. The level of optimisation depends a great deal on the quality of the metrics used to perform the prioritisation of objects in the scene. Although good results have been achieved by using "obvious" metrics such as object velocities, projected screen distances and distance from the user's fixation point (determined with the use of an interactive eye-tracker, see Figure 4) more extensive studies need to be performed to identify the most important factors which affect user perception of events and to determine how the influence of all such factors varies across a simulation scene.



*Fi**gure 2: Multi-resolution collisions between objects.** The bottom row of images show the volumes used to perform collision detection at increasing levels of detail, while the top row shows what is actually seen by the viewer at the moment of impact.*

In [O'Sullivan et. al 1999] and [O'Sullivan and Dingliana 2001], criteria for prioritising collisions are presented. Psychological experiments are described which aim to determine the factors that influence people's perception of dynamic events such as collisions and physical behaviours with the purpose of developing dynamically-calculated metrics to drive the perceptual scheduling of our real-time adaptive physical simulations. In the former it was also demonstrated that the overhead from a full prioritisation and sorting of events in the scene on a per-frame basis becomes too high. A more fruitful approach is to use a small number of different priority groups into which events are interactively distributed. Each priority group is then allocated its share of processing time by the scheduler, with more processing being spent on higher priority groups. This method, whilst preserving a prioritisation scheme, bears considerably less overhead expense than a full continuous sort and in practice delivers good results even with very small numbers of priority groups.

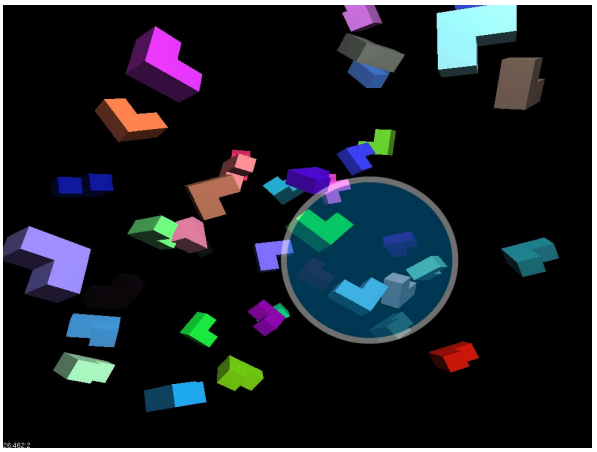


Figure 3: Important collisions, e.g. those close to the viewer's fixation position, should be processed first.

Figure 4: An eye-tracker is used to determine the viewer's point of fixation

Bibliography

[Baraff and Witkin 1997] Baraff, B. and Witkin, A. Introduction to Physically based Modeling. Siggraph '97 Course Notes.

[Barzel et al. 1996] Barzel, R. Hughes, J.F. Wood, D.N. Plausible Motion Simulation for Computer Graphics Animation. Computer Animation and Simulation '96. 183-197.

[Cameron 1990] Cameron, S.A. Collision Detection by Four-Dimensional Intersection Testing. IEEE Transactions on Robotics and Automation. 6(3) 291-302.

[Cameron 1997] Cameron, S.A. Enhancing GJK: Computing Minimum Penetration Distances Between Convex Polyhedra. Proceedings of the Int. Conf. On Robotics and Automation. 3112-3117.

[Carlson and Hodgins 1997] Carlson D.A. Hodgins, J.K. Simulation Levels of Detail for Real-time Animation. Proceedings Graphics Interface '97. 1-8.

[Chenney and Forsyth 1997] Chenney, S. and Forsyth, D. View-dependent Culling of Dynamic Systems in Virtual Environments. Proceedings ACM Symposium on Interactive 3D Graphics 1997. 55-58.

[Chenney and Forsyth 2000] Chenney, S. and Forsyth, D. – Sampling Plausible Solutions to Multi-Body Constraint Problems. Proceedings SIGGRAPH 2000. 219-228.

[Cohen et al. 1995] Cohen, J.D. Lin, M.C. Manocha, D. Ponamgi, M.K. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments. Proceedings ACM Int. 3D Graphics Conference. 189-196.

[Cotin et. al 1998] S. Cotin and H. Delingette and N. Ayache Real-time Elastic Deformations of Soft Tissues for Surgery Simulation. IEEE Transactions on Visualization and Computer Graphics. 5(1) 62-73.

[Dingliana et al. 2001] Dingliana, J. O'Sullivan, C. Bradshaw, G. Collisions and Adaptive Levels of Detail. SIGGRAPH 2001 Technical Sketches Program (To appear)

[Dingliana and O'Sullivan 2000] Graceful Degradation of Collision Handling in Physically Based Animation. Dingliana, J. O'Sullivan, C. Computer Graphics Forum. (Proceedings, Eurographics 2000). 19(3) 239-247

[Funkhouser and Sequin 1993] Funkhouser,T.A. Sequin, C.H. (1993) Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments. Proceedings, SIGGRAPH '93. 247-254.

[Ganovelli et al. 2000] Ganovelli, F. Dingljana, J. O'Sullivan, C. BucketTree: Improving Collision Detection Between Deformable Objects. Proceedings SCCG 2000, 156-163

[Gilbert et al. 1988] A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE Journal of Robotics and Automation. 4(3) 193-203.

[Gottschalk et al. 1996] Gottschalk, S. Lin, M.C. Manocha, D. (1996) OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. Proceedings SIGGRAPH '96, 171-180.

[Hoppe 1998] Hoppe, H. Smooth view-dependent level-of-detail control and its application to terrain rendering. IEEE Visualization '98. 35-42.

[Hubbard 1995a] Philip M. Hubbard, Collision Detection for Interactive Graphics, PhD Thesis, Department of Computer Science, Brown University, March 1995.

[Hubbard 1995b] Hubbard, P.M. Collision Detection for Interactive Graphics Applications. IEEE Transactions on Visualisation and Computer Graphics. 1(3) 218-230.

[Hubbard 1996] Hubbard, P.M. Approximating Polyhedra with Spheres for Time-Critical Collision Detection. ACM Transactions on Graphics. 15(3) 179-210.

[Hughes et al. 1996] M. Hughes, C. DiMattia, M.C. Lin and D. Manocha. Efficient and Accurate Interference Detection for Polynomial Deformation. Proceedings Computer Animation '96. 155-166

[Kitamura et al. 1998] Kitamura, Y. Smith, A. Takemura, H. Kishino, F. A real-time algorithm for accurate collision detection for deformable polyhedral objects. Presence. 7(1) 36-52.

[Klosowski et al. 1998] Klosowski, J.T. Held, M. Mitchell, J.S.B. Sowizral, H. Zikan, K. (1998) Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. IEEE transactions on Visualization and Computer Graphics. 4(1) 21-36.

[Krishnan et al. 1998a] Krishnan, S. Pattekar, A. Lin, M. Manocha, D. Spherical Shell: A higher order bounding volume for fast proximity queries. In Proceedings of the Third International Workshop on Algorithmic Foundations of Robotics. 177-190.

[Krishnan et al. 1998b] Krishnan, S. Gopi, M. Lin, M. Manocha, D. Pattekar, A. Rapid and Accurate Contact Determination between Spline Models using ShellTrees. Proceedings Eurographics 1998. 315-326

[Larsen et al 1999] Larsen, E. Gottschalk, S. Lin, M. and Manocha, D. Fast proximity queries with swept sphere volumes, Technical Report, Dept of Computer Science, University of North Carolina, 1999.

[Lin 1993] Lin, M.C. Efficient Collision Detection for Animation and Robotics. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, December 1993.

[Lin and Canny 1991] Lin, M.C. Canny J.F. Efficient algorithms for incremental distance computation. IEEE Conference on Robotics and Automation, 1008-1014.

[Lin and Gottschalk 1998] Lin, M.C. Gottschalk, S. Collision Detection between Geometric models: A Survey. Proceedings of IMA conference on Mathematics of Surfaces. 37-56

[Lombardo et. al 1999] J. Lombardo and M. Cani and F. Neyret. Real-time collision detection for virtual surgery. Proceedings, Computer Animation 1999. 33-39

[Mirtich 1996] Mirtich, B. Impulse Based Dynamic Simulation of Rigid Body Systems. PhD Thesis, University of California, Berkeley.

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

[Mirtich 2000] Mirtich, B. Timewarp Rigid Body Simulation. Proceedings SIGGRAPH 2000. 193-200.

[Moore and Wilhelms 88] Moore, M. and Wilhelms, J. Collision Detection and Response for Computer Animation. Proceedings SIGGRAPH'88. 289-298.

[O'Rourke and Badler 1979] Decomposition of three-dimensional objects into spheres. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-1(3) 295-305.

[O'Sullivan and Dingliana 2001] Real vs. Approximate Collisions: When can we tell the difference. SIGGRAPH 2001 Technical Sketches Program. (To appear)

[O'Sullivan et al. 1999] O'Sullivan, C. Radach, R. Collins, S. A Model of Collision Perception for Real-Time Animation. Computer Animation and Simulation '99. 67-76.

[Palmer and Grimsdale 1995] Palmer, I.J. Grimsdale, R.L. Collision Detection for Animation using Sphere-Trees. Computer Graphics Forum, 14(2) 105-116.

[Ponamgi et al. 1997] Ponamgi, M.K. Manocha, D. Lin, M.C. Incremental Algorithms for Collision Detection Between Polygonal Models. IEEE Transactions on Visualization and Computer Graphics. 3(1) 51-64.

[Quinlan 1994] Quinlan, S. (1994) Efficient Distance Comp utation between Non-Convex Object. Proceedings International Conference on Robotics and Automation. 3324-3329.

[Rabbitz 1994] Rabbitz, R. Fast Collision Detection of Moving Convex Polyhedra. Graphics Gems IV, P.S. Heckbert, Ed. Academic Press, Cambridge, MA, 83-109.

[Reddy 1998] Reddy, M. Specification and Evaluation of Level of Detail Selection Criteria. Virtual Reality: Research, Development and Application. 3(2) 132-143.

[Rossignac and Borrel 1993] Rossignac, J. Borrel, P. Multi-resolution 3D approximations for rendering complex scenes. Geometric Modeling in Computer Graphics. 455-465.

[Sammet and Webber 1988] Sammet, H. and Webber, R. Hierarchical Data Structures and Algorithms for Computer Graphics. IEEE Comp. Graphics and Applications. 8(3) 48-68.

[Shene and Johnstone 1991] Shene, C. Johnstone, J. On the planar intersection of natural quadrics. Proceedings of ACM Solid Modeling. 234-244.

[Snyder et al. 1993] Snyder, J.M. Woodbury, A.R. Fleischer, K. Currin, B. Barr, A. Interval methods for Multi-Point Collisions between Time-Dependent Curved Surfaces. Proceedings SIGGRAPH'93 321-334.

[Van Den Bergen 1997] Efficient Collision Detection of Complex Deformable Models using AABB Trees." Journal of Graphics Tools, 2(4) 1-13.

[Volino et al. 1995] Volino, P. Courchesne, M. Thalmann, N. Versatile and Efficient Techniques for Simulating Cloth and Other Deformable Objects. Proceedings SIGGRAPH 95. 137-144.

[Von-Herzen et al. 1990] Von-Herzen, B. Barr, A.H. Zatz, H.R. Geometric Collisions for Time-Dependent Parametric Surfaces. Proceedings SIGGRAPH'99. 39-48.

[Youn and Wohn 1993] Youn, J.H. Wohn, K. Realtime Collision Detection for Virtual Reality Applications. Proceedings IEEE Virtual Reality Annual Int. Sym. 18-22.

Appendix A: Tutorial Slides

Bounding Volume Hierarchies for Interactive Collision Handling

Gareth Bradshaw
Image Synthesis Group
Trinity College Dublin

E-Mail : Gareth.Bradshaw@cs.tcd.ie
WWW : http://isg.cs.tcd.ie/gbrdshaw

Interactive Simulation Recap

- Time-Step Simulation
  - Update body's position and orientation based on linear and angular velocities
  - Update body's linear and angular velocities based on acceleration and gravity
  - Determine interactions (*Collision Detection*) and apply relevant impulses (*Collision Response*)
- Modelling objects for detection and response

Weaknesses
[Hubbard 1995a]

- Fixed Time-Step Weakness
  - Position and orientation of body is updated at each time-step
  - Interactions are only handled at these discrete time-steps
  - Simulation time is incremented by fixed amount Δt
    - Large Δt gives good efficiency to simulation
    - Smaller Δt gives more accurate collisions

Weaknesses (2)

- All-Pairs Weakness
  - For N objects we have $N^2/2$ object pairs - each having a potentially colliding pair of objects
  - How do we efficiently handle this exponentially increasing number ? Do we test all object pairs for bounding sphere overlap ?

Weaknesses (3)

- Pair Processing Weakness
  - When checking a pair of objects we check if their models are in contact or even penetrating
    - Polyhedral model : check each face, edge and vertex for the various cases under which this occurs
    - Voxel model : check for overlapping of active voxels between the two models

Algorithm Overview

- Multiphase algorithm which addresses the *All-Pairs* and *Pair-Processing* problems.
  - Broad phase : efficiently cull pairs whose bounding boxes don't overlap. "*Sweep and Prune*" exploits inter-frame coherence to achieve near O(n) performance [Cohen et al. 1995]
  - Narrow phase : utilise spatial localisation techniques to narrow in on the areas of objects which are in contact and use the underlying polygon model to determine the actual points of contact

Narrow Phase for Dynamic Simulation

- Objects are in motion
  - Need to be able to update the spatial sub-division structure at each simulation step
  - Each object has a hierarchy of bounding volumes. Each level of the hierarchy forms a tighter approximation of the object
    - *Bounding Volume Hierarchy (BVH)*

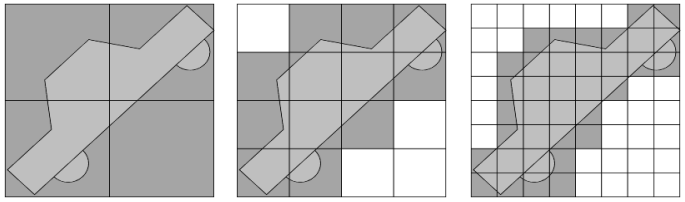
Interruptible Algorithm

[Hubbard 1995a, Dingliana and O'Sullivan 2000]

- Maintain consistently high frame-rates
  - Smooth simulation
    - prevents simulator sickness
- Omits the final stage and interrupts the narrow phase when time slot has expired
  - Time critical computing
  - BVH approximation is used for collision response
    - Tightness of BHV is critical to simulation quality

BVH - Example

- Quad-tree (2D equivalent of oct-tree)



Top Level
4 nodes

Second Level
4 children per node

Third Level
4 children per node

BVH - Desirable Properties

- The hierarchy approximates the bounding volume of the object, each level representing a tighter fit than its parent
- For any node in the hierarchy, its children should collectively cover the area of the object contained within the parent node
- The nodes of the hierarchy should fit the original model as tightly as possible

BVH - Desirable Properties (2)

- The hierarchy should be able to be constructed in an automatic predictable manner
- The hierarchical representation should be able to approximate the original model to a high degree or accuracy
  - allow quick localisation of areas of contact
  - reduce the appearance of object repulsion

Cost of Narrow Phase

- Object interactions determined by traversal of a pair of hierarchies
  - Update a node in the hierarchy (C_u, N_u)
  - Detection of overlap between nodes (C_v, N_v)

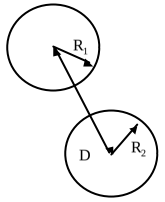
$$T_c = N_u * C_u + N_v * C_v$$

  - N_u & N_v depends on the number of nodes
  - C_u & C_v depends on the primitive used

Sphere-Tree

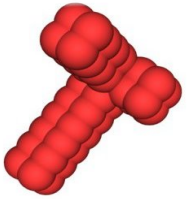
[O'Rourke and Badler 1979 , Hubbard 1995a & 1996, Palmer and Grimsdale 1995, Dingliana and O'Sullivan 2000]

- Nodes of BVH are spheres.
- Low update cost C_u
  - translate sphere center
- Cheap overlap test C_v
- Slow convergence (Linear) to object geometry
  - Relatively high N_u & N_v

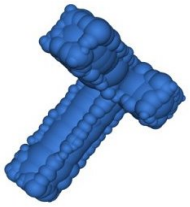


$D^2 < (R_1 + R_2)^2$

Sphere-Tree Construction



OCTREE
Spheres placed on regular grid covering object

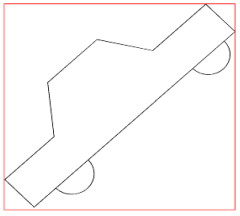


MEDIAL AXIS
Spheres placed on object's medial axis

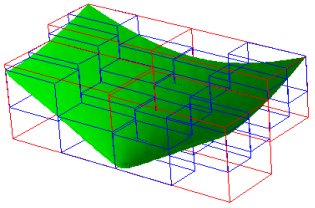
Axially Aligned Bounding Box

[van den Bergen 1997]

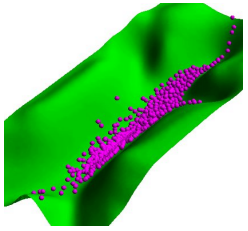
- Overlap cost C_u can be low if objects don't rotate
  - Three 1D interval tests
- Become OBB when object rotates
  - Refit AABB
    - refit around object - high C_u
    - refit around original AABB - low C_u at cost of fit quality
  - Use OBB overlap test
    - expensive
- Can be used for terrains and stationary objects



AABBs and Terrains



two level of AABB representation of a terrain

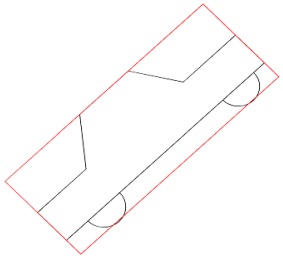


1000 balls rolling down valley terrain collision takes 0.01sec

Oriented Bounding Box

[Gottschalk et al. 1996]

- Better coverage of object than AABB
  - Quadratic convergence
- Update cost C_u is relatively high
  - reorient the boxes as objects rotate
- Overlap cost C_v is high
  - Separating Axis Test tests for overlap of box's projection onto 15 test axes



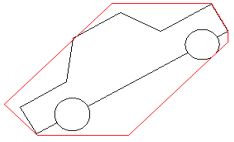
OBB Fitting

- Principle Component Analysis (PCA)
  - Eigen vectors of Covariance matrix or Inertial Tensor
  - Object represented by ellipsoid, the axes of which are used as the basis for the OBB
    - encounters difficulties when axes are similar in length
  - Children are generated by splitting the longest axis into two halves

Discrete Oriented Polytope

[Klosowski et al. 1997]

- Convex polytope whose faces are oriented normal to *k* directions
- Overlap test similar to OBB
  - *k/2* pairs of co-linear vectors
  - k/2 overlap tests
- *k*-DOP needs to be updated in a similar way to the AABB
- AABB is a 6-DOP



Spherical Shell

[Krishnan et al. 1998a]

- Section contained between two concentric spheres & within a solid angle of an orientation vector
- Small update costs C_u
  - update center and orientation vector
- Overlap test C_v is 2 to 3 times that of OBB
- Cubic convergence to object geometry
  - Low N_u & N_v

Swept Sphere Volume

[Larsen et al. 1999]

- Convolution of sphere with geometric primitive
  - Point Swept Sphere (PSS)
  - Line Swept Sphere (LSS)
  - Rectangle Swept Sphere (RSS)
- Provides similar tightness of fit to OBB
  - Fitted using an OBB
- Reduces update cost (C_u) and overlap cost (C_v) whenever geometry is well suited to the PSS/LSS

Hierarchy Construction

- Top Down
  - Fit Primitive to object
  - Recursive subdivision to produce children
- Bottom up
  - Approximate the object with a large number of primitives
  - Merge objects into groups to produce parent nodes

Hybrid Hierarchy

- Each node is fitted with the best from a set of n primitive types.
  - Increase tightness of representation
  - $O(n^2)$ different overlap tests

Collision detection between Deformable Objects: almost another problem

Fabio Ganovelli
Image Synthesis Group
Trinity College Dublin

Email: Fabio.Ganovelli@cs.tcd.ie

CD for Deformable Objects

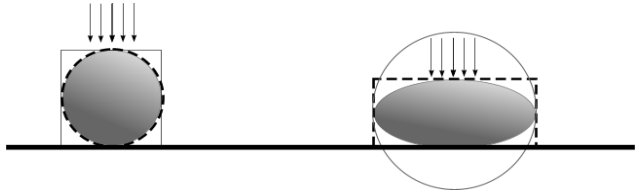
- Most of the approaches to collision detection for rigid objects cannot be naturally extended to this context. They use data structures that:
  - Can hardly be computed on-the-fly
  - Strictly depend on the shape of the object
- Example:
  - Hierarchy of Bounding volumes: OBB-tree, k-DOP, BSP
  - For convex polytopes: feature-based, V-CLIP, GJK
  - Furthermore, there is a problem that does not exist for rigid objects: self-intersection
- …and don't forget a typical feature of deformable objects: change of topology and/or (from the point of view of the mesh) change of tessellation of the surface

Outline

- Hierarchies are still useful: use of the simplest bounding volumes is suitable for on-the-fly updating
- CD for parametric surfaces: exploiting mathematical definitions of the surface to compute bounding volumes and to detect self-intersection
- CD in cloth simulation: self intersection for polygonal surfaces
- CD between a solid and a deformable object: an easier problem
- No-assumption approaches: the surface as a soup of polygons
  - Indexing the space
  - Indexing the polygons
- Conclusion: opportunities for collision detection in deformable objects modelling

Hierarchies

- Axis Aligned Bounding Boxes are the most used, because they are:
  - Easiest to compute
  - Easiest to check for interference
- They do not provide the best fitting volume, but this concept becomes a bit fuzzy for deformable objects
- The manner of computing the bounding volumes marks the most notable difference among the various approaches



The best fitting is the sphere

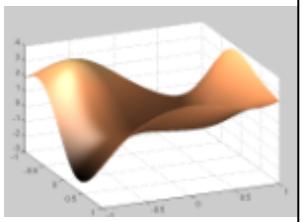
The best fitting is the box

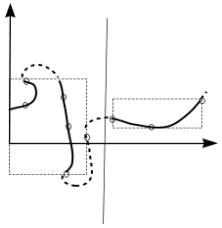
CD for Parametric Surfaces

- Any parametric surface is described by a function

$$f(u) = (f_x(u),\ f_y(u),\ f_z(u))$$

where u is a value in the parametric space
- Advantages:
  - a compact analytical description of the surface
  - simpler deformations
- Naïve computation of bounding volumes
  - compute the bounding boxes of sampling points
  - Accuracy depends on sampling rate
  - The knowledge of the parametric function is not used





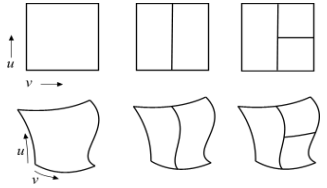
CD for Parametric Surfaces

[Von Herzen et al. 1990]

- **Problem statement** the surfaces are parametric functions over (u, v, t)
- **Assumption**: the functions are continuous and respect Lipschitz Conditions (note: more general than "finite partial derivatives")

$$\left\|\vec{f}(\vec{u}_2) - f(\vec{u}_1)\right\| \le L\left\|\vec{u}_2 - \vec{u}_1\right\|$$

- **Hierarchy:** *k-trees*, binary subdivision on a *k* dimensional box along one dimension (generalization of the *bin-trees*)

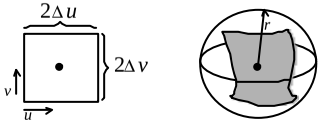


CD for Parametric Surfaces

[Von Herzen et al 1990]

- **Bounding Sphere**: it easy to show that $f(R)$ is bounded by a sphere centered in the middle point of the rectangular region R, with radius:

$$r \ge \max_R \left\{ \left\| \frac{\partial \vec{f}(\vec{u})}{\partial u} \right\|_2, \left\| \frac{\partial \vec{f}(\vec{u})}{\partial v} \right\|_2, \left\| \frac{\partial \vec{f}(\vec{u})}{\partial t} \right\|_2 \right\} (\Delta u + \Delta v + \Delta t)$$



- A tighter bounding volume (a box) is built using Lipschitz conditions on the single components of the parametric functions.

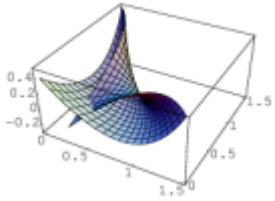
Parametric surfaces and self-intersection

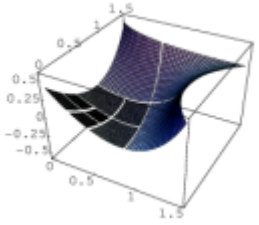
[Hughes et al. 1996]

- **Problem statement** the surfaces are Bezier and B-splines patches

$$F(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij}\ B_{i,m}(u)\ B_{j,n}(t)$$

- **Assumption**: surfaces undergoing polynomial deformation
- **Hierarchy:** in a preprocessing step, each surface is subdivided in smaller patches and respective control points are computed
- **Bounding volumes:** axis aligned boxes as root of the hierarchy and convex hull of the control points for the rest of the tree



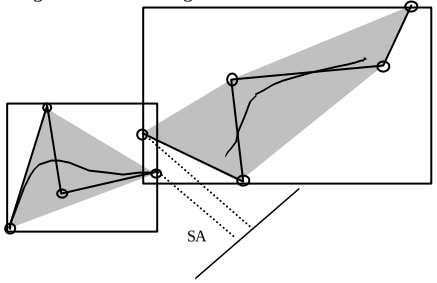


Parametric surfaces and self-intersection

[Hughes et al. 1996]

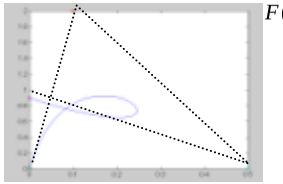
- Collision between different patches: use of the convex hull property: "the surface is inside the convex-hull of the control points"
- Use of incremental algorithm to compute the separating axes, without explicitly constructing the convex hull, in average linear time
- Note: convex hulls are tighter than bounding boxes

Bounding boxes overlap, convex hulls do not



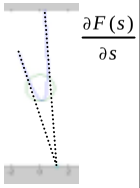
Parametric surfaces and self-intersection

[Hughes et. al 1996.]



- **Self-intersection:** "*if* a spline patch self-intersects *then* there are at least two points whose normals are opposite in direction"
- The *pseudo-normal* patch is still a spline

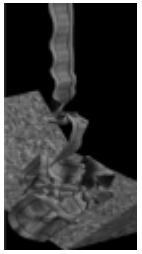
$$N(u,v) = \frac{\partial F(u,v)}{\partial u} \times \frac{\partial F(u,v)}{\partial v}$$



- If the projection of the pseudo-normal patch onto the unit sphere does not contain the origin then there is no self-intersection
- If the convex hull of the control points of $N(u,v)$ does not contain the origin there cannot be self-intersection. Otherwise, recursively split as for collision detection.

Cloth simulation

[Volino et al. 1995]



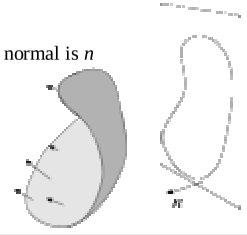
- **Problem statement:** the cloth is modeled as a triangulated surface
- **Self-intersection**: the same principle as the pseudo normal patch, but using the triangles' normals (no analytical description available)
- **If** there is a vector n such that, for each v: $v_i \cdot n > 0$

  and

  the 2D projection of the contour onto the plane whose normal is n does not self intersect

  then

  there is no self-intersection

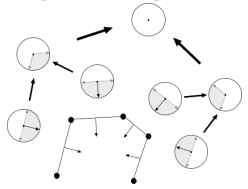


Cloth simulation

[Volino et al. 1995]

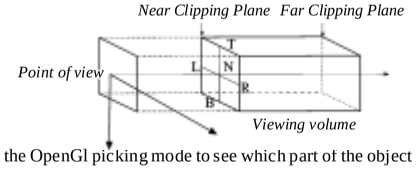
- A hierarchy of axis aligned bounding boxes is used for collision detection. Each bounding box relates to a region R
- The same hierarchy is used for self-intersection. Each node stores the space of all the vectors V such that: $v \in V \Rightarrow v \cdot n_j > 0 \ \forall n_j \in R$

  "If V is not empty there cannot be closed-loop self-intersection"
- Computation of V in the continuum space would be too expensive. The space of the normals is discretised to a few points instead.

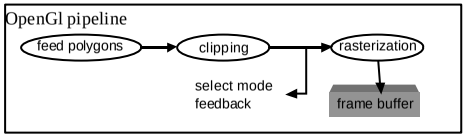


CD between a solid and a deformable object
[Lombardo et al 1999]

- **Problem statement:** to detect the collision between a rigid object and a soup of polygons
- Hardware-based solution: using a viewing volume as a bounding box for the rigid object

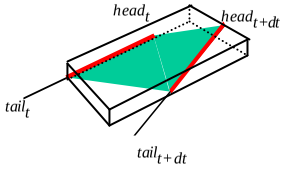


- Using the OpenGl picking mode to see which part of the object would be rasterized, i.e. which polygons intersect the viewing volume



CD between a solid and a deformable object
[Lombardo et al 1999]

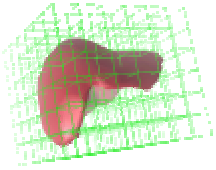
- The viewing volume can be used as a space-time bound.



- Viewing volume can be combined with *clipping planes* to obtain tighter bounding volumes
- It is simple to implement and very efficient for collision between one simple rigid object and a deformable object.
- It requires drawing all the polygons in the scene. If no other technique is used for the broad phase collision detection, the performance can drastically worsened.

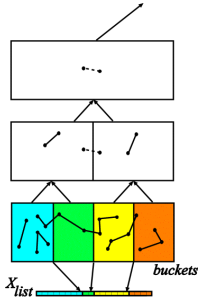
CD between a solid and a deformable object
[Cotin et al 1998]

- **Problem statement:** to detect the collision between a rigid object and a deformable object represented as a triangular surface
- **Assumption:** the rigid object is discretized with a set of points
- The bounding box of the object is partitioned into a grid of *bucket*s and each bucket is associated with the list of vertices included in it
- For each point of the rigid object, the nearest triangle is found among the ones sharing the nearest vertices
- **Further assumption**: The nearest triangle is incident in the nearest vertices.



Bucket-tree
[Ganovelli et al. 2000]

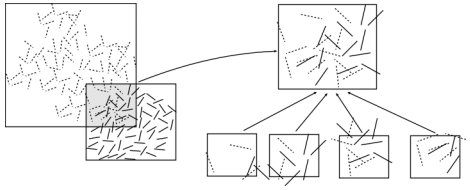
- **Problem statement:** an object is a soup of polygons freely moving
- The bounding box of the object is partitioned in a grid of $n_x \cdot n_y \cdot n_z$ buckets where $n_i = 8^k$ for some k
- A complete oct-tree with k levels is built on the grid of buckets (a bucket is a leaf of the tree)
- The buckets contain the vertices of the polygons
- The nodes of the octree contain the list of the primitives whose vertices are all in the corresponding box



Building hierarchies on-the-fly

[Kitamura et al.1998]

- **Problem statement:** an object is a soup of polygons freely moving
- An oct-tree is built only when two or more bounding boxes overlap.
  - The oct-tree is built top-down over polygons included in the overlapping regions.
  - The root of the oct-tree is a box including all the scene



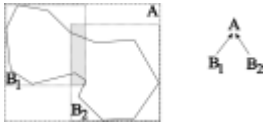
- Note: so far, this approach is the only one naturally supporting cuts
- The authors proved experimentally that this approach is faster than GJK and I-COLLIDE for convex rigid objects

AABB Trees

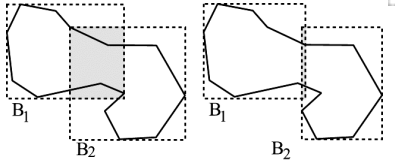
[Van Den Bergen 1998]

- **Problem statement**: the surface of an object is tessellated with polygons
- The hierarchy of boxes can be quickly updated using the following property: let $Sm(R)$ be the *smallest* AABB of a region R and r_1, r_2 two regions. Then:

$$Sm(Sm(r_1) \cup Sm(r_2)) = Sm(r_1 \cup r_2)$$



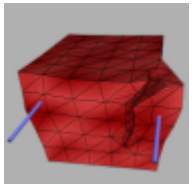
- …so the hierarchy is updated in $O(n)$ time
- Note: this is not the same as rebuilding the hierarchy



Refitted *Rebuilt*

Conclusions

- Deformable objects play a fundamental role in many contexts
- Some solutions to the collision detection problem have been found in specific contexts
- Most of them need some assumptions on the range of deformations, mesh connectivity or step size
- Self-collisions are likely to happen in the presence of cuts and/or lacerations. None of the reviewed approaches can handle this situation.



Collision Response

John Dingliana
Image Synthesis Group, TCD
John.Dingliana@cs.tcd.ie

Overview

- The collision response problem
- Simulation discontinuities and non-interpentration
- Fast response for Interactive Animation
- Calculating collision responses
- Approximate collision response
- Contact modelling
- Scheduling issues

Collision Response

- Collision Response [Moore and Wilhelms 88]: *"automatic suggestions about the motions of objects immediately following a collision; animation systems using dynamical simulation inherently must respond to collisions automatically and realistically"*
- Contact information alone is very useful for interaction, picking, collision avoidance etc.
- In self-regulating dynamic simulations (e.g. virtual environments) collisions are the basis for interaction between different objects, between users and objects

Problems of Collision Response

- In Physically Based Animation, objects need to behave in ways similar to their real-world counterparts
  - We need to ensure that object behave according to believable dynamics
  - We need to enforce non-interpenetration
- Collisions are a source of unpredicatable **simulation discontinuities**. As a result they create additional workload for the simulation mechanism even after the expensive process of collision detection.

Rigid Body Dynamics

- All objects are associated with a set of constants and state variables.

RigidBody

$$\mathbf{Y}(t) = \left\{ \begin{array}{l} \text{Mass} = \boldsymbol{M} \\ \text{InertialTensor} = \boldsymbol{I}_{\text{BODY}} \\ \text{Position} = \boldsymbol{x(t)} \\ \text{Orientation} = \boldsymbol{R(t)} \\ \text{LinearMomentum} = \boldsymbol{P(t)} \\ \text{AngularMomentum} = \boldsymbol{L(t)} \end{array} \right.$$

STATE VARS

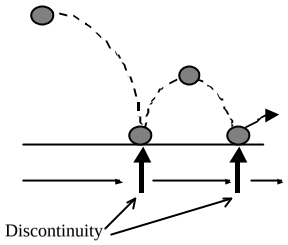
Simulation involves finding the change of **Y** over time:

$$\frac{d}{dt}\mathbf{Y}(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \omega\,(t) * R(t) \\ F(t) \\ \tau\,(t) \end{pmatrix}$$

for details, see [Baraff and Witkin 97]

Rigid Body Collisions

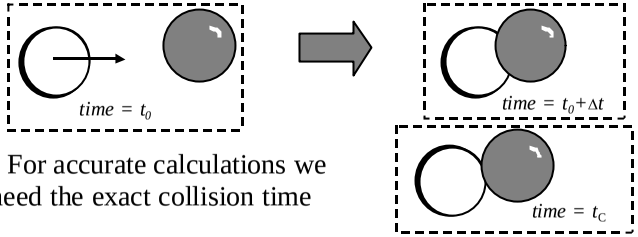
- Collisions are a source of discontinuity in the simulation process
- Particularly in the case of rigid bodies we need to compute an (instantaneous) change in state at the time of collision to prevent colliding objects from penetrating



Steady simulation is interrupted by discontinuity due to collision

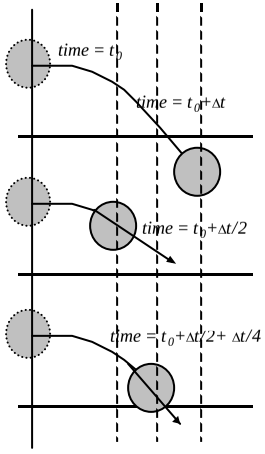
Non-Interpenetration

- Unfortunately in computer animation we are dealing with discrete timesteps. So the instant of collision (or point of discontinuity) is not obvious.
- In most cases collision detection returns the fact that object have interpenetrated at time $t_0+\Delta t$



• For accurate calculations we need the exact collision time

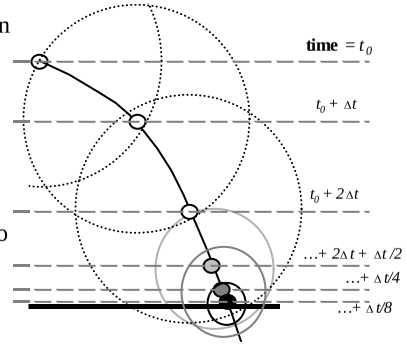
Timestep Bisection



- One simple solution is to step backwards in simulation time and redo collision detection
- A useful (if not reasonable) guess might be $t_0+\Delta t/2$
- This process is repeated until we achieve a result under some threshold accuracy $\in$
- Some approaches attempt to calculate t_C

Adaptive Timesteps

- Reduces wasted computation encountered in bisection by pre-emptively reducing timesteps
- A conservative approximation is taken as to when discontinuity occurs and simulation takes place to this point
- As discontiuity gets closer increasingly small steps are taken



Timestep is halved when the spacetime-bound suggests that the object will interpenetrate in the next step

Timewarp

- [Mirtich 2000] Main problem with previous methods are that they deal with the synchronous evolution of the entire scene
- Asynchronous processing of individual interactions can greatly improve performance
- Basic idea is to rollback to time of discontinuity but only for relevant objects that have penetrated and for objects dependent on these
- Similar to having several different simulation processes communicating at key points (collisions)

Collision Response in Virtual Environments

- Primary concern in interactive virtual environments is speed. We must guarantee high and constant frame rates.
- Many of the “accurate” solutions become infeasible for populated scenes of unpredictable complexity.
- Simplification is a necessity for any target framerate.
- Different approaches adopted different simplifying assumptions.
- Fortunately many simplifications may go unnoticed in an interactive simulation.

Plausible vs. Accurate Response

- [Barzel et al 1996] “what is really of interest is *plausible* motion which is somewhat different (from *accurate* motion)”
- Users are often unable to correctly predict resulting states after events such as collisions
- We can exploit this uncertainty in order to reduce simulation workload
- Using approximate solutions (in place of mathematically consistent solutions) may introduce randomness which can infact enhance the plausibility of certain simulations

Dynamics Culling

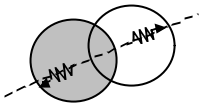
- One approach for reducing workload in virtual environments is by culling the non-visible
- In simulation however we must consider the consequences as non-visible objects may indirectly or in future frames affect the visible scene. [Chenny and Forsyth 97] We must maintain :
  - Consistency of state when it returns into view
  - Completeness: certain visible objects react with the objects in non-visible regions
  - Causality: certain events in the visible region affect the states of the non-visible (consistency required when this becomes visible)

Simulation Levels of Detail

- An alternative approach to reducing workload is by applying varying simplifications to simulation rules across the scene
- e.g. [Carlson and Hodgins 1997] use a different set of rules for simulating objects “out of view or too far to be seen”
- Simplification across the scene can be based on visibility/perceptibility or similar cost/benefit heuristic [Funkhouser and Sequin 1993]
- Adaptive refinement of collision data [Dingliana and O’Sullivan 2000] can also be used to generate consistent results across the visible scene.

Spring Based Response

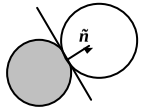
- Places a virtual spring at each collision point to push two objects apart
- Easy to understand and program, applies equally well to rigid and non-rigid bodies
- Numerical effort increases with violence of collision (smaller timesteps required)
- A.K.A. Penalty methods
  - Objects allowed to penetrate
  - Spring force proportional to interpentration



Impulse Based Response

[Mirtich 1998]

- Quick local calculation of instantanious change in velocities is possible at contact points
- Good solution for rigid bodies but it is difficult to model resting contact and static friction



Impulse $J = j\tilde{n}$

causes.. $\Delta \mathbf{P} = \mathbf{J}$

$\Delta L = \tau_{impulse} = (\mathbf{p} - \mathbf{x}) \times J$

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

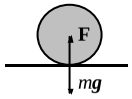
$$\mathbf{r}_i = \mathbf{p} - \mathbf{x}_i$$

$$\mathbf{v}_C = \tilde{\mathbf{n}} \bullet \left((\mathbf{v}_a + (\dot{\mathbf{u}}_a \times \mathbf{r}_a)) - (\mathbf{v}_b + (\dot{\mathbf{u}}_b \times \mathbf{r}_b))\right)$$

for elasticity ϵ, collision point $\boldsymbol{p}$, collision normal $\tilde{\boldsymbol{n}}$

Constraint based methods

- Makes a distinction between contact and collision
- For two objects in contact (relative velocity 0) ensure that the right force is applied to prevent them from interpenetrating



- For objects moving towards each other (negative relative velocities), force applied over time is not enough to prevent interpenetration
- Changing nature of constraints can cause high computational demand

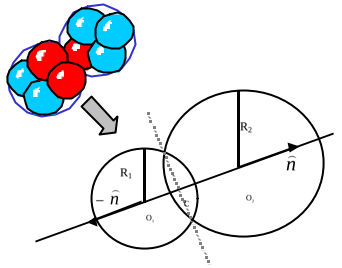
Contact Modelling

- In all of these methods we require:
  - Initial states of colliding objects
  - Contact points or contact manifold
  - Direction to apply impulses, or forces
- We need more than a yes/no input from CD
- Many approaches e.g. [Palmer and Grimsdale 95] always perform narrow phase accurate CD – polygon intersection tests
- Time Critical CD needs results from refinable (pre-narrow phase) collision detection!

Per contact calculations are constant time… How can we introduce L.O.D.?

Contact Approximation in Hierarchical CD

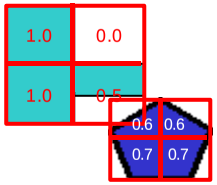
- Hierarchical Models are approximations of an objects volume. As a result calculated contacts can only be approximate.
- **Sphere tree example**: for any colliding nodes we can approximate a collision point ***c***. This applies for cases of interpenetration ($0 < d < \epsilon$) as well as gap ($-\epsilon < d \leq 0$)
- As we go deeper in the hierarchy this approximation becomes increasingly more accurate
- For multiple interpentration we resolve by combination or by approximation of contact order to find a single overall response to the collision



$$\frac{|\mathbf{c} - \mathbf{o}_1|}{|\mathbf{c} - \mathbf{o}_2|} = \frac{R_1}{R_2} \qquad \hat{\mathbf{n}} = \frac{\mathbf{o}_2 - \mathbf{o}_1}{|\mathbf{o}_2 - \mathbf{o}_1|}$$

A Non-deterministic contact model

Volume reps are usually conservative over-estimates of actual object volume we can assign a surface crossing probability (or density metric) to represent the probability of a node interpenetration representing an actual collision.



For each node collision we have:

Surface crossing probability s_i

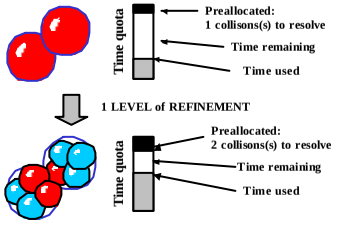
Measure of *Interpenetration* p

Collision probability $C(s_1, s_2, p_{12})$

The easiest approach is where $s_i = 1$, we model behaviours of the bounding volumes performing step-back and non-interpenetration as required. However we can use this information to approximate collision normals (similar to some volume rendering approaches)

Scheduling Collision Handling

- Per-collision response calculations are constant time and non-refinable
- However we can inherit refinability and interruptibility directly from the collision detection mechanism



It is possible to approximate (if not predict) the per-collision computation time.

Based on this the scheduler pre-allocates processing time for response.

Collision detection is refined within the time-quota **minus** preallocated response processing time