Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin

Copyright statement

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

Liability statement

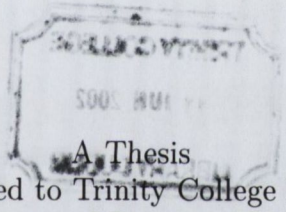
By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

Access Agreement

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Bounding Volume Hierarchies for Level-of-Detail Collision Handling



A Thesis
Submitted to Trinity College Dublin
for the Degree of Doctor of Philosophy

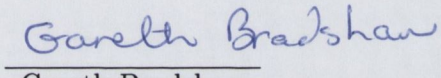
Gareth Bradshaw

May 2002

Declaration

I declare that the work contained within this thesis has not been submitted for a degree at any other university and that the work is entirely my own.

Signature

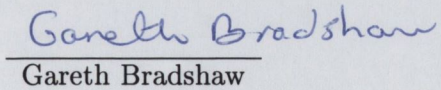


Gareth Bradshaw

Permission to lend and/or copy

I agree that the library in Trinity College may lend or copy this thesis upon request.

Signature



Gareth Bradshaw

Acknowledgements

Firstly, I would like to thank my family and friends for their constant support throughout the course of this work. I would especially like to thank my parents, Brian and Rhoda, who gave me the freedom to choose my own path in life and supported me at every step.

Secondly, thanks to my supervisor, Dr. Carol O'Sullivan, for her encouragement, inspiration and extensive use of red pen.

I must also thank everyone who gave their input on this Thesis and its contents. Special thanks go to those who had the endurance to read it in its entirety, Carol, Robert, Helen, Sinéad and Brian, and to Clodagh for her help preparing the final copy.

Abstract

Enforcing solidity of objects within simulations is a major computational overhead. Detecting interactions between bodies is a large part of this overhead. Many researchers have used hybrid collision detection algorithms to address this issue. Such algorithms use multiple phases, first to eliminate object pairs that cannot be interacting and then to narrow in on the regions of the objects that are in contact.

The second phase of these algorithms, the narrow phase, typically uses a hierarchical representation of the objects. A tree traversal algorithm narrows in on the regions of contact. While many different geometric primitives have been used for these hierarchies, spheres have some distinct advantages, especially for interruptible collision detection systems. However, large numbers of spheres are often required to approximate the objects' geometries.

Octrees and medial axis techniques have been utilised for the construction of hierarchies of spheres, known as sphere-trees. This thesis presents a number of improvements to both these techniques. Existing methods have been critically analysed to determine their strengths and weaknesses and new algorithms, which build upon them, have been developed. As the sphere-trees are intended for use in an interruptible collision detection system, which uses the spheres to approximate the collisions as accurately as it can within an allowable time-slice, the main focus is on the close approximation of the object and fast traversal of the hierarchies.

The main contributions of this thesis include the development of an adaptive medial axis approximation algorithm that allows areas of the medial axis to be constructed as required. This allows a finer approximation of the medial axis in detailed areas and allows more detail to be added to the approximation when there is insufficient information to closely approximate the object. A sphere-tree optimisation algorithm, which further improves the tightness of the representation, is also presented. An optimiser based algorithm, which constructs sphere-trees similar to those generated from the medial axis without requiring its approximation, is detailed. Finally sphere-trees generated with the various algorithms are compared in an interruptible collision detection system.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Solid objects collide, they do not *ghost* through each other. No matter how hard one tries, it is not possible to push one solid object through another, unless of course, one of the objects pierces the surface of the other[1]. Thus it can be said that if the objects interfere with each other, this interference is in the form of a *collision.* Determining when and where these interactions occur is called *Collision Detection* and the effect caused by the interaction is called a *Collision Response.* Collision detection is a problem of kinematics, i.e. relating the motions and positions of objects within an environment, whereas collision response is a problem of dynamics, i.e. applying the "laws of physics" to simulate the resulting changes in the motions of the objects [76].

The problems of collision detection and response are fundamental to computer graphics and simulation of physical situations. A typical example would be a computer animation that features an avalanche or a rock fall. While traditional animation relies largely on human animators to enforce the solidness of objects, this is impractical for large interactive systems. Also the dynamic nature of the environment, and hence the collisions that occur within it, mean that the collisions are not predetermined. Thus a *Collision Handling* system is necessary to manage the large numbers of collisions.

Collision detection and response are not only fundamental to graphics and simulations, but also to other areas. Some of these areas include :

- **Virtual Prototyping.** Having used a CAD/CAM system to design a complicated mechanical system, a simulation can be used to detect design problems that would result in components hitting off each other or not interacting correctly. This reduces manufacturing costs by reducing the amount of effort that needs to be put into the construction and testing of physical prototypes. Having evaluated the performance of the computer model it is then possible to manufacture the system knowing exactly

[1]There is a scientific theory that states that if a pair of quantum particles collide often enough, the probability of tunnelling occurring becomes statistically significant. However there is no evidence that this could happen for objects with parameters outside of quantum distances.

how the parts will interact. Held *et al.* explore techniques for working with massive models such as industrial plants [42].

- **Collision Avoidance and Path Planning.** Collision avoidance systems often contain a collision detection phase. In this phase the path of the robot (or robotic manipulator) will be extrapolated into the future. Having detected an imminent collision the system can then plan avoidance manoeuvres or warn the operator [8, 31, 38, 95, 96, 101].

- **Haptic Rendering.** In order to provide force feedback during simulated interaction, such as virtual surgery and painting, it is necessary to be able to determine the interaction between the virtual environment and the probe/manipulator, which is under the control of the user. In order to provide quality feedback to the user, high processing speeds are necessary [11, 35, 36, 37, 69, 71, 77].

- **Virtual Crash Testing.** Transport vehicles are usually required to undergo thorough crash test procedures. This is an expensive operation. The use of computer simulations can greatly reduce the number of real tests that need to be conducted in the early phases of testing[2]. The information gathered during these simulations can then help designers to improve safety. Many of the large vehicle manufacturers have started using virtual crash testing as part of their design process[3].

- **Molecular Modelling.** Modelling and visualisation of complex chemical structures and their interactions allow chemists to investigate their molecular structure and actions in a virtual environment. Collision detection plays a vital role in determining how chemical compounds combine and in providing force feedback to the operator [52, 102].

Thus, there are many different areas where collision handling systems are necessary. For areas such as Virtual Prototyping and Crash Testing the emphasis is on precise detection and response. As these do not require real-time performance the computation can be performed to provide precision down to the level of the computer model. While a lot of work aims to increase performance, for these areas it is acceptable to spend the order of minutes (or even hours) performing the simulations. However, for real-time animations and virtual surgery (haptic rendering) it is necessary for the collision handling system

[2] An example of such a system is PAM-Crash/Safe from Pennsylvania Transportation Institute, The Pennsylvania State University (www.vss.psu.edu).

[3] Some of the companies using virtual crash testing include : DaimlerChrysler, Mercedes-Benz and Renault.

to operate interactively. The next section discusses requirements for interactive collision detection and introduces the idea of interruptible collision detection, which trades speed for accuracy in order to maintain interactivity.

1.1 Requirements

There are many techniques available for increasing the speed at which a scene can be rendered. Graphics workstations and PC-3D accelerators provide fast rendering of scenes constructed from triangles. Also, dynamic level-of-detail techniques, such as progressive meshes [44, 45, 46], quadric surface simplification [41] and even run-time simplification [112] allow the complexity of the scenes to be reduced, see [22, 88] for a comparison of various LOD packages. However, none of this aids the provision of real-time collision handling. As the number of objects in the simulation increases, any additional processing power, such as that available by adding additional CPUs, will soon be used up.

Achieving interactive frame rates is critical in a computer animation and interactive simulation. In order for the animation to be reasonably believable we would require at least 10 frames per second (fps)[4]. It is also a requirement that the frames be generated at fixed regular intervals, i.e. there needs to be a low variance in the time taken to produce each frame. Failure to produce consistent frame rates is thought to cause of a kind of motion sickness called *simulator sickness* [43]. Another requirement of a collision handling system is the ability to handle the arbitrary motions that result from user guidance, such as manoeuvring a jet, and the complex interactions involving many objects.

Throughout the lifetime of an animation the amount of work required in both rendering and collision handling will vary. Take for example an animation of a rock fall where thousands of fragments are involved. As the pieces of rock start to break away from the rock face there are relatively few collisions involved. However as the rock face continues to deteriorate the rocks begin to pile up at the bottom of the ravine. As this pile of rubble grows, so does the number of collisions that occur. This increases the amount of work required to resolve the collisions, which in turn reduces the achievable frame-rate. If we wish to produce a real-time animation of this scenario we have to provide enough computing power to handle the worst case.

Interruptible[5] collision detection, introduced in [47] and [49], aims to achieve these goals in spite of potentially complex scenarios. The interruptible algorithm takes a level-of-detail approach to collision detection and resolves the collisions as accurately as it can within a given time-slice. Thus, it aims to trade accuracy for computational cost. As the computational requirements of the collision handling increases, the accuracy of the resulting collisions decreases in order to maintain a consistent frame-rate. To achieve this,

[4]By today's standards this value is very low, most would expect 60fps or higher.

[5]The term interruptible is used as the algorithm does as much processing as it can before it runs out of time, i.e. before the collision detection process is interrupted.

objects are represented by a hierarchy of successively finer approximations, such as those shown in Figure 1.1[6]. These hierarchies are traversed in a breadth-first manner, which incrementally improves the approximation of the collision information.

As the number and complexity of the objects increases, the accuracy at which the collisions can be approximated, before interruption occurs, is reduced. This allows the simulation to degrade gracefully, while giving it a good chance of maintaining its desired frame-rate. With a traditional (non time-critical) algorithm the increase in scene complexity would result in higher processing times. Hence the animation would start to slow down and become jerky. The inaccuracies resulting from the interruption of the algorithm are often preferred to slow jerky animations.

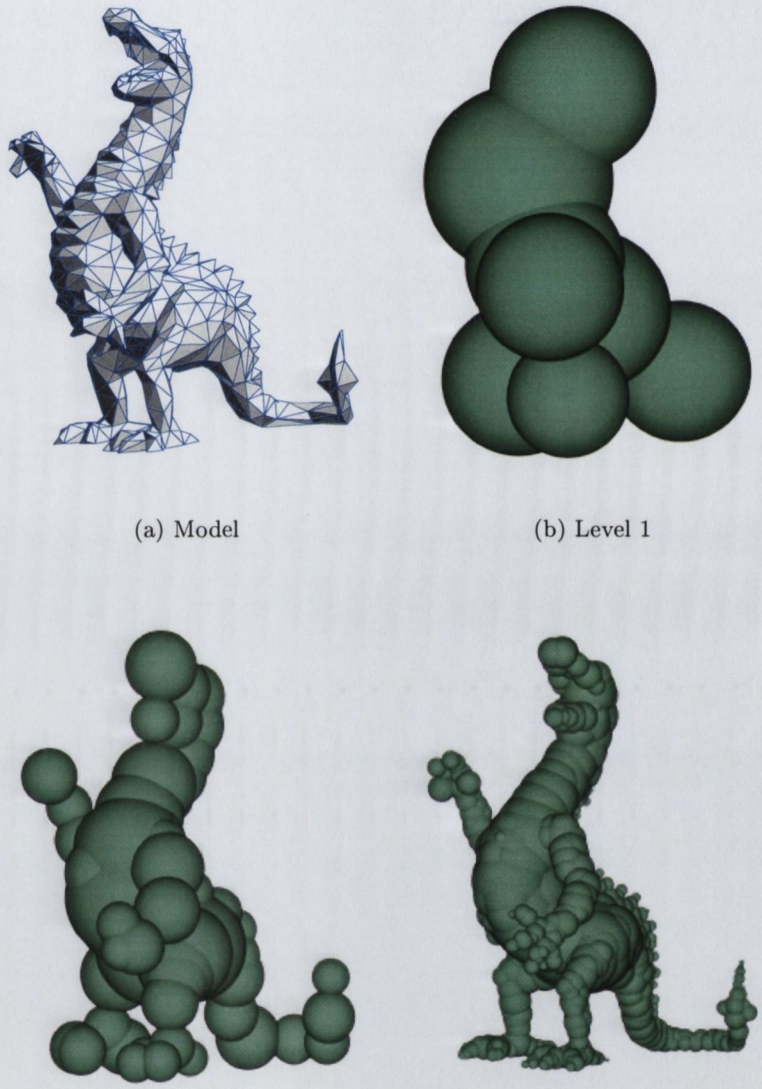
There are two effects that can result from prematurely interrupting the collision detection algorithm. If the system ignores unresolved collisions the objects can float into each other, this is known as *inter-penetration*, see Figure 1.2(a). However, if the system chooses to treat such collisions as definite collisions then the objects can be seen to repulse each other, see Figure 1.2(b). Often repulsion is a more favourable artifact than inter-penetration [81], especially when using the collision detection for planning and avoidance in robotic environments [113]. Repulsion also maintains separation between the objects, whereas allowing them to interpenetrate could result in objects entering each other and then not being able to leave due to a change in the level of approximation being used for collision detection (popping).

1.2 Focus of this Thesis

This thesis aims to examine the requirements of the interruptible collision detection algorithm. Specifically, it addresses the construction of tight fitting sphere-trees to provide a higher quality of approximation and examines the requirements of using these approximations for collision response. The main contributions are as follows :

- An adaptive medial approximation technique that allows the medial axis to be constructed on demand and focuses on improving the approximation in areas where the spheres created from the medial axis ill-fit the object.
- Improved sphere reduction techniques that reduce the spheres generated from the medial axis into a manageable set while maintaining a high degree of fit and consistency.
- A generic sphere-tree construction algorithm that decomposes the problem into smaller object approximation problems, which can then be solved using the algorithms presented. When dividing the object into sub-regions, the algorithm min-

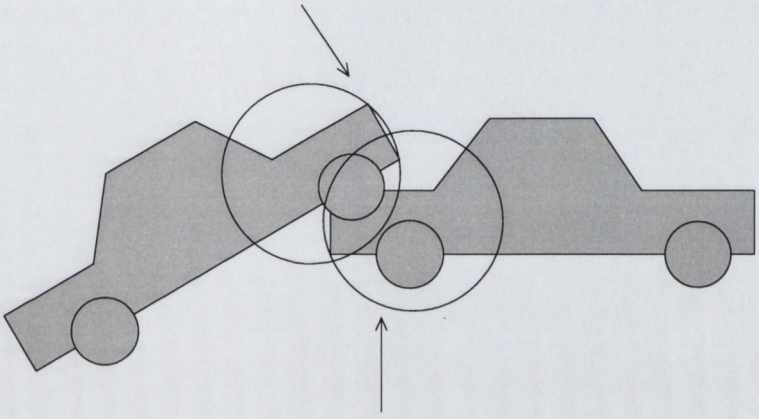
[6]Data from http://graphics.cs.uiuc.edu/~garland/research/quadrics.html



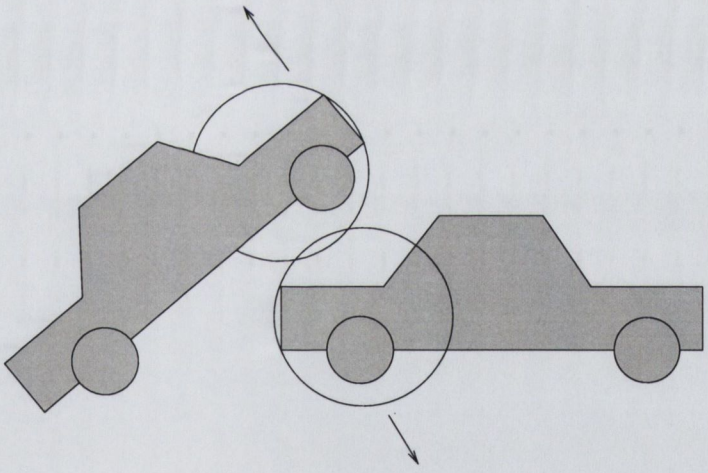
(a) Model (b) Level 1

(c) Level 2 (d) Level 3

Figure 1.1: Example of a dragon approximated with 3 levels of spheres.



(a) Penetration



(b) Repulsion

Figure 1.2: The two cases resulting from premature termination (interruption) of the collision detection algorithm.

imises the amount of overlap between neighbouring regions so as to eliminate redundancy within the sphere-trees.

- A sphere-tree optimisation algorithm that further improves the degree of fit in the approximations.
- Additional approximation algorithms based on generalising the octree sub-division to produce tighter approximations, by allowing more freedom in the way spheres are placed, while maintaining its good sub-division characteristics.
- A purely optimisation based approximation algorithm that yields similar results to those obtained using the medial axis but does not require the overhead of constructing the medial axis approximation.
- The introduction of a hybrid sphere-tree structure that contains both minimum error and minimum volume spheres at each node. These sphere-trees provide for both spatial localisation and object approximation.

The remainder of this thesis is structured as follows: Chapter 2 reviews the current state of the art in collision detection and presents a sphere-tree traversal algorithm; Chapter 3 presents the simplest of the sphere-tree construction algorithms, the Octree Method. This method is critically analysed and a new algorithm derived. The resulting algorithm approximates the object more closely without reducing the spatial sub-division properties of the hierarchy; Chapter 4 details a more sophisticated sphere-tree construction algorithm, the Medial Axis Method; Chapter 5 analyses this method and presents a number of improvements for the approximation of the medial axis and the generation of sphere sets from that approximation; Chapter 6 details the generic sphere-tree construction algorithm and the sphere-tree optimisation algorithms; Chapter 7 compares the different algorithms in terms of object approximation and sphere-tree construction. The resulting sphere-trees are then evaluated in an interruptible collision detection system; Chapter 8 presents the conclusions and comments on further areas of research that could result from this thesis.

Chapter 2

State Of The Art

Collision handling is an area of computer graphics that has been receiving much attention from researchers. Collision handling consists of several areas, including collision detection, contact modelling and collision response. These in turn can be further divided into sub-areas. This chapter reviews some of the work undertaken in the area of collision detection, and visits some of the related areas such as collision response and distance computation. While many of the algorithms assume rigid (or articulated) objects, usually represented by a polyhedral model, some are applicable to higher order models, such as those for deformable objects and time dependent parametric surfaces. Lin and Gottschalk present a taxonomy for 3D model representations, including polygon meshes, composite solid geometry (CSG), implicit and parametric surfaces. They also present many ideas for performing collision tests between these different representations [68].

2.1 Basic Collision Detection Algorithm

Collision handling systems are often used within simulations. Within these simulations a number of objects are undergoing motion. The progress through the simulation is represented by an abstract time, known as *simulator time*. As the simulation advances the simulation time increases. As time goes by two main activities are taking place; frames are rendered every δ_r time units and the motions of the objects are updated every δ_s time units. Thus the basic simulation can be considered to be made up of a number of steps :

1. Update each object's position and orientation for the current time-step,
2. Update each object's velocity (angular and linear) according to its acceleration,
3. Compute the collisions between the objects in their new position,
4. Determine the changes to object velocities resulting from the collisions,
5. Render the frame.

There are, however, a number of problems associated with trying to achieve the high consistent frame rates required for performing interactive simulations and animations. These are as follows [47] :

- *Fixed-Timestep Weakness.* This weakness arises from the discrete intervals at which simulations sample time. The system maintains a simulation time, which it steps through. As the animation progresses, a frame is rendered every δ_r units in simulator time, and the simulation is updated every δ_s units. A larger δ_s increases the efficiency of simulating the collisions, but increases the chance of missing a collision, especially when objects are travelling at high speeds. The inability of an algorithm to dynamically adjust its simulation time-step is called the fixed-timestep weakness.

- *All-Pairs Weakness.* When an algorithm is performing collision tests on a scene containing N objects, there are potentially $O(N^2)$ pairs of objects that could be in contact. Therefore the potential number of *colliding pairs* increases quadratically with the number of objects in the scene. If an algorithm needs to perform a collision test for each pair of objects it is said to suffer from the all-pairs weakness.

- *Pair-Processing Weakness.* Some algorithms have difficulty dealing with troublesome circumstances. For example, the Lin-Canny algorithm exhibits cycling behaviour when one body is actually penetrating another, thus requiring one to limit the number of iterations it is allowed to perform. An algorithm's shortcomings, in terms of robustness and efficiency, when resolving a collision between two objects, are collectively referred to as the pair-processing weakness.

Some algorithms address the fixed-timestep weakness through the use of 4D geometry [12], *space-time* bounds [47, 49] and time-bounds for parametric surfaces [107]. Mirtich [74] adapted Jefferson's *timewarp* [54] algorithm, for performing optimistic synchronisation in multi-processor systems, to the arena of collision handling. In his system each object maintains its own simulation time, which is only stepped back when a collision is found to invalidate the object's state. This differs from retroactive detection in which all objects are stepped back when a collision occurs and conservative advancement which advances all objects up to the first discontinuity (i.e. collision), which is conservatively approximated. Such algorithms are beyond the scope of this thesis and more information can be found in the cited papers.

Many researchers have tackled both the all-pairs and pair-processing weaknesses by utilising a multi-phase (hybrid) algorithm for collision detection. The algorithm can be considered to have two or three phases. The broad phase efficiently reduces the number of the potentially colliding pairs by eliminating objects that are obviously too far away from

each other to be in contact. When two objects are found to be potential colliders a narrow phase algorithm is triggered. Narrow phase algorithms, of which there are many, often use spatial localisation techniques, such as *Bounding Volume Hierarchies* (*BVH*), to reduce the areas of the objects that need to be considered. The final phase, the exact phase, uses the results of the narrow phase algorithm to perform accurate collision detection between the objects[1]. The specific nature of each of these phases, and how they fit together, depend on the representations used for the model. The following sections review many of the strategies adopted for each of these phases.

2.2 Broad Phase Algorithms

The first phase of the hybrid collision detection algorithm is the broad phase. This aims to quickly eliminate pairs of objects that cannot possibly be in contact, thus combating the all-pairs weakness. This is very important as the number of potential pairs of objects increases quadratically as the number of objects in the scene increases. A number of strategies have been adopted, which are outlined below.

2.2.1 Sweep and Prune

Cohen *et al.* [17] and Lin *et al.* [70] present a system called I-COLLIDE, which uses a sort-based pruning algorithm for its broad phase processing. This relies on the observation that if two axis aligned bounding boxes intersect, there is an overlap between their projections onto the basis axes, i.e. the X, Y and Z axes. *Inter-frame coherence* is exploited to achieve near linear performance by maintaining the projections in ordered lists. As objects move a limited distance between frames, the interval lists remain almost sorted. This allows them to be updated in near linear time using *Insertion Sort.*

The system performs exact interactive collision detection for large numbers of objects. The exact phase of the algorithm uses Voronoi regions, to perform polytope intersections. These Voronoi regions represent the areas around the polytope that are closest to each face, edge and vertex. In [83], Ponamgi *et al.* present a hierarchical version of the "sweep and prune" algorithm, where each level represents a tighter fitting set of bounding boxes, not unlike the AABB-trees to be described in Section 2.3.2.

2.2.2 Overlap Table

Wilson *et al.* [111] consider the storage implications of using massive models, i.e. the memory overhead required for large CAD models. The scene is composed of objects and a pre-computed *overlap graph*, which represents the proximity of these objects. The collision

[1]Many researchers consider the narrow phase and the exact phase to be a combined phase. In this thesis they are treated as two separate phases as this allows interruptible collision detection to be easily distinguished from exact collision detection.

detection is performed while traversing the graph. This requires only a few sub-sections of the scene to be loaded into memory at any one time.

Palmer and Grimsdale [82] use a similar scheme. Their broad phase algorithm localises objects that might collide, using a *global bounding volume table* where each object is represented by a bounding sphere. When objects are updated they are tested for collisions using sphere-trees and exact polygon intersection tests.

2.2.3 4^{th} Dimension

Similarly to Hubbard's *space-time* bounds [49], Cameron approaches collision detection as a 4D problem, with the 4^{th} dimension being time [12]. As the simulation progresses, the objects move through space creating a 4D shape, i.e. the object is swept through some period of simulator time. The collisions are then detected by looking for intersections between these 4D representations. However, object rotations and non-linear motions are problematic cases for the algorithm because of the complicated shapes created by rotating bodies in 4D.

2.3 Narrow Phase Algorithms

Researchers have explored many different structures for narrowing in on the areas of contact, i.e. spatial localisation, during the narrow phase of the collision detection algorithm. Researchers have used a variety of different hierarchical structures, known as Bounding Volume Hierarchies (BVHs) to achieve the required spatial sub-division. Some important requirements for a good BVH include :

- the hierarchy conservatively approximates the volume of the object, each level representing a tighter fit than its parent,
- for any node in the hierarchy, its children should cover the parts of the object covered by the parent node,
- the hierarchy should be created in a predictable automatic manner, not requiring user interaction,
- the bounding volumes within the hierarchy should fit the original model as tightly as possible, representing the original model to a high degree of accuracy.

Each BVH can be evaluated using a number of criteria. For interactive simulations the emphasis is on achieving high and consistent frame-rates. Therefore a major concern is how well the hierarchy facilitates this goal. For an interruptible collision detection algorithm the narrow phase algorithm may not fully resolve the collision. Thus the approximate collision information, available from the BVH, needs to approximate the points (and types) of

contact (contact modelling) and the resulting response at every level of the approximation [18].

In order to perform a theoretical comparison of the various representations it is necessary to be able to evaluate their performance within the collision detection system. The following equation has been used in [33, 57, 58, 61, 104] to evaluate various types of bounding volume hierarchies :

$$T = N_u \times C_u + N_v \times C_v \tag{2.1}$$

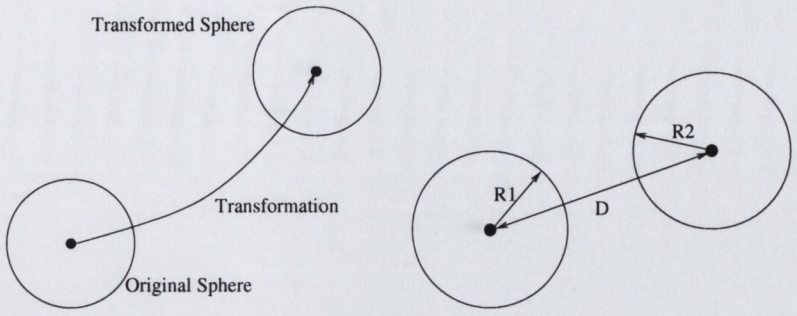
where :

T	is the total cost function for detecting interference between a pair of objects represented by bounding volume hierarchies,
N_u	is the number of primitives, i.e. bounding volumes, that are updated during the traversal of the hierarchies,
C_u	is the cost of updating a primitive due to an object's motion,
N_v	is the number of overlap tests that are performed,
C_v	is the cost of performing an overlap test between a pair of primitives/nodes from the hierarchies.

There is often a trade-off between the complexity of performing updates (or overlap tests) and the number of primitives required to approximate the object's geometry. As the complexity of the primitives in the BVH increases so do the values of C_u and C_v. However, such primitives usually have a higher number of degrees-of-freedom and are more flexible when trying to form an approximation. This generally reduces the number of nodes required to achieve a given level of fit. Hence for more complicated primitives the values of N_u and N_v are lower. There is also a relationship between these values. Each primitive is updated at most once per time-step (obviously we only update the nodes that are tested for overlap) whereas they are involved in multiple overlap tests. Thus an upper bound for N_u is N_v, i.e. $N_u \leq N_v$.

2.3.1 Sphere-Trees

The simplest of all bounding volume primitives is the sphere. Many researchers have used *Sphere-Trees* as their BVH including Hubbard [47, 48, 49], O'Sullivan and Dingliana [79], Quinlan [86] and Palmer and Grimsdale [82]. As a sphere is rotationally invariant, the update step involves a simple transformation of the spheres' center points, Figure 2.1(a) illustrates this in 2D. In 3D, this requires 12 multiplications and 9 additions (floating point). A pair of spheres overlap (see Figure 2.1(b)) if and only if :



(a) Update (b) Intersection

Figure 2.1: Update and intersection for a sphere (shown in 2D).

$$\sqrt{D_x^2 + D_y^2 + D_z^2} \leq R_1 + R_2 \tag{2.2}$$

where D is the vector between their centers. Obviously, the most expensive part of this overlap test is the square root. However this can easily be removed by squaring both sides of the inequality, to give :

$$D_x^2 + D_y^2 + D_z^2 \leq (R_1 + R_2)^2 \tag{2.3}$$

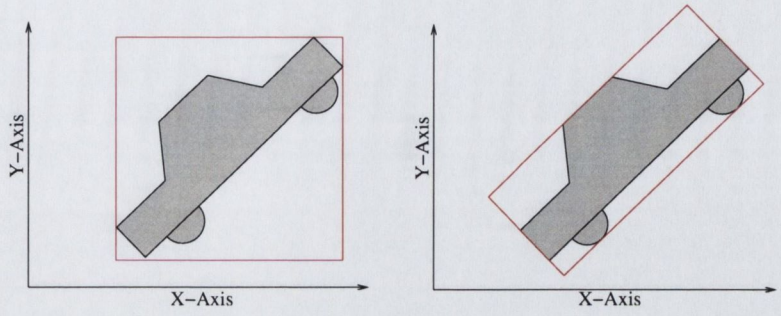
which is much more computationally efficient, requiring only 4 multiplications, 3 additions and 3 subtractions (to evaluate D).

The ease with which a sphere-tree can be updated and tested leads to very low values of C_u and C_v in Equation 2.1. However, complex objects often require a large number of spheres to approximate their geometry, which increases N_u and N_v. The actual arrangement of the spheres is also an important factor in determining the number required for approximation.

The simplest sphere-tree construction algorithm uses an octree, described in [94], to arrange a regular grid of spheres into a hierarchy. O'Sullivan and Dingliana use this scheme for constructing sphere-trees for interruptible collision detection. Hubbard looked at using octrees, simulated annealing and medial axis techniques for constructing sphere-trees. The most promising of these approaches uses the object's *medial axis* as a guide for initial sphere placement. This allows spheres to be placed along the *skeleton* of the object, obtaining a tighter fitting set of spheres.

2.3.2 AABB-Trees and OBB-Trees

Van den Bergen [104] and many other researchers have used *Axis Aligned Bounding Boxes* (*AABBs*) for their bounding volumes. Van den Bergen used the AABB-trees not only for rigid objects but also for deformable ones. The nodes of the hierarchy are boxes whose



(a) AABB (b) OBB

Figure 2.2: 2D Axis Aligned Bounding Box (AABB) and 2D Oriented Bounding Box (OBB) surrounding an entire object.

orientations are fixed to align with the axes of the enclosed object's co-ordinate frame. Others such as Gottschalk *et al.* [32, 33] and Krishnan *et al.* [60] allow the bounding boxes to take on any orientation thus becoming *Oriented Bounding Boxes* (*OBBs*). Figure 2.2 shows 2D representations of both types of primitive.

As is evident from the illustration, OBBs are less restricted in the way they can be arranged and therefore can provide a closer fit around each section of the model than AABBs. The OBB-tree provides quicker convergence to the underlying geometry of the object than the AABB-tree [61]. Thus the AABB-tree generally requires more primitives to achieve the same degree of fit as an OBB-tree, which has the effect of increasing the terms N_u and N_v in Equation 2.1. When objects represented by AABB-trees are undergoing rotation, there is a relative orientation between their local co-ordinate frames. Thus the bounding boxes are no longer axis aligned and need to be treated as if they were OBB's, although some simplifications are possible [104].

A simpler test for overlap between two AABBs is to construct new AABBs enclosing the rotated versions of the original ones. This then requires only 3 interval overlap tests, one on each axis, to test for contact and so results in a vast reduction in C_v at the expense of C_u and the tightness of fit.

The simplest algorithm for performing an intersection test between a pair of OBBs is to test if any of the edges from one OBB intersect a face on the other (or if one OBB is entirely within the other). As there are 12 edges and 6 faces on each OBB, this requires $2 * 12 * 6 = 144$ edge-face combinations to be tested for intersection. This would naturally be quite expensive (in terms of C_v). In the *RAPID* collision detection system, the theory of *Separating Axes* is used to provide a faster test (*SAT*) [33, 70]. This states that :

> "A line $\vec{L}$ is a separating axis if and only if the perpendicular projections, of two convex polytopes, onto $\vec{L}$ are disjoint i.e. the intervals formed by the projections do not overlap. If $\vec{L}$ is a separating axis then there exists a plane,

> orthogonal to it, which is a separating plane. If no separating axis exists, then no separating plane exists, which implies that the polytopes are touching."

From this theory they prove that there is a set of only 15 axes that need to be tested to determine if "two arbitrarily positioned and oriented rectangular boxes in 3-space are in contact". This provides for an efficient collision test, requiring at most 200 arithmetic operations for a pair of OBBs. Although this is expensive when compared to $C_v = 10$ for the sphere, in practice algorithms based on OBB's generally perform better than those using spheres or AABBs [70]. Van den Bergen's empirical results suggest that a simplified SAT can be used if a small (6%) chance of missing a collision can be tolerated. Chung and Wang [15, 16] also use the SAT for convex polytopes.

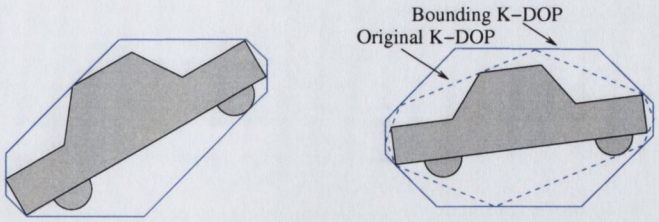
The H-COLLIDE collision detection system utilises OBB-trees for performing collision detection. H-COLLIDE is specifically designed for use in force-feedback systems, which require thousands of updates per second [36] and has been used for interactive modelling and painting [35]. Hudson *et al.* [50] propose the use of OBB-trees (with sweep and prune) to add object-object collision detection to VRML (V-COLLIDE).

Eberly [19, 20] presents a closed-form algorithm for computing a pending intersection between two OBBs under motion. This allows the intersection to be computed over an interval rather than relying on discrete static samples taken at regular time intervals.

Barequet *et al.* [4] present a comparison of BVHs constructed using a number of different box-like primitives, referred to as BOXTREEs. They construct hierarchies of axis aligned boxes, arbitrarily oriented boxes and arbitrarily oriented pie slices (wedges). Their tests show that pie slices, oriented along the shortest principle component of each region, are best at eliminating unnecessary regions of the objects.

2.3.3 Discrete Oriented Polytopes

Discrete Oriented Polytopes (DOP), as described by Klosowski *et al.* [57, 58], aim to provide a low value of C_v while providing tight fitting bounding volumes, i.e. low N_v. Similarly to an AABB, the faces of the bounding polyhedra are aligned with certain axes. They use pairs of orientations that point in the opposite directions to each other. This means that a k-DOP has faces that are normal to $\frac{k}{2}$ axes. Thus the contact test simply consists of $\frac{k}{2}$ interval overlap tests. In fact, an AABB is a 6-DOP as the orientations of the faces are restricted to the orientations of the 3 primary axes (X, Y and Z) in both the positive and negative directions. In order to ensure that the orientation of each bounding plane is limited to the k discrete directions, the boundary must be recomputed as the object rotates. It would obviously be very expensive to compute the *k-DOPs* during the animation. Klosowksi *et al.* pre-compute their k-DOPs and create new ones around them when the object rotates. The resulting k-DOPs are guaranteed to bound the original ones - but with a looser fit (see Figure 2.3).



(a) Original 8-DOP (b) Updated 8-DOP

Figure 2.3: Using 8-DOPs to approximate the boundary of a car.

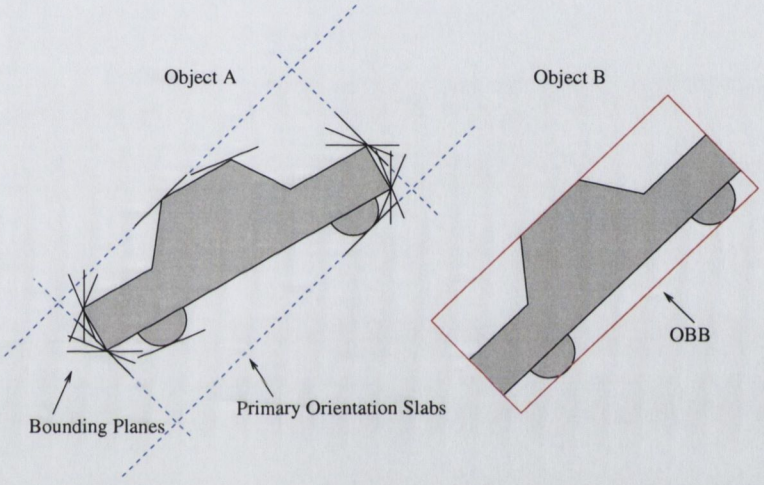


Figure 2.4: Testing for overlap between the primary orientation slabs of an object and the OBB of another.

2.3.4 QuOSPO Trees

He's *Quantised Orientation Slabs with Primary Orientations* [39] combine OBBs with k-DOPS. Each node is represented by an OBB and a large number of planes whose orientations are limited to k discrete orientations, as for k-DOPs but with much larger values for k. When performing overlap tests the bounding box of one object is transformed into the quantisation space of the other object, i.e. the orientation of each plane is limited to the set of allowable orientations. The set of planes that match the OBB's faces most closely are chosen as the principle orientations. Interval overlap tests are then performed between the OBB faces (which have been re-mapped) and the planes for the principle orientations. If the objects are still considered to be in contact, the process is repeated with the roles of the objects reversed. This allows for the tightness of OBBs while potentially reducing the number of axis overlap tests that are needed. This is due to the formation of a pair of bounding boxes that are oriented within a common co-ordinate frame (see Figure 2.4).

2.3.5 Spherical Shells

A *spherical-shell* is the section of a sphere enclosed between two radii and within a solid angle of an orientation vector [61]. While this bounding volume does not share the sphere's rotation invariance, C_u is still quite small. The update requires that the center of the spheres be re-positioned, and the vector be re-oriented (the total being less than twice the value of C_u for a sphere). The cost of performing an overlap test, C_v, is stated to be between 2 and 3 times that of an OBB. However the bounding volumes produced using spherical shells theoretically provide a tighter fit than OBBs. The spherical-shells are reported to exhibit local cubic convergence[2] to the underlying geometry provided it is smooth and of low curvature. OBB exhibit quadratic convergence and AABBs/spheres have only linear convergence. Thus, to approximate a surface to a given degree of accuracy theoretically requires fewer spherical shells than OBBs, AABBs or spheres. This reduces the terms N_v and N_u, N_v being the more important as C_v dominates. In [60] Krishnan combines spherical shells with OBBs to construct *Shell-Trees* when performing collision detection between models constructed using Bézier patches.

2.3.6 Swept Sphere Volumes

A *Swept Sphere Volume* (*SSV*) is the convolution of a sphere with some underlying geometrical shape, i.e. the sphere is swept out across a core primitive. When performing proximity queries, Larsen *et al.* [62] use points, lines and rectangles as the core primitives, each one providing a higher convergence to the underlying model. The nodes in the hierarchy can therefore be a sphere, a cylinder with rounded ends (sphere swept along an arbitrarily oriented line) or a rectangle with rounded edges/corners (a sphere swept across an arbitrarily oriented rectangle).

A rectangle swept sphere provides similar fitting power to that of the OBB, i.e. quadratic convergence to the underlying geometry. A line swept sphere's convergence rate is somewhere between this and the linear convergence achieved with spheres. Obviously, the cost of performing an overlap test depends on the core primitives involved. For long thin areas, line swept spheres provide a good fit, with a moderate C_v, whereas rectangle swept spheres are suited to flat areas and have quite a high C_v. Rectangular swept sphere volumes are used for distance queries within the PQP proximity query package [63].

2.3.7 C-Trees

Youn and Wohn [114] introduced a collision detection hierarchy called a *C-Tree*. They use this structure to provide a boundary representation for objects that contain a conventional kinematic hierarchy, i.e. articulated objects where movement of one part of the body also

[2]Cubic convergence means that the approximation to the surface would be accurate to the second order if the surface can be expressed as a Taylor series [61].

affects the sub-parts. In order to detect collisions between individual articulated sections of an object, the collision algorithm uses a duplicate C-tree treating a single object as two distinct ones. The elements of the tree can be a mixture of spheres and convex polyhedra. However they do not present an algorithm for the automatic construction of such hierarchies. The manual construction of the bounding volume hierarchies makes it difficult to accurately approximate the objects, especially for complicated geometries. The IRIS performer toolkit uses a similar strategy for performing collision handling [91].

2.3.8 S-Bounds

Cameron [12, 14] presents an algorithm for performing collision detection between models constructed using "Composite Solid Geometry" (CSG). This modelling technique uses boolean expressions to construct objects using simple primitive shapes e.g. to create a hole in an object requires a simple boolean difference between the object and a cylinder. He uses bounding functions called *S-bounds* (super-bounds) to express a collision as a non-null region in $A \cap B$ (boolean intersection), where A and B are objects represented by CSG trees. As the use of S-Bounds can return an answer of "don't know", a more definite method needs to be employed when the S-Bounds algorithm does not give a definite answer. He uses spatial sub-division techniques, to further prune away areas of the CSG tree that do not need to be considered, prior to employing a more exhaustive method that uses the geometry resulting from the CSG tree to catch the remaining unanswered queries. Zeiller *et al.* [115, 116, 117] also use S-bounds for performing collision detection between CSG trees.

2.3.9 Voxel Space

Kitamura *et al.* [56] and Smith *et al.* [98] propose voxel based collision detection algorithms that can be used for both rigid and deformable objects. When objects are found to overlap, the required areas of their polyhedral meshes are inserted into a voxel grid using an octree sub-division of the environment's bounding cube. When triangles from different objects are found to occupy the same voxel, an accurate triangle-triangle intersection test is performed. Of course, a triangle may occupy many voxels as illustrated in Figure 2.5. Smith *et al.* [97] present an efficient algorithm for using octrees to update voxelisations of moving objects. Their algorithm uses a compact octree representation to utilise large voxels in areas where sub-division is unnecessary.

Zhang *et al.* [118] utilise voxelisation within their algorithm, in which triangles from one object are checked against edges of the other that share the same voxel. This was used to perform efficient interference detection between clothes and a human figure (including self-intersection of the cloth).

García-Alonso *et al.* [28] compute the overlapping regions between the bounding boxes of a pair of potentially colliding objects. Active voxels from the overlapping regions are

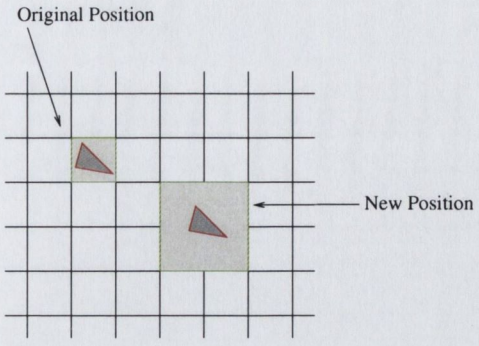


Figure 2.5: As the object moves in voxel space multiple voxels may be occupied when part of the object overlaps the voxel boundaries.

transformed into a common voxel space and accurate tests are performed between the areas of the objects' geometries that occupy the same voxels.

He and Kaufmann [40] also discuss collision detection between volumetric objects. To combat the problems of working in voxel space they use a more traditional bounding volume approach. Each voxel can be represented as an OBB or a sphere. The best case scenario for using a sphere is where it approximates a cubic cluster of voxels, which uses only 37% of the sphere's volume. They present a modified OBB test that exploits the uniform orientation between the boxes and thus is only 2-3 times the cost of a sphere overlap test.

2.4 Interruptible Collision Detection

As mentioned in Section 1.1, consistent high frame-rates are extremely important for real-time simulations. Hubbard's interruptible collision detection algorithm achieves this consistency by utilising a time-critical algorithm that approximates the contact points as well as it can in the allowed time [47, 49]. Thus as the number of collisions increases, the accuracy of the algorithm degrades in order to maintain the frame-rate. He uses *space-time* approximation to provide a conservative estimate of where the object will be in the future. This allows the processing to concentrate on impending collisions and guarantees that collisions will not be missed between frames. He also uses a sphere-tree as a hierarchical geometric approximation to perform efficient collision testing. A list of potentially colliding nodes is maintained throughout the traversal. Initially this list will contain the pairs of root nodes for objects that are potentially colliding. As the node pairs are tested, those that test positive generate new pairs (from their children), which are then added to the list for checking. This process is done in a time-critical fashion and terminates when the allotted time slice has expired. As each successive level of the hierarchy forms a tighter approximations of the object, the accuracy of the collision information improves

as the traversal progresses.

O'Sullivan and Dingliana [78, 79, 81] have adapted Hubbard's work to include Cohen's sweep and prune algorithm [17] for the broad phase. This exploits inter-frame coherence to provide efficient broad phase processing for highly populated scenes. They also replace Hubbard's round-robin breadth-first traversal of the sphere-trees with prioritised scheduling algorithms. This allows important collisions to be resolved to a higher degree of accuracy, making better use of the available run-time. O'Sullivan [78] discusses a model for human perception and introduces a metric for evaluating the importance of a collision. This essentially allows the narrow phase processing to be scheduled in such a way as to improve the visual plausibility of the animation. Dingliana and O'Sullivan [18] address the problem of collision response in an interruptible system. As the collisions are rarely fully resolved, only approximate information is available to the response system. The contact modelling, and thus the response, can therefore only be applied using the approximate data. A relatively simple Dynamics Model is applied to this data to produce plausible motion, which often looks less sterile than physically correct motion [6]. The perceptual impact of the resulting response is discussed in [80].

2.5 Exact Algorithms

Much research has concentrated on performing exact collision detection between polyhedral models. Almost any surface can be represented as a set of polygons, provided the number is sufficiently high to represent any curved areas that may exist. Exact collision detection algorithms provide interference tests that are mathematically accurate to the level of the underlying mesh. A survey of such algorithms can be found in [55]. Higher order representations such as parametric patches and implicit functions are usually more suitable than polyhedral models for curved geometries. However, few algorithms explicitly use these representations for collision detection.

Many researchers have used bounding volume hierarchies to narrow in on the regions of contact between two models. The leaf nodes of these hierarchies contain lists of the triangles that are contained within them. When two bounding volumes are found to be overlapping, the triangles within them are tested using a simple triangle-triangle test.

Moore and Wilhelms [76] use an extended version of the *Cyrus-Beck* line clipping algorithm [90] to test for edge-face intersections between convex polyhedra. They deal with the collision detection and response for both rigid bodies and those articulated with a variety of joint types.

Other researchers have used closest point algorithms for collision detection. Most of these algorithms rely on the objects being convex in shape. It is often claimed that these can be easily extended to handle non-convex objects by decomposing them into hierarchies of convex polytopes, using an algorithm such as that presented by Ehmann *et al.* [21]. However, as the amount of non-convexity increases so does the number of convex elements

required to represent the object. Thus the amount of work that needs to be done by the collision detection algorithm increases with the complexity (non-convexity) of the object. Therefore, these algorithms can handle some non-convexity within the objects, but become more computationally expensive as the objects become more non-convex.

Closest point algorithms for convex polytopes usually fall into two categories: *Feature based* and *Simplex based.* Feature based algorithms are so-called because they consider objects to be composed of a set of features, i.e. faces, edges and vertices. They express the collision detection problem in terms of a relationship between these features. Lin *et al.* [65, 67] described the first such algorithm, which uses incremental distance computation for determining collisions between convex polyhedra. This algorithm, known as Lin-Canny, forms the core of the I-COLLIDE collision detection system. The algorithm tracks the closest features between a pair of objects. It exploits inter-frame coherence by using "feature-stepping", which allows the closest features to be quickly updated by examining those adjacent to the previously used ones. This enables the algorithm to maintain near constant performance between time-steps, for a fixed number of objects. However, it does not handle the case where objects inter-penetrate. When this situation occurs the algorithm enters a cycle, requiring a limit to be placed on the number of iterations allowed. This deteriorates the algorithm's performance unless the objects are moving slowly relative to the time-step. Ponamgi *et al.* [83] later introduced the notion of *pseudo internal Voronoi regions* to detect when the polytopes are penetrating. An overview of using Lin-Canny for non-convex objects, by constructing a hierarchy of convex polytopes, can be found in [66].

Mirtich's *V-Clip* algorithm addresses the chief drawbacks of the Lin-Canny algorithm [73]. This algorithm uses Voronoi planes to search for the closest points between the features of the polyhedra. Unlike the Lin-Canny, algorithm V-Clip does not enter a cycle when the polyhedra penetrate and does not require any numerical tolerances to be specified. The V-Clip algorithm is more straightforward to implement as it contains fewer special cases and is considered to be the fastest published algorithm for rigid convex objects.

The second family of "exact" algorithms are the Simplex-based algorithms. These approaches define a polyhedron as a set of Simplices, which are a generalisation of the triangle into any dimension, and perform operations on these simplices to track the closest points between two objects. Gilbert *et al.* [30] presented the first such algorithm, commonly known as *GJK*. Unlike the Lin-Canny algorithm, GJK can handle inter-penetrating objects and returns a measure of this inter-penetration. Rabbitz [87] improved this algorithm to exploit coherence between frames, while Cameron [13] developed it further to produce *Enhanced GJK*. Van den Bergen [105] presents solutions to the termination problems that can result from arithmetic round-off errors and provides methods for generating simplices on geometric primitives such as boxes, spheres, cones and cylinders. The enhanced GJK algorithm has a lower memory requirement than V-Clip and does not require

any preprocessing, which makes it more suitable for deformable and fracturable objects [64].

Quinlan [86] also considers distance computation as a method of achieving collision detection. Non-convex objects are represented using hierarchies of convex shapes, namely spheres. A search routine is used to reduce the time complexity of performing distance calculations in robot guidance. By allowing a small relative error in the result, Quinlan improves the efficiency of the algorithm. If 100% error is allowed, the returned distance is meaningless - however the result is zero *if and only if* a collision has occurred, thus allowing non-interruptible collision detection. However this work only considers a single object, i.e. the robot, moving in the scene.

2.6 Deformable and Parametric Objects

Much of the work on collision detection is concerned with rigid or articulated bodies, usually represented by a polyhedral mesh. However, some researchers have considered deformable objects and objects that are represented by higher order surfaces. This section summarises some of the work in these areas.

Moore and Wilhelms [76] deal with collision response in the context of flexible objects modelled using polygonal surfaces. As the vertices are in motion, each one defines a line between its pervious and current positions. These edges need to be tested against the triangles of another object to detect the collisions. This can be quite an expensive algorithm if there are large numbers of polygons.

Von Herzen *et al.* [107] detail a collision detection algorithm for objects that are composed of time-dependent continuous parametric surfaces. The algorithm finds the earliest collision or near-miss between objects, thus avoiding the problems caused by fixed time-step simulations, i.e. missing collisions that occur between time-steps. When performing collision detection between different types of surfaces, analysis is needed only once per surface type, as opposed to $O(n^2)$ combinations.

Snyder [99] performs interference detection between two parametric surfaces. However, the result only indicates if the two objects have collided, not where they touched. This data is important for the contact modelling phase of collision response, but is unnecessary for detecting impending collision when controlling robotic systems. In later work the determination of the contact points is addressed [100]. This work also tackles the situation where two objects come in contact over a large area by representing the contact area as a set of regularly sampled points. This not only applies to static parametric surfaces but also to time-dependent surfaces, which change shape as a function of time.

Volino and Thalmann [106] present an algorithm for performing efficient self-collision detection on flexible objects such as clothes and between the fabric and other bodies. Their algorithm deals with large numbers of polygons by utilising geometric properties to cull out a large number of potential self-collisions that could not possibly occur. This is

achieved by examining the shape of the object in the areas of folds.

Hughes *et al.* [51] present an algorithm for performing collision detection between parametric surfaces undergoing quadratic deformations. They use hierarchical sweep and prune to narrow in on sub-patches of the surface that are potentially in contact. They also deal with self-intersections involving individual patches of the surface.

Lin and Manocha [66] perform collision detection between models constructed using B-splines. Intersections between the splines are determined by finding the algebraic solutions to a set of equations.

Abdel-Malek *et al.* [2] present an algorithm for performing collision detection between complex objects, constructed with CSG techniques, by enclosing them in a number of parametric surfaces. Collision detection is performed by tracing out the curve of intersection between these surfaces.

Lombardo *et al.* [71] present an interesting algorithm for performing collision detection in a key-hole surgery simulator. As the surgical tools enter the body through a small opening, the movement is limited to motion around that fulcrum. Also, as the tool is cylindrical in nature, they implement collision detection as a query for which areas of the deformable body parts would be visible if looking along the cylinder that represents the tool. This is achieved using the *OpenGL* selection mechanism, which allows the collision detection to be performed in hardware. They also address the fixed-timestep problem by sweeping out the volume covered by the tool as it moves.

Ganovelli *et al.* [26] use the *Bucket-Tree* data structure for collision detection between deformable objects. This is a structure similar to the AABB-tree where the model elements are moved between the leaf nodes (called Buckets) as the object deforms.

2.7 Sphere-Tree Traversal

As discussed in Section 2.3, the narrow phase of the collision detection algorithm zones in on potential areas of contact by traversing a hierarchical representation of the objects. When using an interruptible algorithm it is important that each level of this hierarchy provides the necessary information for collision response. Although many geometric primitives have been used for non-interruptible collision detection the sphere-tree, described in Section 2.3.1, is still favourable when allowing interruption to occur. Attractive properties include :

- *Rotational Invariance* : A sphere is invariant to the rotation of the objects and therefore its update cost is very small. No matter what type of motion the bodies are going through the spheres can be updated by simply translating their centers.

- *Efficient* : The cost of performing an overlap between two spheres is extremely low, requiring only a few floating point multiplications and additions.

- *Suitability for Response* : While providing graceful degradation of the object approximation, each level of spheres in the hierarchy provides an approximation of the contact information necessary for collision response [18].

Many researchers have utilised sphere-trees for performing spatial localisation in collision detection. The stair-case traversal algorithm uses the sphere-tree data structure to narrow in on the regions of objects that are in contact. This section details the stair-case traversal algorithm and its use as part of an interruptible collision handling system.

2.7.1 The Traversal Algorithm

Having determined that a pair of objects is potentially colliding, using a broad phase algorithm such as Cohen's *sweep and prune* [17], the narrow phase algorithm determines the areas of the objects that are in contact. This is achieved by traversing the pair of sphere-trees associated with the objects. The traversal starts with the roots of both sphere-trees. If these two spheres overlap then it is necessary to test the next level, however if they do not overlap then the objects are not colliding. There are two approaches to the next stage of the traversal; to move down to the next level of both the trees and test pairs of children or; to move down to the next level in one of the trees. The second option, proposed by Palmer and Grimsdale [82], reduces the amount of comparisons that result from a positive overlap test. For hierarchies with n children per node a pair of overlapping spheres results in $O(n)$ further comparisons as compared with $O(n^2)$ if we move down a level in each tree. Figure 2.6 shows this algorithm in 2D, using circles (spheres) arranged on a quadtree (octree) for neatness. The traversal proceeds as follows : (1) the root nodes of the two trees have been found to be overlapping; (2) the B.1 nodes (tree B, level 1) are tested against the root node of tree A, the B.1 nodes that do not test positive are discarded (unshaded); (3) the remaining B.1 spheres are tested against the A.1 spheres, the ones that test negative are again discarded. This process results in a pair of level 1 spheres that are in contact, in a real scenario there may be many such pairs. A sub-traversal is then performed for each such pair (4 & 5).

O'Sullivan adapted this staircase traversal for interruptible collision detection. Having determined the colliding pairs, in the broad phase, all the sphere-trees pairs are traversed simultaneously in a breadth-first manner. The traversal is conducted in a time-critical manner, terminating when the allotted time period has expired. The algorithm maintains a queue of sphere pairs[3]. The queue is initialised with pairs of root nodes for the potentially colliding objects. As each pair is taken from the queue, the children of one of the spheres are tested against the other sphere, and new pairs are created for those that overlap. If both the spheres involved in the collision are leaf nodes they are entered into a list of resolved collisions otherwise the new pair is entered into the queue. When the algorithm

[3]O'Sullivan actually used a more complicated data-structure that allows objects to be scheduled according to the perceptual importance of the collisions.

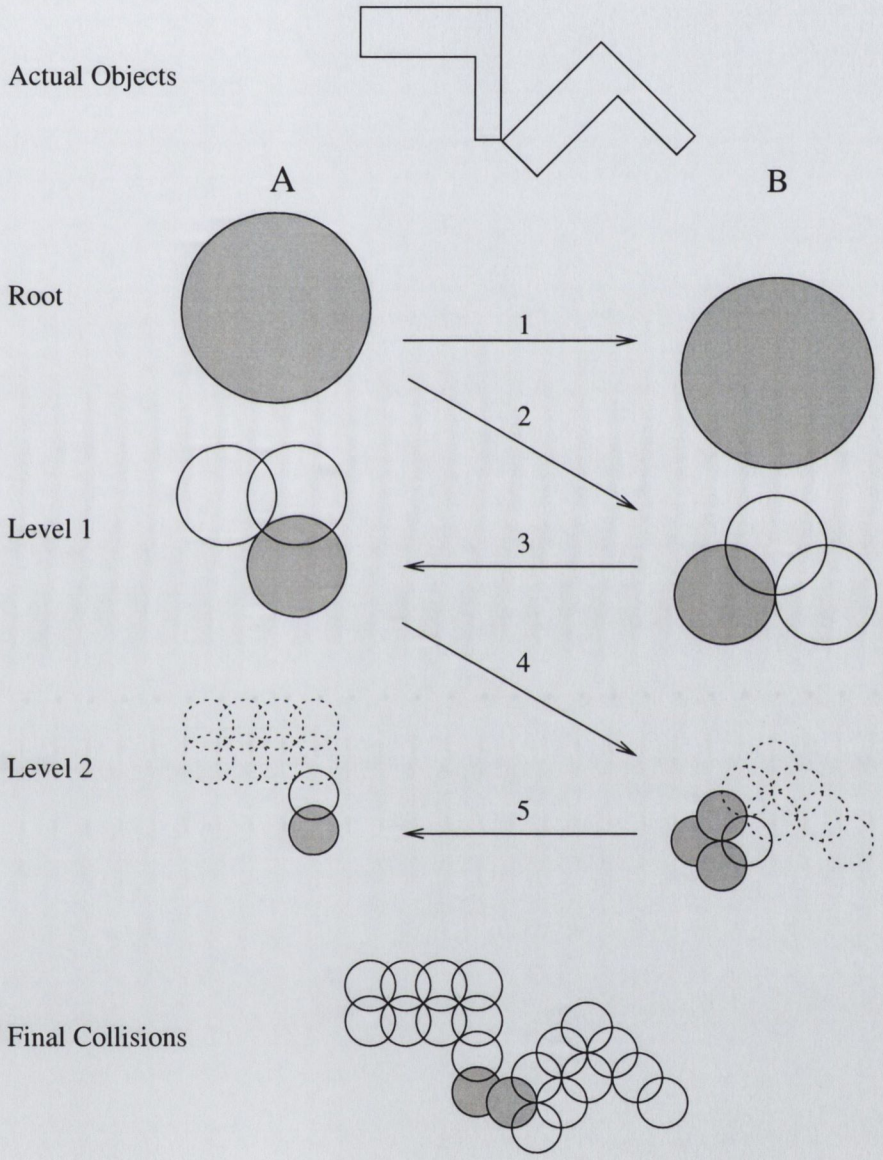


Figure 2.6: Staircase traversal of a pair of sphere-trees.

has to be interrupted - the current approximation of the collisions is used for all pairs of objects. Instead of taking the first pair of spheres in the queue, O'Sullivan [81] categorises the pairs of spheres as being high or low priority based on a model of human perception. The high priority pairs are checked for collisions in a round-robin fashion. When all the high priority collisions have been resolved, and the processing has not yet been interrupted, the low priority collisions are processed, again in a round-robin fashion.

2.8 Conclusion

This chapter has reviewed much of the research to date in collision detection and related fields. Many researchers have focused their work on this important topic. Table 2.1 summarises some of the "mile-stone" algorithms in rigid body collision detection.

Many of the classical algorithms express collision detection as a closest point problem. A number of the reviewed algorithms track the closest points between the objects in order to determine when collisions have occurred. These algorithms often require that the objects be convex and, therefore, non-convex bodies need to be decomposed into hierarchies of convex polytopes.

Many other researchers have adopted a hybrid collision detection algorithm to tackle the problem in stages. The top-level process aims to quickly discard pairs of objects that cannot possibly be interacting. The next phase aims to localise in on regions of the objects that are in contact so that only small areas need to be considered when conducting exact surface intersection tests. This phase uses hierarchical representations to cull away the uninteresting areas. Many different geometric primitives have been used for these hierarchies. There is usually a trade-off between the number of primitives required to represent the objects and the cost of working with a given type of primitive. Table 2.2 summarises many of these primitives.

Hubbard's interruptible collision detection algorithm takes a level-of-detail approach to collision detection. Hierarchical representations of the objects are traversed using a time-critical algorithm, which systematically refines the collision approximations. This allows the amount of time spent on collision detection to be carefully controlled, thus allowing more consistent frame-rates to be achieved. For an interruptible algorithm, spheres have distinct advantages over other bounding volumes. However, as they are used to approximate the surface of the object, the spheres need to fit the object very closely.

Recently, researchers have become increasingly more interested in performing collision detection with deformable objects. Some research in this area has also been reviewed in this chapter.

Table 2.1: Summary of Rigid Body Collision Detection Algorithms

ALGORITHM	CHARACTERISTICS	PROS	CONS
Moore & Wilhelms (see [76])	Test if any vertex from A is inside B and vice-versa. Test for edges of B that cut through the faces of A. Based on Cyrus-Beck line clipping algorithm.	• Simple to implement. • Can supply contact points.	• Objects must be convex. • Does not exploit inter-frame coherence. • Every object face needs to be considered for intersection.
Lin-Canny (see [65, 67])	The first feature based iterative closest point algorithm.	• Exploits inter-frame coherence.	• Objects must be convex. • Doesn't handle inter-penetration[a].
Palmer & Grimsdale (see [82])	Hybrid collision detection algorithm. Hierarchies of spheres used to localise collisions prior to triangle-triangle tests.	• Objects are only considered when they are in close proximity. • Efficient localisation of interacting regions. • Can handle non-convex and articulated objects.	• Octree method used to make sphere-trees can produce poor approximations. • Faces may be checked multiple times as they can be contained in multiple leaves of the sphere-tree.
Interruptible Collision Detection (see [47, 49])	Level-of-Detail collision detection algorithm. Bounding volume hierarchy made up of spheres approximate the objects.	• Provides consistent frame-rates. • Accuracy of the collision information degrades gracefully. • Collisions can be prioritised to resolve the important ones more accurately.	• Bounding volume hierarchies need to be tight fitting as they approximate the contact points. • Tight fitting sphere-trees do not lend themselves to deformable objects[b].

continued on next page...

[a]Ponamgi introduced pseudo internal Voronoi regions to detect penetration.

[b]Sphere-trees based on an octree can be computed on the fly but provide poorer fitting hierarchies.

... continued from previous page

ALGORITHM	CHARACTERISTICS	PROS	CONS
Enhanced GJK (see [13, 30])	Simplex based iterative closest point algorithm.	• Exploits inter-frame coherence. • Can handle inter-penetration. • No pre-processing required.	• Objects must be convex. • Slower than V-Clip.
V-Clip (see [73])	Feature based iterative closest point algorithm. Uses Voronoi regions.	• Exploits inter-frame coherence. • Can handle inter-penetration. • Fastest published "exact" algorithm.	• Objects must be convex. • Voronoi regions need to be pre-computed. • Relatively high memory requirement (compared with Enhanced GJK).

Table 2.2: Summary of Bounding Volume Primitives for Collision Detection

PRIMITIVE	UPDATE PROCEDURE	OVERLAP TEST	PROS AND CONS
Sphere	Apply transform to center of spheres.	Check if distance between centers is less than sum of radii.	↑ Low update and overlap costs. ↑ Can approximate contact points. ↓ Large number of spheres often needed.
Axis Aligned Bounding Box (AABB)	Apply transform to bounding box and fit new AABB that encloses original[a].	3 overlap tests, i.e. bounding boxes projected onto X, Y & Z axes.	↑ Efficient intersection test. ↓ Large number of AABBs required. ↓ Refitting decreases fit further.
	Apply transform to bounding box and construct OBB[b].	As for OBB.	↑ No refitting required. ↓ OBB intersection test is expensive. ↓ Large number of AABBs still required.
Oriented Bounding Box (OBB)	Transform the basis vectors and one of the vertices to reconstruct OBB.	Separating Axis Test (15 interval projections).	↑ Much tighter fitting than spheres or AABBs, fewer primitives required. ↓ Expensive overlap test.
Discrete Orientation Polytopes (k-DOP)	Apply transform to the k-DOP and refit so that each face is aligned with one of k allowed orientations.	$\frac{k}{2}$ interval overlap tests. All must overlap for k-DOPs to be intersecting.	↑ Controllable number of discrete orientations - controls cost and fit. ↓ Refitting decreases the tightness of fit.

continued on next page...

[a]It may be worth transforming one of the AABBs into the others local co-ordinate frame so that only one of the AABBs needs to be refitted.

[b]To apply a transformation to a bounding box requires the transformation of one of the vertices and the three vectors that define the edges.

... continued from previous page

PRIMITIVE	UPDATE PROCEDURE	OVERLAP TEST	PROS AND CONS
Quantised Orientation Slabs with Primary Orientations (QuOSPO)	Transform OBB of object B into co-ordinate frame of object A. Choose orientations to make new OBBs for A & B.	2 sets of 3 interval overlap tests[a].	↑ Relatively inexpensive overlap test. ↑ Provides similar fit to OBBs. ↓ High memory requirement for large numbers of slabs (discrete orientations).
Spherical Shells	Transform sphere center and the vector around which the shell is formed.	Four overlap tests between a shell restricted to one of its extreme radii and the other with variable radii. This requires finding the sign of quartic polynomials but not their roots.	↑ Inexpensive to update. ↓ Expensive overlap test (2-3 times that of OBB). ↑ Fast convergence to smooth surfaces with low curvature, fewer primitives required.
Sphere Swept Volume (SSV)	**PSS**: As for Sphere. **LSS**: Transform both end points. **RSS**: Transform one vertex and the two vectors defining the rectangle.	Collision detection is expressed as closest point test. Determine if closest points are on the edges of core shape (using Voronoi regions). Otherwise, use rectangle-rectangle algorithm for closest points (point-line etc. are sub-routines within this).	↑ Allows different primitives to be used in different areas of object. ↑ Update and overlap test depend on primitive. ↓ Many different cases for update and overlap tests. ↓ Large memory requirement if primitive types are to be chosen at runtime.

[a]If the first test indicates an overlap, objects A and B need to swap roles and another update and overlap test performed.

Chapter 3

Octree Method

The Octree method for sphere-tree construction has been widely used by researchers. It is by far the simplest algorithm for constructing sphere-tree and has been adopted by Palmer and Grimsdale [82], Hubbard, [48, 49] and O'Sullivan and Dingliana [18, 79]. This chapter presents the octree based algorithm for the construction of sphere-trees. An improved algorithm is then developed. This produces tighter sets of spheres by allowing more freedom in how they are arranged.

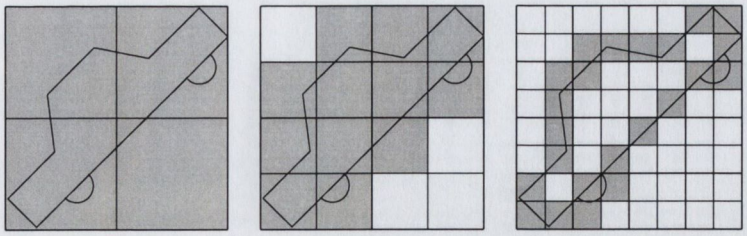
3.1 Constructing Sphere-Trees from Octrees

The *octree* is a data structure that provides a recursive subdivision of 3D space [94]. Each node of the octree is an isothetic cube, i.e. a cube whose faces are axis aligned. This is essentially a simplification of the AABBs described in Section 2.3.2.

The algorithm for the construction of an octree is recursive in nature and starts with the construction of a cube that contains the entire object. This root node is then sub-divided into eight child nodes, i.e. divided in half in each dimension, with each child node covering one eighth of its parent's volume. The child nodes that overlap the surface of the object are added to the octree. Those child nodes that do not contain any part of the surface are considered dead and are eliminated from the tree, this includes nodes completely contained inside the object. The algorithm then performs the same sub-division on each of the remaining child nodes. This recursion can continue to any required depth, each level approximating the object to a finer degree. Figure 3.1 illustrates this using a quadtree, the 2D equivalent of the octree.

This algorithm requires that we can determine whether an isothetic cube overlaps the surface of the object. If the object is polyhedral in nature this is a question of determining if the cube contains any part of one or more of the polygons that make up the surface mesh. This is simplified because the cubes are isothetic rather than arbitrarily oriented [34].

Once we have constructed an octree for the given object, it is a simple matter to



(a) First Level (b) Second Level (c) Third Level

Figure 3.1: Quadtree sub-division of an object (2D equivalent of an octree).

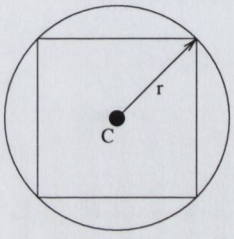


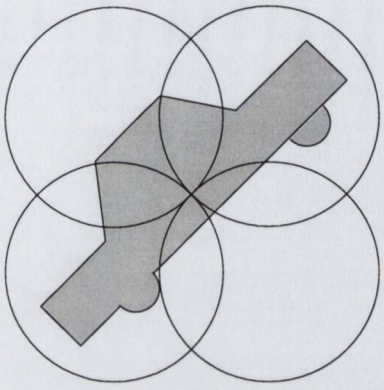
Figure 3.2: Each cube within the octree defines a sphere which surrounds it.

construct the sphere-tree. This is achieved by placing a sphere around each of the nodes. As the nodes are isothetic cubes, the sphere is placed around the center of the cube and the radius of the sphere is equal to the distance from the center to the corners, see Figure 3.2. The spheres fitted around the children of an octree node are children of the sphere fitted around that node. The set of child spheres covers the area of the object covered by the parent octree node, as illustrated in Figure 3.3. Hubbard suggests fitting a tighter sphere at the root level using Ritter's algorithm, which aims to fit a tightly bounding sphere around a set of points [89], the set of points being the vertices in the polyhedral mesh.

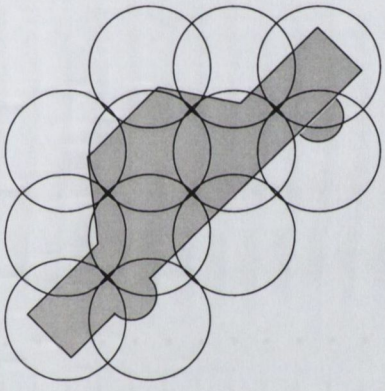
3.1.1 Pros and Cons

Each level of the octree contains cubes that are half the size of the parent cube. During the construction of the sphere-tree, spheres are created around these cubes. These spheres are half the radius of their parent sphere. Thus each level of the sphere-tree provides a successively tighter approximation of the object with each sphere covering a sub-volume of its parent. However, the rigid placement of the spheres does limit the achievable tightness of fit. This is undoubtedly a very simple algorithm for the construction of sphere-trees, but more sophisticated schemes can yield a closer approximation.

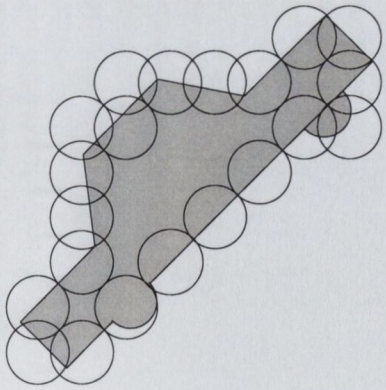
The octree algorithm only ever covers the surface of the object and thus the interior of the approximation does not contain any spheres. As the spheres in the lower levels of the octree become very small this can result in a collision being missed and one object tunnelling inside another, where it may become trapped. This makes a collision detection



(a) First Level



(b) Second Level



(c) Third Level

Figure 3.3: Sphere levels constructed using quadtree (2D equivalent of an octree).

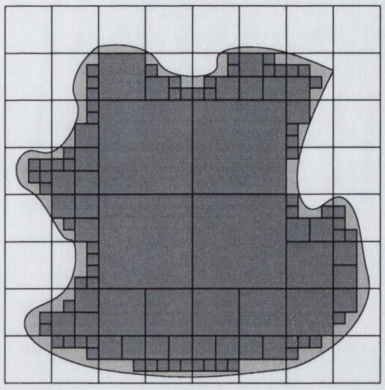


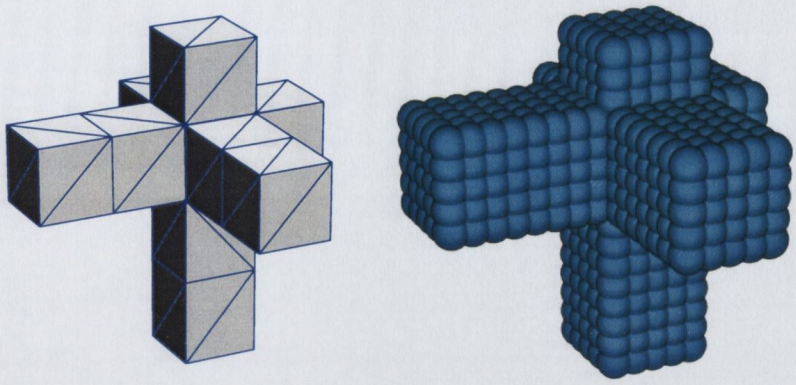
Figure 3.4: Octree nodes that are entirely inside the object create larger terminal nodes.

algorithm that uses sphere-trees constructed with this method more susceptible to the fixed time-step weakness (discussed in Section 2.1). The next section details a solution to this problem by extending the octree to cover the interior of the object.

3.1.2 Making Solid Octrees

In order to construct an octree that fills the interior of the object it is necessary to amend the above algorithm. The algorithm creates nodes from the child cubes that contain part of the surface, i.e. surface polygons. If we wish to have a sphere-tree that also contains spheres in the interior we must include those octree nodes that lie inside the object. If we were to treat these nodes in the same way as those that overlap the surface we would sub-divide the node into a set of children, thus introducing a significant number of extra spheres. This would add a significant overhead to the collision detection algorithm. However, the set of nodes that are inside the object and do not overlap the surface can be treated as a special case. If we were to sub-divide such a node we would find that *ALL* its children would also be internal to the object. Thus it is not necessary to sub-divide the node, and we can simply keep this node as a leaf node, as illustrated in Figure 3.4. A similar approach was used in [97] for compacting octree representations of volumetric objects that are in motion.

Such nodes can be determined by the following criteria : If the sphere placed around an octree node does not overlap the surface of the object but its center point is contained within the object then the node can be created as a terminal node. In order to determine if a node is to be treated with this special case it is necessary that we be able to determine if a point, i.e. the center of the sphere, is inside the object. Appendix A describes an algorithm for determining if a point is contained within a closed polyhedron.



(a) Model (b) Sphere-Tree (4^{th} Level)

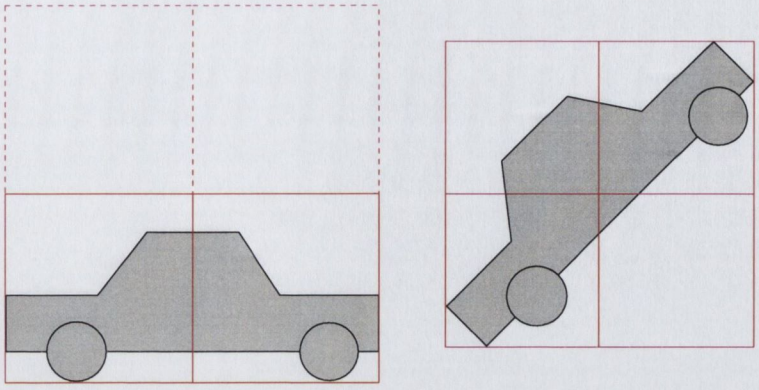
Figure 3.5: An example of a model that is well suited to the octree based algorithm.

3.2 Limitations of the Octree Method

When constructing a sphere-tree with the Octree Method, the configuration of the sphere-tree is dependent solely on the bounding box of the object and the object's shape is not explicitly used. Thus each level of the sphere-tree contains spheres that are identical in size and are positioned in a grid like arrangement. While this does ensure that the sphere-tree will have the necessary sub-division properties, it does not lead to very tight fitting approximations. Some shapes are, however, ideally suited to the octree method. A shape such as that in Figure 3.5 can be well approximated by an octree and the approximation has nice properties for contact modelling and collision response [18]. Also, the regular nature of the octree allows it to be quickly updated if the objects are undergoing deformations. This section details the improvements that have been made to the basic octree algorithm and derives a new algorithm that creates tighter fitting sphere-trees while maintaining good spatial sub-division.

3.2.1 Orientation and Position

When constructing the sphere-tree from an octree, the spheres are placed around the nodes of the octree. These nodes are isothetic (axis aligned) cubes. Thus, the orientation of the model within its local co-ordinate frame is a large factor in the goodness of fit. Figure 3.6 shows the first level of the octrees associated with an object in two different orientations. It is clearly visible that rotating the model by 45° prior to the construction of the octree reduces the size of the nodes. However, the number of nodes covering the object has increased and the lower right node has a lot of empty space. These problems arise from the lack of knowledge the octree algorithm has about the object it is approximating. If the algorithm were allowed to re-orient the set of spheres produced, it might be able to



(a) 0-degrees (b) 45-degrees

Figure 3.6: Changes in the orientation of an object affects the structure of the octree.

improve the quality of the approximation.

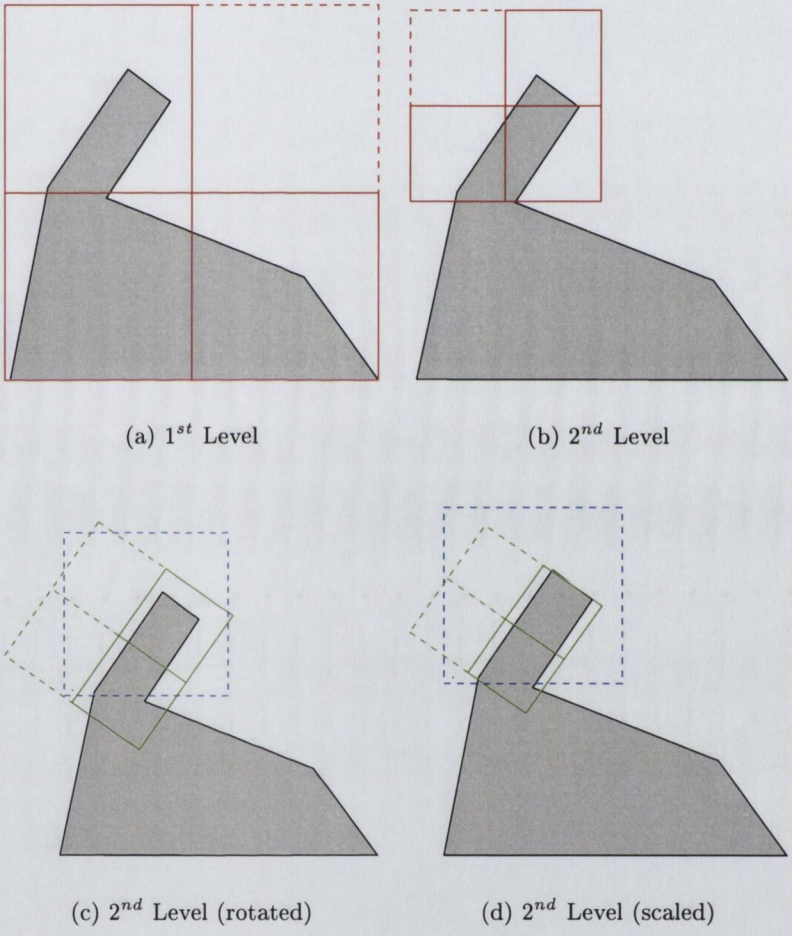
This can be generalised to allow each set of child nodes to have their own orientation and position. Allowing a node's children to be oriented at a different angle to the parent node would allow more freedom to approximate the object closely. This obviously destroys the nice properties of having *isothetic* nodes in the octree. However, this is not an issue when using the octree to construct a sphere-tree as each of the nodes will be replaced with its bounding sphere, which is rotationally invariant.

Figure 3.7 shows an example, in 2D, of an object that would benefit from allowing each node's children to be arbitrarily oriented. At the top level, Figure 3.7(a) the nodes are axis aligned. For the second level, the top left node produces a very poor approximation using axis aligned child nodes (Figure 3.7(b)). This configuration will also require three children to cover the object. The alternative configuration, Figure 3.7(c) would only require two children and would approximate that section of the object more closely. Thus, by allowing each set of nodes to be constructed within their own co-ordinate frame it is not only possible to improve the approximation but to also reduce computational overhead.

3.2.2 Size

The octree algorithm is also very limited in the way it chooses the radius of each sphere. When sub-dividing an octree node, the children are always half their parent's size. Thus all the spheres in a given level are exactly the same size, i.e. the radius is half that of the previous level. Figure 3.7(d) shows another approximation of the object. This approximation is even tighter than the last (Figure 3.7(c)) as the size of the nodes has been reduced.

Allowing the algorithm flexibility in the size of the nodes it uses gives it another degree of freedom to approximate the object. Thus allowing greater scope for achieving



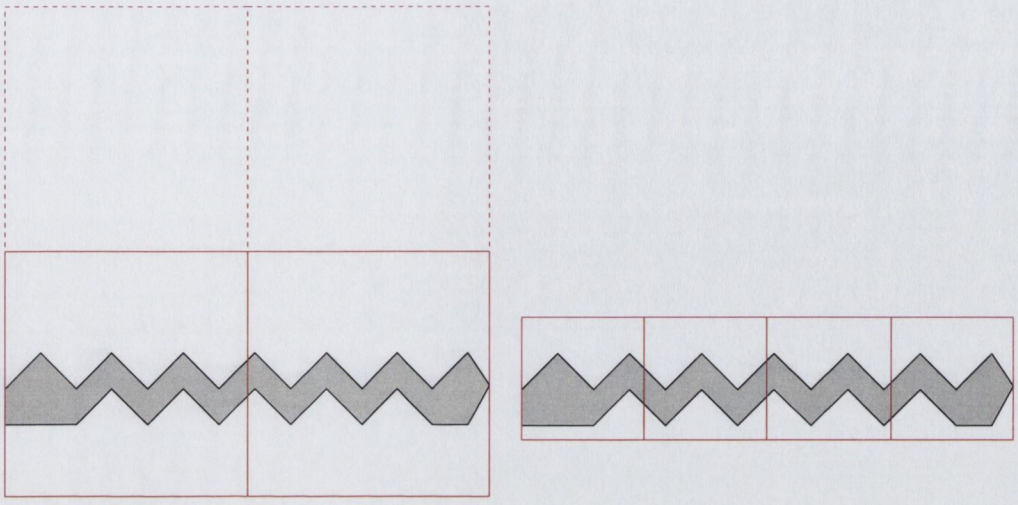
(a) 1^{st} Level

(b) 2^{nd} Level

(c) 2^{nd} Level (rotated)

(d) 2^{nd} Level (scaled)

Figure 3.7: Each set of nodes (siblings) can benefit from using a different local co-ordinate frame and independently sized sets of spheres.



(a) 1^{st} Level (b) 2^{nd} Level

Figure 3.8: Allowing the dimensions of the grid of spheres to change can improve the approximation.

tight fitting bounding volumes. As the algorithm is much more flexible in the way it arranges the spheres, each level of the hierarchy is likely to contain a variety of different sized spheres. This means that the collision response algorithm will not be able to make some of the assumptions that the octree algorithm may have allowed.

3.2.3 Grid Dimension

The octree algorithm only ever creates a $2 * 2 * 2$ grid of children. This clearly guarantees the children nodes will collectively cover the region of space covered by their parent. In order to approximate the object it is not necessary to cover the entire volume of the parent node. It will suffice to cover the area of the object that was contained in the parent node. This can be exploited to allow the grid of spheres to be of arbitrary dimension as well as orientation and scale.

Take for example a long thin object, such as that shown in 2D in Figure 3.8. Using a 2*2 grid, this object is bound by two squares, as featured in Figure 3.8(a). However this approximation is quite loose, i.e. there is quite a lot of empty space within the nodes, and only half the number of allocated nodes are being used. Using a different set of dimensions for the grid can allow a much tighter fit to be achieved, as seen in Figure 3.8(b). Although the second configuration uses more nodes to cover the object, it also provides better sub-division. There is a trade-off between the number of nodes used to cover the object and the culling power of the sub-division structure. Using more nodes will break up the object into more pieces, each of which can be culled. However, it is also necessary to test each of these nodes to decide if the sub-trees can be culled.

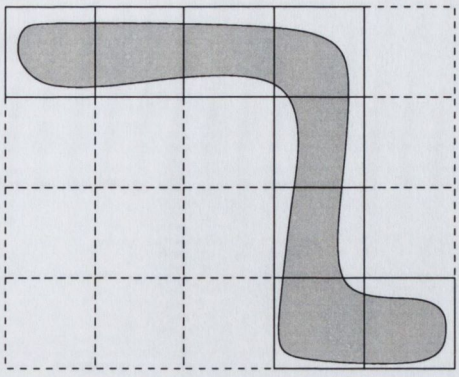


Figure 3.9: The algorithm may produce a grid of any dimension, provided it contains an allowable number of occupied nodes.

A more flexible scheme would be to dynamically determine the dimensions of the grid, upon which the spheres are placed. Any dimensions can be chosen, providing the number of nodes created is less than a specified maximum. For example, the grid may be 1*5*4, which has 20 nodes, but this would be acceptable if the object overlaps a sub-set of the nodes, e.g. 8, as shown in Figure 3.9. The octree algorithm can generate at most eight children per sphere. However, as the new algorithm is allowed to vary the size of the spheres, this is no longer the case.

3.3 The Grid Algorithm

The previous section identified a number of ways in which the basic octree method can be improved. By loosening the way in which the spheres are placed, tighter fitting sets of spheres can be generated. This section outlines an algorithm that uses this looser definition to approximate a section of the object by using an arbitrarily oriented grid of spheres.

The algorithm can be formulated as an optimisation problem, this is shown in Algorithms 1 & 2. The objective of the algorithm is to find, for a given size of sphere, the grid that requires the fewest spheres to approximate the object. The rationale behind this is that arranging the spheres so as to use only the smallest number required will remove the need for the spheres that contribute little to the approximation, enabling the algorithm to reduce the number of spheres required. Using the number of spheres required to cover the object as an objective function will not give the optimisation algorithm sufficient information to find a good solution. A better alternative is to formulate a function that characterises how well filled each sphere is. Thus, the optimiser will aim to increase the amount of the surface covered by some of the spheres allowing other spheres to be discarded. A similar function was used in [10], to group points from 3D clouds into 2D curves.

In order to encourage the optimisation function to reduce the number of spheres re-

quired to cover the surface, each iteration should aim to update the current arrangement so that the high ranking spheres, which already cover a large amount of the surface, will cover even more and reduce the dependence on the remaining spheres (which cover less of the surface). If we consider the surface to be represented by a densely sampled set of points then this equates to moving the grid so as to decrease the dependence on the lower ranked spheres in the set, hoping that they will eventually be discarded. We can compute the number of points requiring a given sphere by counting the points contained within the sphere that are not already covered by a higher ranking sphere. Thus the optimisation function, presented as Algorithm 2, can be expressed as :

$$M = \sum_{\forall S_i} f(Count(S_i)) \tag{3.1}$$

where :

S_i	is one of the spheres,
$Count(S)$	is the number of points that require S to be kept,
$f(x)$	is a function that has greater than linear growth so as to favour dense groupings of points within each sphere, typically $f(x) = x^2$.

Thus, to find the minimum number of spheres to cover the object it is necessary to find the orientation (and position) of the grid that maximises M. Any general purpose optimisation algorithm can be used to find these parameters. The optimisation algorithm requires an initial guess from which to start. The simplest way to generate an initial configuration would be to use the spheres generated by the octree method, i.e. 8 axis aligned cubes covering the area of their parent. A better solution is to use spheres placed within the *Oriented Bounding Box* (*OBB*) of the required area.

Gottschalk *et al.* use *Principal Component Analysis* (*PCA*) to generate OBBs [33]. They determine the eigen-vectors of the covariance matrix created from the region's convex hull. This finds the smallest ellipsoid that encloses the set of points. However, for regions that are roughly cubic (or square in cross-section) the resulting bounding box is often quite poor [110]. García used the inertial tensor instead of the covariance matrix, which seems to be less susceptible to these problems [27]. Barequet presents an algorithm for getting tighter approximations of the bounding box [5].

We have found that allowing the optimisation algorithm to vary both the orientation of the grid and the size of the spheres makes it difficult to find a good solution. Therefore, a simple linear search is used to vary the size. Starting with the minimum size required to cover the object with a single sphere, the size is reduced by a small amount, say 10%, until too many spheres are required to cover the object. The set of spheres with the least error is chosen to represent that area of the object.

The optimisation function aims to minimise the number of spheres required to approximate an area of the object. This in turn allows the algorithm to use smaller spheres. A second optimisation function can be used to refine the grid's orientation etc. to minimise the error in the approximation. However, a more versatile solution is to allow each sphere to move independently. This can be achieved using our generic optimisation algorithm, which will be developed in Section 6.2, to minimise the spheres' volume.

3.4 Conclusions

The rigid manner in which the octree method arranges the spheres and its lack of use of the object's geometry seem to be the main factors that contribute to its poor approximation of an object. As spheres, which are rotationally invariant, are to be placed around the nodes of the octree there is no requirement for them to be axis aligned. There is also no requirement to cover the entire volume of the parent node. It is sufficient to cover the parts of the object that lie within the parent node. Therefore, it is desirable to allow the algorithm to change the size of the spheres and to allow the grid, upon which they are placed, to be of arbitrary dimension.

Loosening the rigid scheme used by the octree method will allow much more freedom in how the spheres are arranged. This will allow the algorithm to find better arrangements of spheres by taking the object's geometry into account, increasing the algorithm's potential to achieve close approximations.

Algorithm 1 GRID algorithm for generating spheres.

Input : Set of surface points defining region to be approximated, P.
Maximum number of spheres, N.
Output : Set of spheres, S.

GRID*(S, P, N)*
  $B \leftarrow$ OBB for points P {*get an initial grid*}
  $s \leftarrow$ length of longest edge of B

  $S \leftarrow$ set of spheres covering P, arranged on grid
    of size s aligned with B {*make initial spheres*}

  $Fit_S \leftarrow 0$
  repeat
    $s \leftarrow s * 0.90$ {*reduce grid size by* 10%}

    $B_{opt} \leftarrow$ optimise the orientation of B to
      minimise the value of the CoverMetric
      for spheres of size s {*optimise grid*}

    $T \leftarrow$ set of spheres covering P, arranged on grid
      of size s aligned with B_{opt} {*make new spheres*}

    $Fit_T \leftarrow$ evaluation of how well set T approximates object
    if $\|T\| \leq N$ **and** $Fit_T > Fit_S$ **then**
      $S \leftarrow S_1$ {*new best set*}
      $Fit_S \leftarrow Fit_T$
    end if
  until $\|T\| > N$

Algorithm 2 Evaluate metric in Equation 3.1 for a given grid.

Input : Set of surface points, P. Grid orientation, O.
Grid origin, v. Grid size, s.
Output : Set of spheres S. Evaluated metric, *metric*.

COMPUTEMETRIC*(S, P, O, v, s)*

$T \leftarrow$ set of spheres covering P, arranged on grid defined by size s, orientation B and origin v {*make new spheres*}

for all $g \in G$ **do** {*count points in spheres*}
  $Count[g] \leftarrow$ number of points from P inside g
end for

$S \leftarrow \{\}$ {*evaluate metric*}
$metric \leftarrow 0$
for i = 1 to $\|G\|$ **do**
  $bestG \leftarrow NIL$ {*sphere covering most points*}
  $bestCount \leftarrow 0$
  for all $g \in G, g \notin S$ **do**
    if $Count[G] > bestCount$ **then**
      $bestG \leftarrow g$
      $bestCount \leftarrow Count[G]$
    end if
  end for

  $metric \leftarrow metric + bestCount^2$ {*accumulate term*}

  if $bestCount > 0$ **then**
    $S \leftarrow S \cup \{bestG\}$ {*add to sphere set*}

    for all $p \in P$, p inside $bestG$, p not already covered by S **do**
      for all $g \in G$, g containing p **do**
        $Count[g] \leftarrow Count[g] - 1$ {*point is now covered*}
      end for
    end for
  end if
end for

Chapter 4

Medial Axis Method

In his thesis Hubbard explored two ways of improving upon the use of *Octrees* for constructing sphere-trees. He used *simulated annealing* and the object's *medial axis* for the construction of *Sphere-Trees* [49]. This chapter describes the latter method, which essentially contains three phases; The approximation of the medial axis of the model using a Voronoi diagram; creating a base set of spheres using the interior vertices of the Voronoi diagram; and the construction of a *Sphere-Tree* from this set of spheres.

4.1 Constructing the Medial Axis

The medial axis surface of an object represents its *skeleton.* The usefulness of the medial axis for the generation of sphere-trees comes from the symmetric nature of the object around the skeleton [49]. Blum and Nagel defined the medial axis as the centers of a set of maximally sized spheres that fill a figure [7], naturally leading to the conclusion that the medial axis would be a good place to center the spheres to approximate the object.

However, finding the medial axis of an object is a very computationally expensive problem. As the medial axis is to be used only as a guide for the placement of the spheres it is not important to construct the exact medial axis, so an approximation will therefore suffice. The medial axis of an object can be approximated by the 3D Voronoi diagram of a set of points distributed over the object's surface, a 2D example of a Voronoi diagram is shown in Figure 4.1. For each sample point on the object, the Voronoi diagram represents the region of space that is closer to that point than any other point in the set. These regions, the Voronoi cells, are convex polyhedra whose vertices form the vertices of the diagram. The faces of the Voronoi diagram separate each point from its neighbouring points. As the Voronoi cell represents the region closest to a given point each face will be equidistant from the two points that it lies between. For a pair of points lying on opposite surfaces of the object this face will lie close to the middle of the object and so will be part of the approximate medial axis for that object. The vertices of the Voronoi diagram that are inside the object, and the faces whose vertices are all internal, can be used to

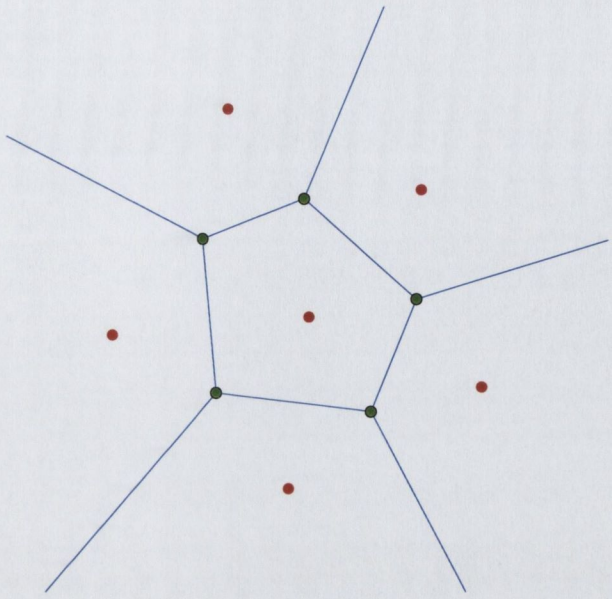


Figure 4.1: An example of a Voronoi diagram in 2D.

approximate the medial axis of the object. The larger the number of samples, the more accurate the approximation. Thus it is necessary that we are able to generate a set of sample points across the surface of the object.

4.1.1 Sampling a Triangulated Polyhedron

Hubbard uses a two phase process for the generation of the set of surface samples. First he tries to uniformly distribute a set of samples across the surface. After constructing the Voronoi diagram for this set of points, additional points are added to correct problem areas of the diagram. Such problems result from the choice of surface points and take the form of breaks in the medial axis and sections of the medial axis joining areas of the object that should remain separate.

To generate the initial set of points, Hubbard first distributes points across the surface of the polyhedron, using a method derived from Turk's algorithm [103]. A relaxation technique is next used to even out their distribution. In order to get this initial distribution of points, each triangle has a probability of receiving a point that is proportional to its area. To assign a point to a triangle, a random number between 0 and 1 is generated. This number is used to determine which triangle will receive the point. This is achieved by ordering the triangles according to area and using the number as a fraction of the cumulative sum of the areas. The chosen face now receives a point randomly located within it using barycentric co-ordinates [1].

Turk's algorithm next applies an iterative relaxation technique that pushes the points around under the influence of their neighbours. The forces between points on different triangles can result in points being pushed off the surface. To overcome these problems,

Hubbard groups neighbouring triangles into planar clusters. When performing relaxation each cluster is treated as an individual unit. This ensures the points will not leave the surface. To try to maintain the uniformity of the distribution between clusters, each edge of the cluster exerts a force on the points that mimics a set of evenly distributed points on the opposite side of the edge. This, however, drives points away from the edges - requiring an extra set of samples to be distributed along them.

In Section 5.1 an alternative sampling algorithm is presented. This scheme distributes points on a regular grid within each triangle/cluster and then adaptively adds more points to guarantee the fit of the spheres generated and to guarantee that all areas of the object are covered. This method also handles the cases covered by the additional samples added in Hubbard's method without needing any numerical tolerances or special sampling along the edges.

4.1.2 Constructing the Voronoi Diagram

In order to construct the Voronoi diagram it is necessary that we be aware of its structure. Each point in the diagram will be surrounded by a Voronoi cell, i.e. the region closer to it than any other point. Each Voronoi vertex is the circumcenter of four sample points, i.e. a sphere centered at the vertex will touch the surface of the object at four points. These points are referred to as the vertex's *forming points*. Each Voronoi vertex lies on exactly four cells and adjoins four edges. This gives each vertex exactly four neighbours [72]. It can often look as if a vertex adjoins more than four cells but this is not the case. An edge of the Voronoi diagram is allowed to have zero length, thus a vertex with more than four neighbours is simply a pair of vertices with a very short edge between them. Hubbard suggests that sets of such vertices should be combined after the construction of the diagram to prevent complications in the later stages.

The algorithm used to construct the Voronoi diagram is an iterative one, based on Bowyer's algorithm [9], which allows additional points to be added later on to correct any errors in the medial axis. As the algorithm assumes that all points being added to the diagram are inside the region already covered by it, an initial diagram must be constructed. The initial diagram is constructed using a tetrahedron that bounds the object. The corners of the tetrahedron are used as the set of initial forming points. For each of these points there is a cell, which surrounds it. These cells meet at a vertex in the center of the tetrahedron. Four additional vertices are constructed outside the tetrahedron by projecting the center point through the tetrahedron's faces. As each Voronoi vertex needs four neighbours, each one is connected to the rest. Figure 4.2 illustrates this in 2D, using a triangle instead of a tetrahedron and three forming points (and dummy vertices) instead of four.

The algorithm proceeds to build the Voronoi diagram by adding each surface point in turn. Each new point extends the diagram to contain a new cell, representing the region

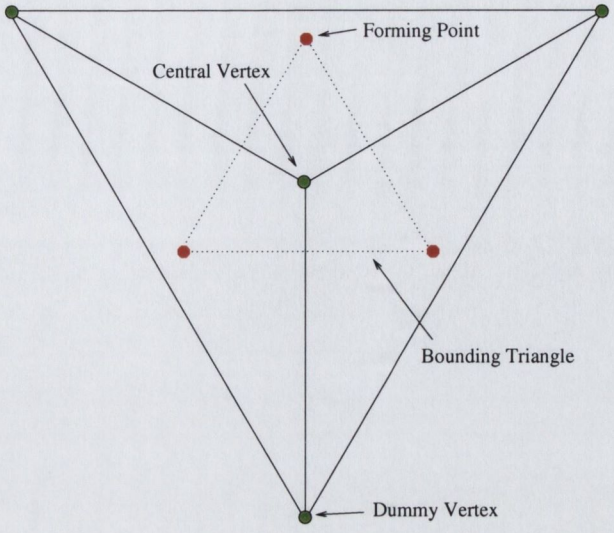


Figure 4.2: Initial Voronoi diagram in 2D. Vertices are in green, and forming points in red. Solid black lines join the neighbouring vertices.

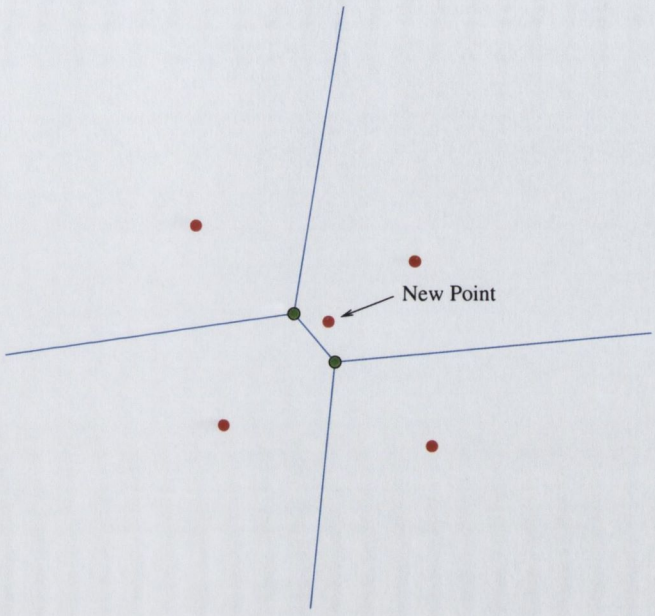
closer to that point than any other forming point, see Figure 4.3. The addition of this new cell causes the surrounding cells to shrink. This is achieved by deleting a number of the vertices and replacing them with new vertices. The algorithm deletes every vertex that is closer to the new point than its forming points. The replacement vertices are then created so that the faces of the Voronoi cells are once again equidistant from the forming points of the two cells sharing the face.

When a vertex is already equidistant from its forming points and the new point numerical imprecision becomes an issue. It is difficult to decide if the vertex should be deleted or not. Hubbard uses an algorithm, described next, which chooses the deletable vertices so as to make the diagram both accurate and robust, i.e. the structure of the diagram will not be compromised. Also the algorithm cannot tolerate coincident forming points and therefore these must be removed.

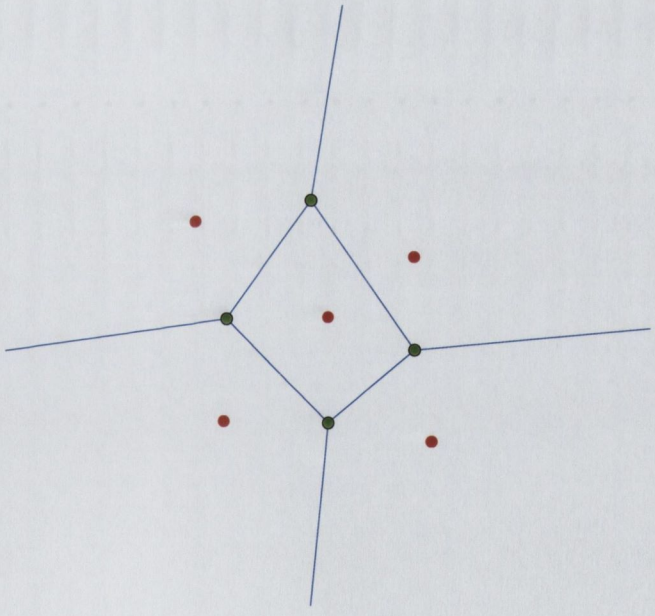
4.1.3 Selecting the Vertices to Delete

When selecting the vertices to be deleted from the Voronoi diagram, while adding a new surface point, it is important to maintain the topological structure of the diagram. Inagaki *et al.* state five criteria for preserving this structure [53] :

- the set of deletable vertices is non-empty,
- all the deletable vertices form a connected sub-graph,
- at least one vertex from each cell is *NOT* deletable,
- the deletable vertices on each cell form a connected sub-graph,



(a) Four Points



(b) Five Points

Figure 4.3: Adding a new point to the Voronoi diagram creates a new cell representing the region for which it is the closest forming point.

- the non-deletable vertices on a cell form a connected sub-graph.

A greedy algorithm is used to construct the deletable set. The algorithm starts with the vertex closest to the point being added, which is guaranteed to be deletable. The neighbouring vertices are next considered and those that satisfy the above criteria are added to the deletable set. The search then moves onto the neighbours of those vertices. The algorithm does not however consider the consequences of adding a vertex to the deletable set, i.e. choosing to delete a vertex may prevent the algorithm from later including a vertex that needs to be deleted.

Hubbard improves on this search algorithm by considering the vertices in order of suitability. Vertices that are much closer to the new point than their forming points are clearly deletable. Thus the measure of deletability, for a vertex, is the distance to its forming points minus the distance to the new point. The search for the set of deletable vertices starts at the vertex with the highest value and maintains a priority queue of the possibly deletable vertices, which is initialised with the positively valued neighbours of the starting vertex. The priority queue allows the algorithm to consider the vertices in order of value. If a vertex cannot be added to the deletable set it is held back for later consideration. When a deletable vertex is found, its positively valued neighbours are added to the priority queue, and the vertices that were being held back are returned to the queue as they may now be valid deletable vertices.

This algorithm does not find the globally optimum set of deletable vertices, but it does find a set to which no more vertices can be added and which maintains the topological structure of the Voronoi diagram. In [49], Hubbard presents results that indicate that this algorithm produces a more accurate Voronoi diagram than using Inagaki's criteria alone.

An alternative criterion for the selection of a valid set of deletable vertices would be to only allow vertices that maintain the topological structure of the Voronoi diagram. As the set of deletable vertices is built up the necessary conditions must be maintained :

- the set of forming points to be used to create a new vertex cannot be co-planar,
- each new vertex will have four neighbours, three of these being the other new vertices with which it shares three forming points. When constructing the deletable set there will be at most three such neighbours for each new vertex.

These also aim to maintain the topological structures of the Voronoi diagram, and ensures that each new vertex will be created from a set of non-coplanar points. This test is more straightforward to implement and has been more efficient during the construction of Voronoi diagrams used in this thesis.

4.1.4 Updating the Voronoi diagram

In order to create the Voronoi cell surrounding the new point, the algorithm has to replace the deletable vertices with new ones. Bowyer's algorithm creates a new vertex for each pair

of neighbouring vertices V_d and V_u, where V_d is a deletable vertex and V_u is an undeletable vertex. These vertices will share three forming points, which together with the new point will create the new vertex. The circumcenter of the tetrahedron formed by these four points gives the vertex's location, as shown in Figure 4.4(a). As with all Voronoi vertices, the new vertex V_n has four neighbours, V_u and three other new vertices. These other neighbours can be found from the set of new vertices by doing a brute force search for the the ones that share three forming points with V_n, see Figure 4.4(b).

If the set of deletable vertices is not entirely correct, it is possible that the set of forming points for a new vertex are all coplanar. Thus it would be impossible to create the circumsphere, and hence give the vertex a location. Inagaki *et al.* note that the position of the new vertex should lie on the edge connecting V_d and V_u and that the circumcenter of the tetrahedron would be infinitely far away. Thus the position of the vertex can be approximated with V_u, which is preferable to using V_d as it is not closer to the new point than its forming points [49].

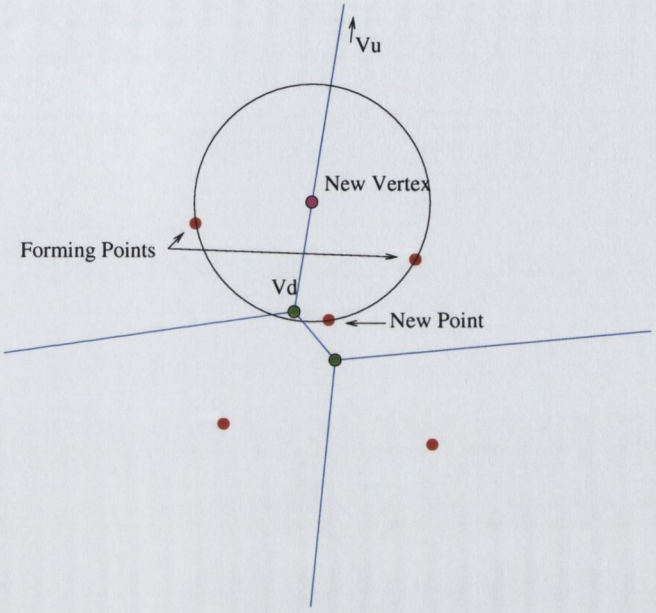
4.2 Fixing the Medial Axis

Approximating the medial axis by constructing a Voronoi diagram requires that the surface be sampled in an even, consistent manner. For a general polyhedral object this is a difficult problem. There are essentially two types of error that can occur, as illustrated in Figure 4.5. The first results in a break in the medial axis, where it leaves the object through its surface. The second is where the medial axis re-enters the surface to form a bridge.

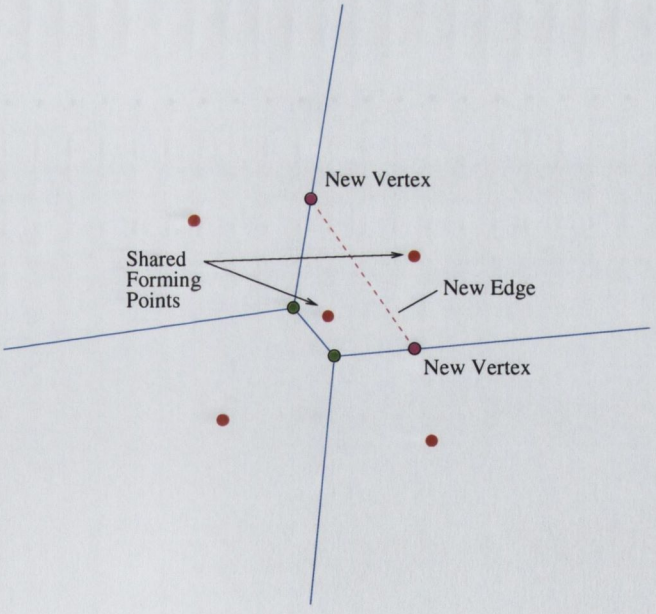
Hubbard suggests a scheme for correcting errors in the medial axis that result from sampling problems. He introduces the notion of gap crossing cells as a way of detecting these errors [49]. The Voronoi cells that lead to the medial axis approximation containing breaks, or joining two separate parts of the object (by crossing the gap between them), are called gap crossing cells. A cell, around point p, is considered gap crossing if it intersects a side of the object on which p does not lie. This situation results in the faces of the cell leaving the object to form a break in the medial axis or a bridge if it re-enters the object at some other point. His specific definition is :

> "The two cells around P_j and P_k are gap crossing if the shared face intersects a cluster that does not contain both P_j and P_k."

New points are introduced to the Voronoi diagram at strategic locations to correct these errors. Obviously the new cells, created as a result of the addition of the new points, will then need to be checked for gap-crossing. The new point is created by projecting the point(s) that does not lie on the cluster onto the plane of cluster. If neither of the projected points lies inside one of the cluster's triangles, then one of the points is snapped

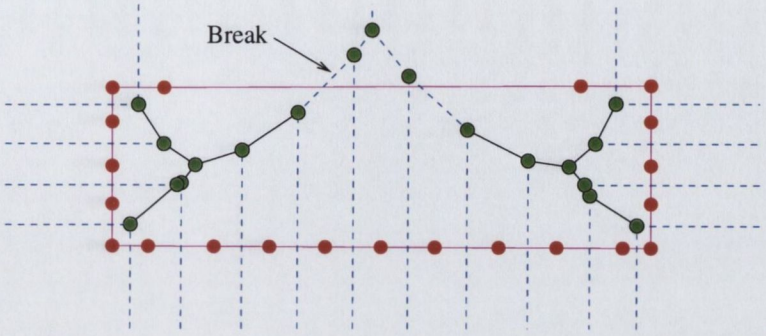


(a) New Vertex

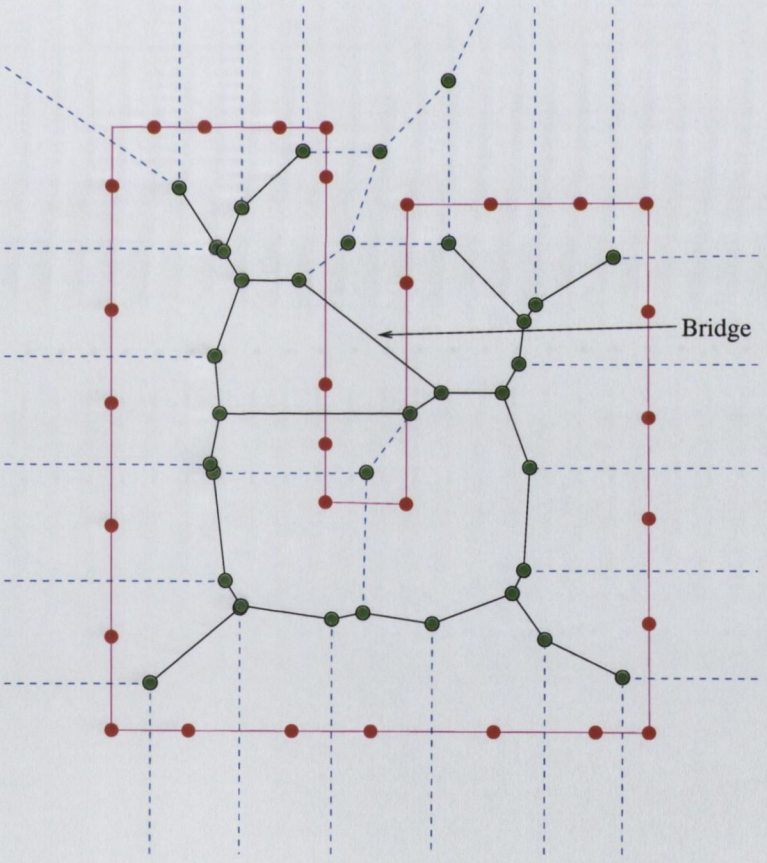


(b) New Edge

Figure 4.4: Creating a new vertex for a pair of vertices V_u and V_d and a new edge from two of the new vertices.



(a) Broken Medial Axis



(b) Joined Medial Axis

Figure 4.5: Examples of how sampling can cause errors in the medial axis.

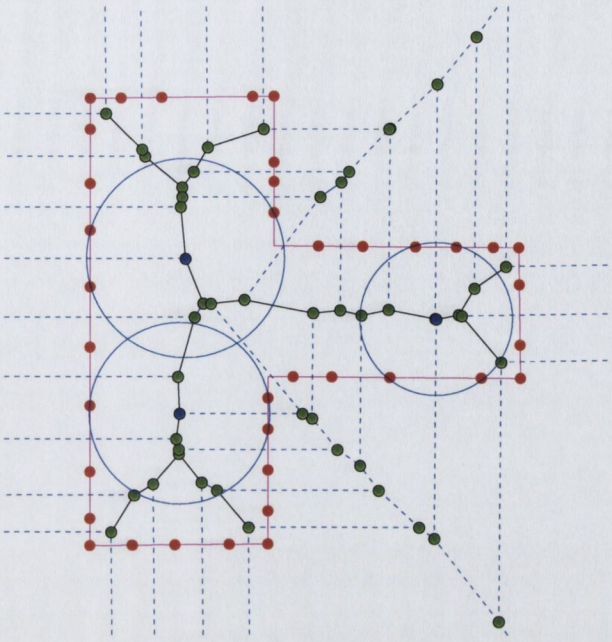


Figure 4.6: Examples of spheres placed around the Voronoi vertices that are inside the object.

to its closest point on any of the triangles within the cluster that the face intersected. If the projected point has to be repositioned in this way, it is not guaranteed to fix the problem and many new gap-crossing cells may result.

4.3 Constructing the Sphere-Tree

The construction of the sphere-tree starts with the creation of a set of spheres, from the medial axis approximation, which represents the object. Each internal Voronoi vertex gives the center of a sphere and the distance to its forming points determines its radius. As the medial axis of the object can be defined as the set of maximally sized spheres that fill its interior, the vertices that are inside the object represent good locations around which to place spheres. Each vertex is equidistant from its forming points, therefore using this distance as the radius of the sphere ensures that it will touch the surface in at least four locations, Figure 4.6 shows this in 2D with each circle touching the surface in three places. Hubbard uses a point in object test that categorises points according to whether they are behind the closest part of the surface. Although this algorithm does categorise some internal points as being outside the object it suffices for this purpose, see Appendix A for a description of this and some alternatives.

Obviously the set of spheres generated from the medial axis is not much good for performing collision detection. There will be a lot of spheres in the set and they are not arranged in a hierarchical manner, which is essential for performing the spatial localisation required for the narrow phase of the collision detection algorithm. Therefore it is necessary

to further process the set of spheres to form a hierarchical structure.

The construction of the sphere-tree proceeds in a top down fashion. The root of the tree, which contains only a single sphere is created using Ritter's algorithm [89]. This algorithm approximates the smallest sphere that contains a set of points, i.e. the set of surface samples or the forming points. Each non-terminal node of the tree is required to have a pre-specified number of children (N_c). Thus for the first level of the tree, the set of medial spheres must be reduced down to contain only N_c spheres, all of which are children of the root node. This is achieved using a successive merge algorithm in which pairs of spheres are combined until the required number of spheres is reached.

Successive levels of the tree are constructed using a sub-set of the medial spheres. This set contains only the spheres, from the original medial axis approximation, that were merged to create the parent node. These are once again successively merged until the specified number of spheres has been reached. This cannot continue to an arbitrary depth as the medial axis approximation has a limited number of spheres. The number of medial spheres is determined by the number and position of the surface samples and so there may be insufficient spheres in the medial set to create a tree of the required depth.

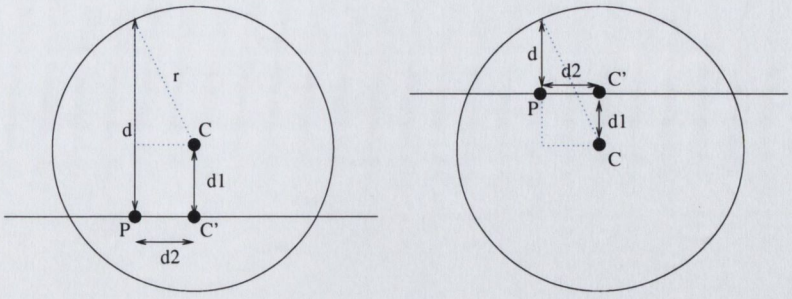
Sphere Merging

When merging a pair of spheres, S_1 and S_2, a new sphere is created to cover the parts of the object that these spheres covered. The object is represented by a set of surface points, as it is very expensive to use the polyhedral surface itself. If the set of forming points is used for this purpose, each sphere initially contains its four forming points. When the merge occurs, the two sets of points are combined and Ritter's algorithm determines the new bounding sphere S_{12}.

Each iteration of the algorithm reduces the set of spheres by combining two spheres into one. If each sphere was allowed to merge with any other sphere it would be very expensive to decide which pair of spheres to merge. Hubbard reduces the computational complexity of this process by only considering certain pairs of spheres. As each sphere in the initial set corresponds to a vertex in the Voronoi diagram, the sphere can be considered to be adjacent to a sub-set of the other spheres, i.e. its neighbours. When two spheres S_1 and S_2 merge, any spheres adjacent to either of the spheres become adjacent to the new sphere S_{12} [1].

The algorithm uses a heuristic to determine which pair of spheres to combine. Each time the algorithm merges a pair of spheres it picks the pair that will introduce the least amount of error into the approximation. For each of the candidate pairs, the new

[1] Any sphere that has no neighbouring spheres will be unable to merge with others and therefore will be contained in the reduced set. This will reduce the quality of the approximation. In the implementation used for this thesis, when a sphere is found to have no neighbours a set is computed to contain all the spheres that it overlaps. This avoids the situation of having spheres that have no neighbours with which to merge.



(a) C in front (b) C behind

Figure 4.7: Computing the distance from a point P to the surface of the sphere, for the two cases of C being in front of or behind the plane on which P lies.

bounding sphere is computed and its error evaluated. When a merge takes place, the new candidate pairs, which are formed during the merge, will also need to be evaluated. When computing the fit of the spheres, Hubbard computes the distance from each forming point to the sphere. This is not as accurate as measuring the distance from the sphere to the surface but is more efficient. To compute the distance from a surface point to the surface of the sphere, the center of the sphere is projected along the point's normal onto the plane defined by the point. The distance can then be computed as :

$$d = \sqrt{r^2 - d_2^2} + d_1 \tag{4.1}$$

if the center of the sphere, C, lies in front of the plane, as in Figure 4.7(a) and :

$$d = \sqrt{r^2 - d_2^2} - d_1 \tag{4.2}$$

if it lies behind it, as in Figure 4.7(b).

4.4 Pros and Cons

Using the medial axis for constructing sphere-trees certainly has major advantages over the octree method, reviewed in Chapter 3. Although the algorithm is much more complicated, the sphere-trees constructed with it approximate the object in a tighter way. This improvement in accuracy can be attributed to the freedom that the algorithm has in where it can place spheres. In the octree method, the position of the spheres is almost totally independent of the object's geometry. In contrast, the medial axis method explicitly uses the object's geometry, in the form of its skeleton, for the creation of the initial set of spheres. The medial axis method is far more computationally intensive than the octree method, however this is not an issue when the sphere-trees are constructed as a pre-processing step. The octree method has advantages when working with deformable

objects as it can be updated dynamically while the object is deforming.

However, there are a number of areas in which the medial axis technique can be improved. When generating a set of sample points, it is impossible to know how to choose a good set of points for the construction of the medial axis, i.e. how many sample points will give you the desired results. The sampling scheme presented uses a number of numerical parameters that are also difficult to determine analytically. Having constructed the Voronoi diagram and used it to create a set of initial spheres, there is no guarantee of the quality of this set, i.e. the set of spheres cannot approximate the object to a specific tolerance. Additionally, there is no guarantee that the set of spheres fitted to the medial axis will cover the entire object. The sphere-tree construction algorithm does aim to minimise the error in the approximation, but it is limited in how it reduces the set of spheres i.e. it is unable to adjust the other spheres in the set. Thus it does not distribute the error among the remaining spheres, which would reduce the overall error. Chapter 5 presents a number of improvements to this algorithm.

Chapter 5

Improved Medial Axis Method

In Chapter 4 the medial axis method for the construction of sphere-trees was introduced. The method constructs a Voronoi diagram from a set of sample points distributed across the surface of the object. The Voronoi vertices that are inside the object are subsequently used to construct a base set of spheres, approximating the object, from which a hierarchy is constructed. This chapter discusses a number of improvements that can be made to this algorithm.

5.1 Adaptive Sampling

The first phase of the medial axis method aims to generate a set of sample points that can be used to construct the medial axis approximation. In order to generate the correct medial axis, these points need to be distributed evenly across the surface of the object. While this is very simple for some shapes, it is quite complex for general polyhedral meshes [49]. Hubbard first assigns each triangle in the mesh a number of points based on its area. A relaxation algorithm then tries to redistribute these points more evenly across the surface. Hubbard employs a heuristic to simplify this process. He groups the triangles into planar groups called clusters and applies a separate relaxation to each.

While this does eliminate the problem of points leaving the surface, there are a number of difficulties associated with this process. For highly curved objects, such as those modelled with NURBS (Non-Uniform Rational B-Splines), it is rare to find planar areas. Thus the clusters usually consist of only one triangle. This greatly reduces the effectiveness of the relaxation technique. There are also a number of numerical tolerances required to perform the relaxation, which can be hard to determine.

A simpler method, which does not require any numerical tolerances, is to sample the cluster/triangle in a grid-like fashion, as shown in Figure 5.1. This scheme will always yield an even arrangement of points within an individual triangle/cluster. However, it may use more points than were assigned to the triangle, which will increase the computational costs, but will not make the medial axis any less correct.

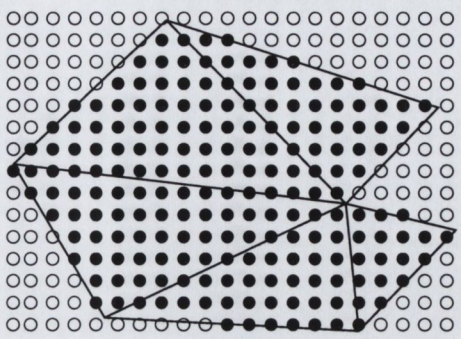
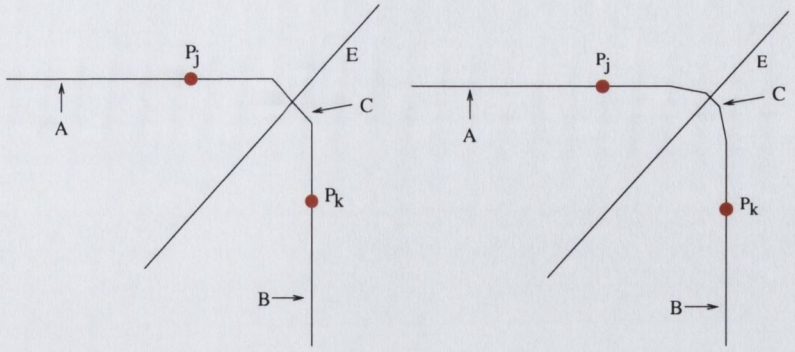


Figure 5.1: Distributing points within a cluster by sampling the clusters bounding box in a raster fashion.

As it is extremely difficult to determine how many samples should be used for a given surface, Hubbard uses a post-processing step to add extra samples to correct problems with the medial axis. The process, described in Section 4.2, looks for gap crossing cells, which are cells whose faces intersect areas of the surface that do not contain the points used to create them. Extra points are added to correct these errors. While this does correct problems with the medial axis, it does not make any guarantees about how well the set of spheres will approximate the surface or even that the set of spheres will cover the entire surface. Thus it may not be possible to approximate the object to the desired accuracy. Another problem with this strategy is in the way it determines when new points are required. Figure 5.2 shows an example of where the algorithm will choose to add extra points when it is not really necessary as there is no error in the medial axis. In the situation shown in Figure 5.2(a), the Voronoi edge E intersects cluster C but is not considered gap crossing as P_j and P_k are on clusters A and B, which are neighbours of C. Thus no extra points will be inserted into the Voronoi diagram. However, in the second situation (shown in Figure 5.2(b)) there is a small face between A and C and another between B and C. This means that E is now considered gap crossing (as it intersects a cluster that is not a neighbour of A or B). This will cause the algorithm to add extra points to correct the medial axis. However, it is questionable as to whether there is indeed a problem with the medial axis at this point. While the medial axis will be affected by the change in the surface, this change will not significantly affect the resulting set of spheres. Therefore the addition of the extra point will cause an unnecessary overhead. For complicated meshes, which contain areas with many small polygons, this situation will occur quite frequently.

There are a number of alternative algorithms, for the computation of Voronoi diagrams, that could be used for the construction of the medial axis [23, 24, 25, 59, 72]. However, none of these algorithms solve the problem of knowing where to place the sample points. Thus an incremental algorithm remains preferable as it allows iterative improvement of the approximation. Generating a good set of spheres is more important than accurately approximating the medial axis. Thus an adaptive sampling technique, which iteratively improves the set of spheres, is employed. Each medial vertex will be used to construct a



(a) E not gap-crossing (b) E is gap-grossing

Figure 5.2: Small clusters can cause faces that do not represent a problem to be considered gap-crossing.

sphere whose radius will be the distance from the vertex to sample points used to create it. A new point, which will cause v to be replaced, is inserted into the Voronoi diagram, as illustrated in Figure 5.3. For a convex body, if q is the point on the surface that is closest to the center of the sphere (v) then a measure of how far the sphere goes outside the surface is given by:

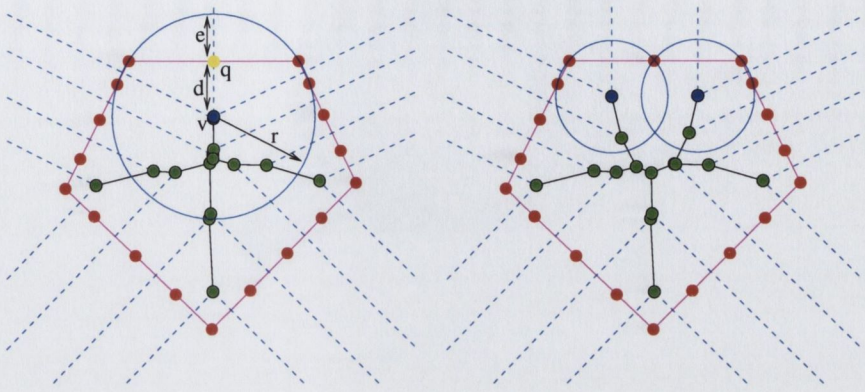
$$e = r - \|q - v\| \tag{5.1}$$

where :

- e is the distance the sphere protrudes past the surface,
- r is the radius of the sphere,
- v is the vertex at the center of the sphere,
- q is the closest point on the surface.

Inserting a new sample point at q will guarantee a reduction of e. The insertion of the new point results in the vertex v being removed and replaced by new vertices to form a Voronoi cell around q. The new vertices that lie inside the object will then create new spheres. This is illustrated in Figure 5.3(b).

Section 4.2 presented two problems that Hubbard's algorithm aims to fix: (1) where there is a break in the medial surface; and (2) where the medial axis leaves one part of the object and re-enters another part. The adaptive sampling scheme will fix the latter problem, if it results in a large sphere joining two parts of the object, as this sphere will be divided if e is large enough (Figure 5.4). Figure 5.5 shows a comparison between the adaptive sampling algorithm, shown as Algorithm 3, and the regular algorithm. The adaptive algorithm, which started with a set of *circa* 500 spheres produced from the non-adaptive algorithm, clearly produces a closer approximation of the object, with the non-adaptive algorithm producing a very uneven and bumpy result.



(a) Before (b) After

Figure 5.3: Addition of a new point to reduce the error of the approximation. The point (q) is positioned to improve a specific part of the approximation.

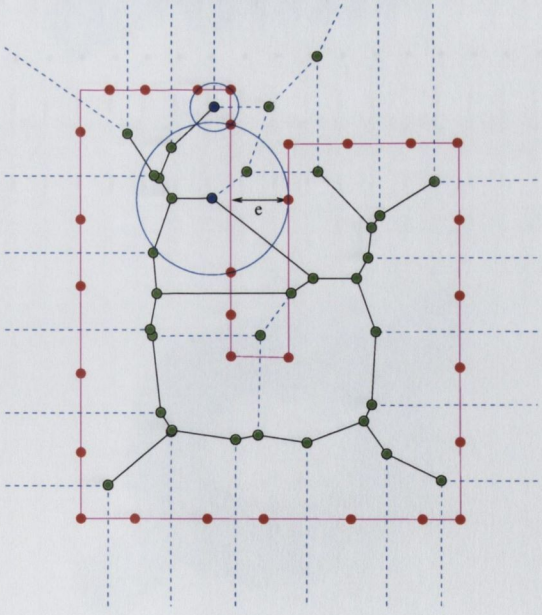
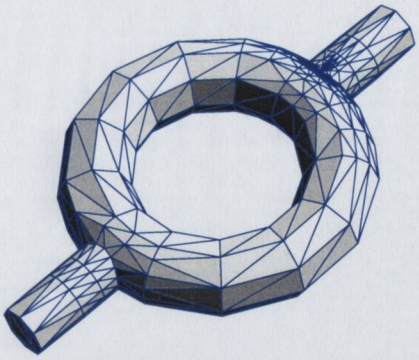
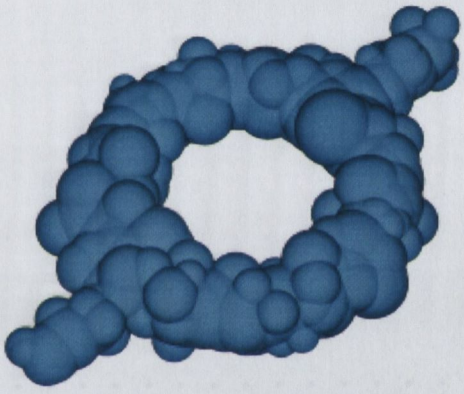


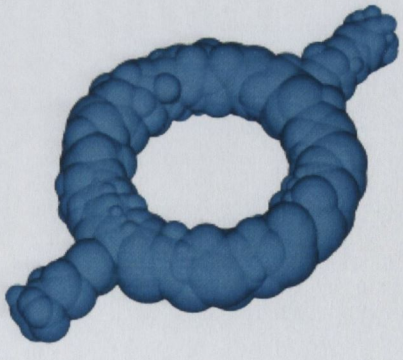
Figure 5.4: Where the medial axis crosses from one part of the object to another, large resulting spheres will be divided by the adaptive sampling algorithm if e is larger than the desired accuracy.



(a) Model



(b) Non-Adaptive



(c) Adaptive

Figure 5.5: Comparison between non-adaptive and adaptive sampling, both using *circa* 1000 spheres.

Algorithm 3 Adaptive construction of Voronoi Diagram

Input : Voronoi diagram, V. Surface, S.
Maximum allowable error, $maxErr$.
Output : Updated Voronoi diagram, V.

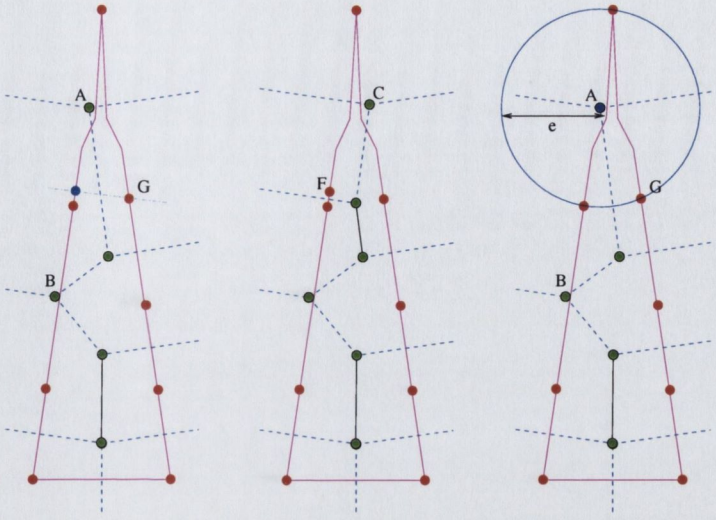
ADAPTIVEVORONOI*(S, V, maxErr)*
  $M \leftarrow$ set of medial vertices in V
  $v \leftarrow$ vertex from M with worst fitting sphere
  while $Error(v) > maxErr$ **do**
    $p \leftarrow$ point on S closest to v
    insert p into Voronoi diagram, V
    $M \leftarrow$ updated medial vertices (excluding v)
    $v \leftarrow$ vertex from M with worst fitting sphere
  end while

5.2 Complete Coverage

When constructing spheres using the medial axis it is difficult to ensure that the entire object is covered with spheres. Figure 5.6 shows an example of a problem that can occur. The thin nature of the shape means that there is a very small area in which the Voronoi vertices must lie in order for them to produce spheres. Figure 5.6(a) shows two medial vertices, labelled A and B, which are just outside the object. These vertices are not used to construct spheres and therefore the tip of the spike is left uncovered.

Hubbard's notion of gap-crossing cells may be employed to try to fix this problem. According to the strict definition both these vertices are caused by gap-crossing cells. Therefore additional sample points could be generated to handle the problem. Figure 5.6(b) shows the Voronoi diagram resulting from the addition of one extra sample point, F (created by projecting G onto the opposite surface). As the object narrows sharply, this has resulted in the medial axis being pushed out through the opposite surface of the object. Once again the tip of the spike is left uncovered. While this algorithm may eventually add enough sample points to fix the problem it may take many iterations. An extension to the adaptive sampling algorithm allows us to deal with this situation quite easily by using one of the external vertices to construct a sphere, Figure 5.6(c).

As before, it is useful for us to concentrate on what we are trying to achieve when using the medial axis method. The aim is to produce a set of spheres that approximate the object to a high degree of accuracy. Strictly speaking, the medial axis is not allowed to contain any vertices that are outside the object. However, the use of some such vertices allows us to guarantee that the entire surface will be covered. Rather than considering the polyhedral model of the object, it is more efficient to consider a set of points distributed across the surface. This set could be the samples used to construct the medial axis but a



(a) A (b) B (c) C

Figure 5.6: An example of a situation where it is very difficult to produce spheres to cover the object's entire volume.

more densely packed set of points would be preferable as it will cover the surface better.

In order to guarantee that all the points on the surface are covered, it is necessary to add extra spheres that will cover those not already covered by the medial spheres. For any such point there will be a number of vertices that can produce a sphere capable of covering it. If we are using the forming points of the Voronoi diagram, it would make sense to limit ourselves to choosing a sphere centered around one of the vertices surrounding that forming point. This would limit the computational complexity and confine the search to vertices that are in the region close to the object and therefore close to the medial axis. As it is more economic to use a sparse sampling for the construction of the medial axis, the coverage points may not be forming points. However, as the Voronoi cell around a forming point will have a number of vertices, which will make spheres that touch the forming point, there will be at least one sphere which can cover each point within the Voronoi cell, as shown in Figure 5.7.

Thus to cover a previously uncovered point, only the vertices of the surrounding Voronoi cell need to be considered. As the Voronoi cell represents the region of space that is closer to its forming point than any other, a point will be contained in the cell of its closest forming point. There will be a number of vertices associated with the cell, each of which may be capable of covering the uncovered sample. There are a number of criteria by which the sphere can be chosen, such as using the smallest sphere, the sphere that is closest to being inside the object or the sphere that covers most of the uncovered samples.

As the surrogate vertex (i.e. the one chosen to improve coverage) will not be contained inside the object, it is desirable that we minimise the inaccuracies that may be introduced.

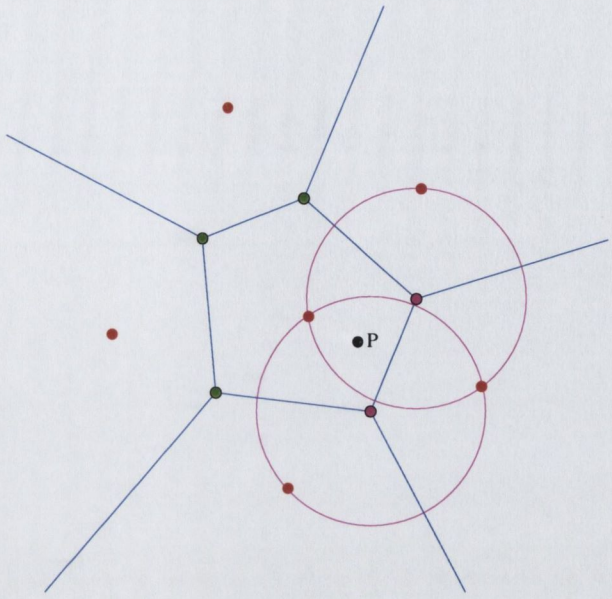


Figure 5.7: Every point P within a cell will be covered by at least one of the spheres created from the vertices of that cell as all such spheres will pass through the forming points of the cell.

The adaptive sampling algorithm holds the key to choosing which vertex should be used for this purpose. To avoid introducing any artifacts, due to the selection of these vertices, we should choose a vertex that will be removed during later iterations of the adaptive sampling algorithm. Thus it seems like a good choice to use the vertices that will be most easily removed during the next iteration of the adaptive sampling algorithm. Such vertices can be easily chosen using the measure of deletability discussed in Section 4.1.3, which ranks the deletability of a vertex as distance to its forming points minus the distance to the new point. The sphere produced by using this metric may be large, as seen in Figure 5.8, but this simply means that it will be removed as the algorithm proceeds. As the adaptive sampling algorithm attempts to improve only those spheres that poorly approximate the surface of the object, it will stop trying to improve the extra spheres once they become small. Therefore, the number of extra samples needed to ensure coverage should be much lower than using the notion of gap-crossing cells. This is because the algorithm allows the medial axis to leave the object provided it generates a set of spheres to approximate the object fully. Figure 5.9 shows a section of a model that contains a number of very thin areas. The figure also shows the set of spheres generated using the adaptive algorithm with the coverage check, presented as Algorithm 4, and the non-adaptive algorithm. It is clearly visible that using the adaptive algorithm with extra "coverage" spheres, gives a more complete representation of the object. There are still a few small areas that are not completely covered as the algorithm only guarantees the sample points. However, these should not cause significant problems as they are small and a more dense sample set could be used if desired.

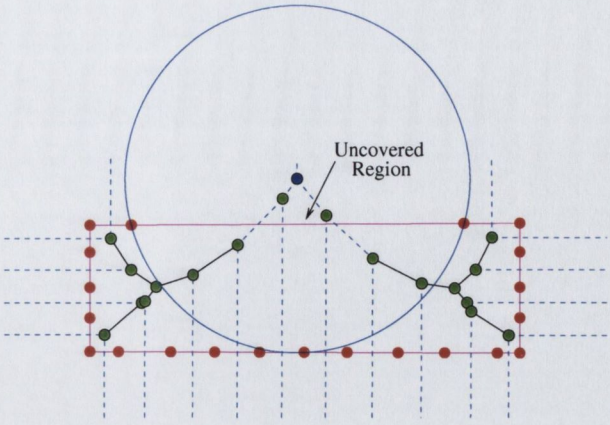
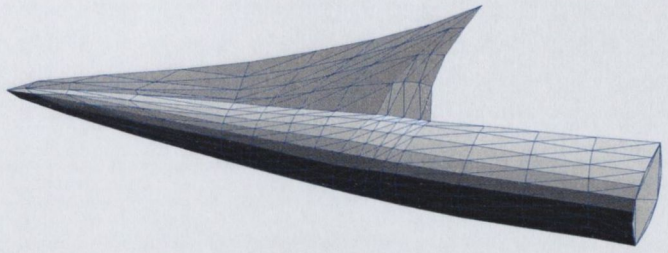


Figure 5.8: When surrogate spheres are selected it is often desirable to replace them as soon as possible to reduce the impact they have on the sphere set.

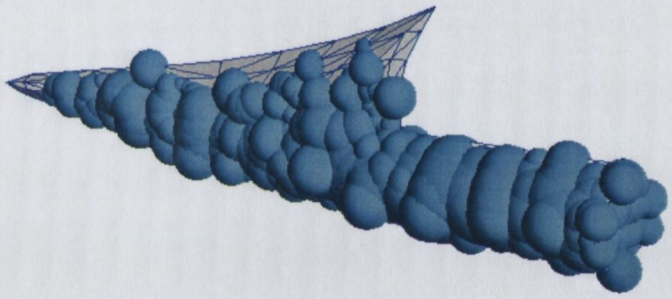
Algorithm 4 Pseudo-medial sphere selection algorithm

Input : Voronoi diagram, V. Surface, S. Surface points, P.
Output : Set of pseudo-medial spheres, M.

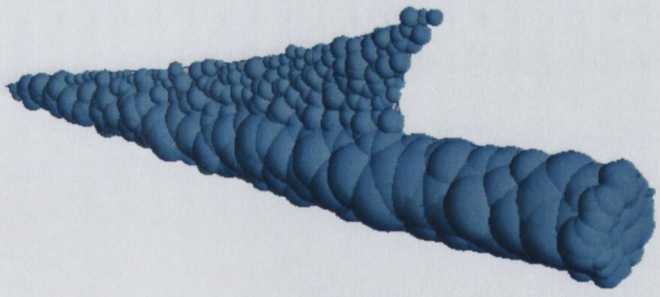
```
COVERAGESPHERES(M, V, P)
  M ← {}                                              {empty set}

  for all vertex v ∈ V do                             {pure medial vertices}
    if v is inside surface S then
      M ← M ∪ {v}
    end if
  end for

  for all p ∈ P do                                    {cover uncovered points}
    if p not covered by M then
      t ← NIL                                         {best sphere to cover point}
      C ← cell from V containing p
      for all vertex v ∈ C do
        s ← sphere around v
        if s contains p and s is more deletable than t then
          t ← s
        end if
      end for

      M ← M ∪ {t}                                     {add to medial set}
    end if
  end for
```



(a) Model



(b) Non Adaptive



(c) Adaptive (with coverage spheres)

Figure 5.9: An example of how the use of vertices from outside the object can help to ensure that the object is more completely covered with spheres.

5.3 Sphere Reduction

When constructing the set of spheres to create a section of a sphere-tree, it is necessary to reduce the set of medial spheres so that it contains the number of spheres required for the sphere-tree. Section 4.3 described an algorithm that achieved this by using a merging strategy. This was a greedy algorithm that iteratively merged pairs of spheres until the desired number of spheres were left. At each iteration, the pair whose combined sphere had the lowest error was chosen. This does not necessarily lead to the lowest error at the end of the process as no account is taken of the effect the merge will have on the final result. Also, when spheres S_i and S_j are merged to create the combined sphere S_{ij}, none of the other spheres are affected. This can lead to sets of spheres that do not distribute the error between them. Figure 5.10 illustrates this in 2D, where the sphere that resulted from the merge has a much higher error than any of the other spheres. There are a number of alternatives to this scheme that may yield better results.

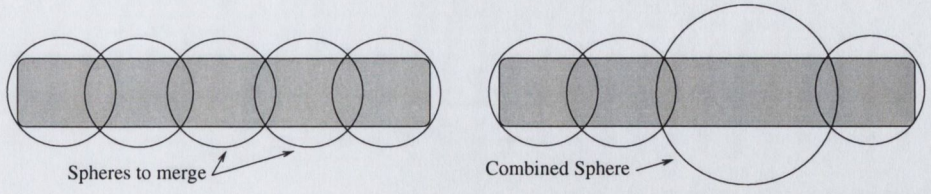
5.3.1 Improved Merge

A number of minor improvements can be made to the merging strategy used by Hubbard. When combining a pair of spheres, Hubbard used Ritter's approximate bounding sphere algorithm to construct the new sphere [89]. We favour a more accurate method and thus use White's minimum enclosing ball algorithm [109]. Also, we ensure that every sphere is capable of taking part in at least one merge. Any spheres that end up with no neighbouring spheres are given an artificial set of neighbours, consisting of any spheres it overlaps. In the event that this results in an empty set of neighbours, the sphere is made a neighbour of all the remaining spheres in the set. Also, when the number of spheres reaches a sufficiently low number every pair of spheres is considered to be mergeable. This is typically done when the number reaches 2 or 3 times the target number of spheres.

Finally, special consideration is given to merges that actually reduce the error in the approximation, i.e. "beneficial merges". As an approximation is only as good as it's worst error, we favour merges that improve the worst spheres in the approximation. We do not treat other beneficial merges as a special case as we have found that this can adversely effect the final results.

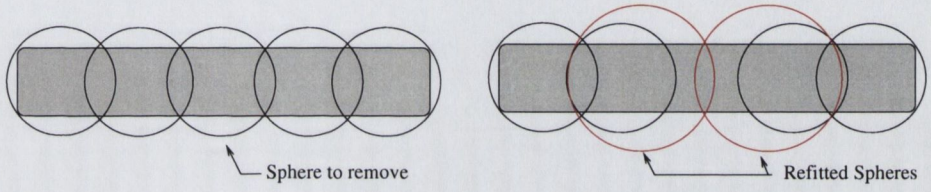
5.3.2 Sphere Bursting

Another way to improve upon the sphere merging is to allow the reduction of the sphere set to have a much more global effect on the remaining spheres. Merging two spheres will produce one larger sphere, as illustrated in Figure 5.10. The effect of reducing the number of spheres is localised to the combined sphere. Removing (or bursting) one of the spheres and allowing some of the other spheres to collectively cover the newly uncovered region will better distribute the error introduced by decreasing the number of spheres. Figure 5.11(b)



(a) Before (b) After

Figure 5.10: Merging two spheres together leaves the other spheres unchanged and therefore can result in a poor approximation.



(a) Before (b) After

Figure 5.11: Removing a sphere and allowing the surrounding spheres to cover the newly uncovered area.

shows the result of removing one of the spheres from the set shown in Figure 5.10(a). In this case, two of the spheres absorb the increase in error yielding a better final result.

When removing a sphere, conservative coverage of the surface must be maintained. As with the merge algorithm, using a set of surface points is more efficient than working with the actual polygons. The removal of a sphere will leave a number of the sample points uncovered. These points must be covered by the remaining spheres in the set. The algorithm for achieving this, presented as Algorithm 5, must distribute the uncovered points between the remaining spheres so as to limit the error introduced. Each newly uncovered point must be assigned to one of the remaining spheres. As there can be a large number of spheres left in the set, determining the optimal sphere to reassign each point to is very computationally expensive. A simpler approach is used. To try to minimise the increase in size of the remaining spheres, each point is assigned to the sphere that is closest to covering it. Finding the sphere to which the point is closest involves measuring the distance from the point to the shell of the sphere :

$$D(S, P) = \|S_c - P\| - S_r \tag{5.2}$$

where :

$D(S,P)$	is the distance from sphere S to point P,
S_c	is the center of sphere S,
S_r	is the radius of sphere S.

The value of $D(S,P)$ will become negative if the point P is already contained within the sphere S, which indicates that the point should definitely be assigned to that sphere. Of course, the point may already by assigned to that sphere in which case no further work needs to be done. Finally, each sphere that is assigned new points must be updated so that it covers the new set of points.

The burst algorithm can thus be used to reduce the set of spheres, created from the medial axis, down to a specified number of spheres. This algorithm operates in a similar fashion to the merge algorithm, removing the sphere that will introduce the least error at each iteration. The effects of removing each of the spheres can be pre-computed and updated whenever a sphere receives new points. Again special consideration is given to sphere removals that actually improve the fit of the spheres involved. Choosing the sphere that gives the biggest decrease in error before those that introduce error allows the algorithm to provide tighter fitting sets of spheres. Figure 5.12(a) shows a typical arrangement of 9 spheres approximating a cube. To further reduce the set, if required, the merge algorithm would combine two of these sphere together, producing something like Figure 5.12(b). However, the burst algorithm will produce a much nicer arrangement of spheres, as featured in Figure 5.12(c).

Hubbard used Ritter's approximate bounding sphere algorithm [89], to fit a sphere around each set of points. This algorithm can often yield rather poor fitting spheres. In order to maintain tight fitting approximations, a minimum volume bounding sphere algorithm, such as those presented in [29, 108, 109], can be used. However, as illustrated in Figure 5.13, the smallest sphere that bounds a set of points may not always represent the one with the best fit to the surface. This happens particularly when the points all lie on the same side of the object. It is very computationally inefficient to try to fit the minimum error sphere every time a new bounding sphere is required, i.e. when evaluating the effects of removing a sphere. A more efficient method is to use either the minimum volume bounding sphere or the original sphere enlarged to cover all the points, whichever is tightest fitting. This keeps the sphere near the medial axis unless it is beneficial to move it.

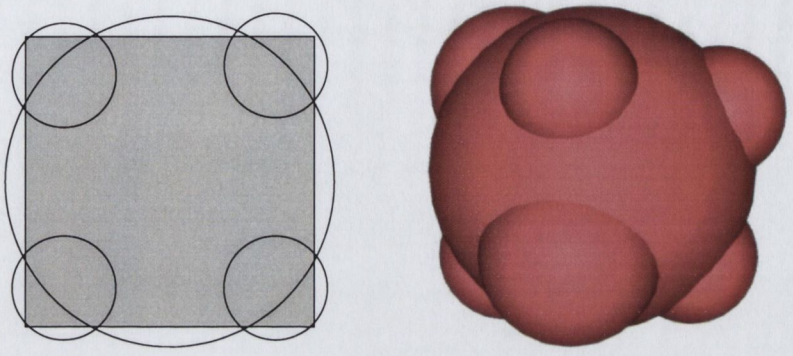
5.3.3 Expand & Select

The burst algorithm for sphere reduction was designed to allow the error introduced by the removal of a sphere to be distributed amongst some of the remaining spheres. While there are a number of situations where this will improve on the merge algorithm, it allows

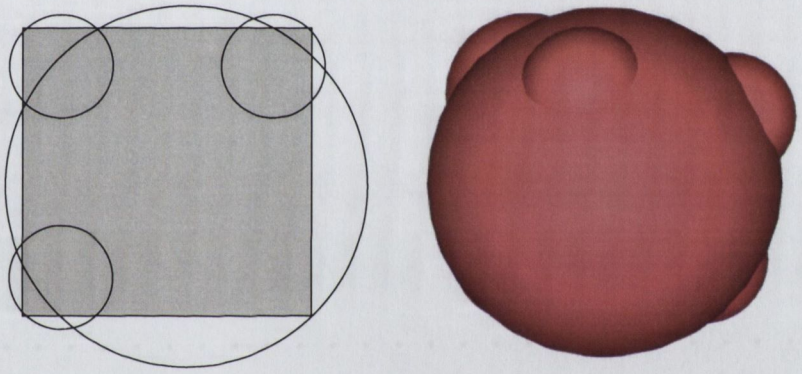
Algorithm 5 Removal of a sphere

Input : Set of spheres, S. Surface points covered by each sphere, *Points*.
Sphere to be removed, *remSph*.
Output : Updated set of sphere and points assigned to them.

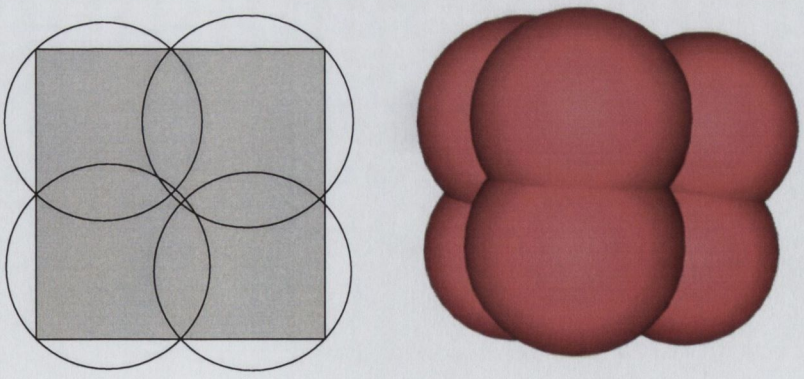
```
REMOVESPHERE(S, Points, remSph)
  S ← S − {remSph}                                  {remove sphere from set}

  Update ← {}
  for all p ∈ Points[remSph] do                     {reassign points}
    bestD ← ∞
    bestS ← NIL
    for all s ∈ S do
      if D(p, s) < bestD then
        bestS ← s                                   {closer sphere}
        bestD ← D(p, s)
      end if
    end for

    if p ∉ Points[bestS] then
      Points[bestS] ← Points[bestS] ∪ {p}           {assign point to sphere}
      Update ← Update ∪ {bestS}                     {flag for update}
    end if
  end for

  for all s ∈ Update do                             {update spheres}
    S[s] ← BOUNDINGSPHERE(Points[s])
  end for
```



(a) Initial

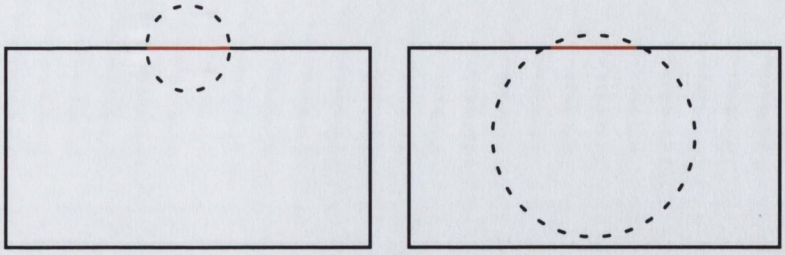


(b) Merge



(c) Burst

Figure 5.12: Comparison of merging and bursting a cube approximated by 9 spheres (2D equivalent shown on left).



(a) Minimum Volume (b) Low Error

Figure 5.13: The minimum volume sphere may not always represent the sphere with the minimum error.

only the neighbouring spheres to absorb the increase in error. The greedy nature of the algorithm means that it can result in sets of spheres with a large worst error, i.e. the worst sphere in the approximation may be much worse than the others. A much better strategy is to select a set of spheres, containing the required number, which distribute the error evenly between them.

If the error within the reduced set of spheres is perfectly distributed across the object, each sphere will have the same error associated with it. By ensuring that all the spheres have the same error, the algorithm will have a better chance of achieving a tight fit. Equation 5.1 gave us a metric to measure the distance from a sphere to the surface of a convex object. This equation can be rearranged to allow us to compute the radius, for a given sphere, so that it will hang out over the surface by at most e :

$$r = e - \|q - c\| \tag{5.3}$$

where :

- r is the radius of the sphere,
- e is the distance the sphere protrudes past the surface,
- c is the center of the sphere,
- q is the point on the surface.

This equation allows us to expand the spheres in the medial set so that they all hang over the surface by the same amount, as shown in Figure 5.14. Constructing the reduced set of spheres that all have the same *stand-off distance* (e) will potentially give a more consistent approximation. This has the potential to reduce the error of the worst spheres in the approximation. To achieve this, the medial spheres can be expanded to the desired standoff distance, using Equation 5.3, and a sub-set of the spheres selected. When the object is not convex, the stand-off distance represents an over-approximation of the error present in the approximation.

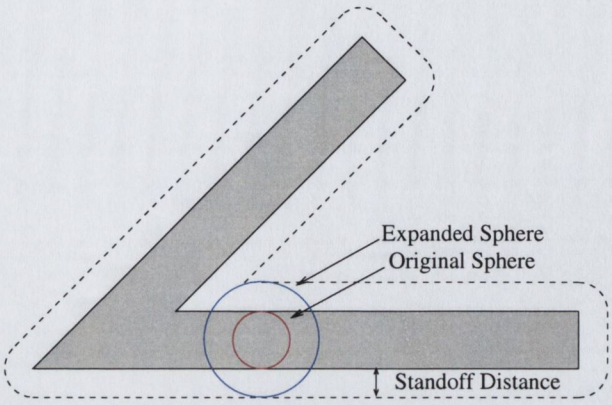


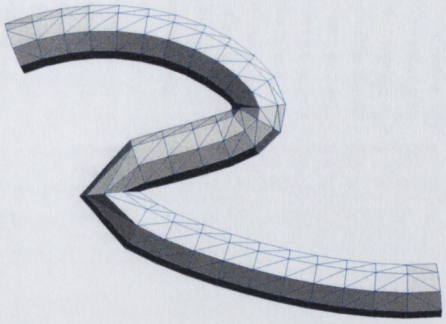
Figure 5.14: Expanding a sphere to have a given stand-off distance.

It is not possible to pre-determine the value for e that will result in the required number of spheres being selected. In order to construct a set of spheres containing a certain number of spheres, we must search for the correct value of e. A simple search algorithm, derived from the Binary Search, can be used to achieve this. For each value of e, the spheres are expanded to have the required stand-off distance and the redundant spheres are eliminated. The algorithm looks for the lowest value of e that results in a set containing an allowable number of spheres. Figure 5.15 shows an example where the algorithm is looking for the lowest value of e that requires 9 spheres to approximate an 'S'-shaped object. When the value of e is too low, Figure 5.15(b), the algorithm needs more then the desired number of spheres to cover the object and so increases e. When the value of e is large, Figure 5.15(b), the entire object is enclosed and the value of e can be reduced. The search algorithm maintains upper and lower bounds for e and generates a set of spheres using a value for e that is the mid-point of this interval. Depending on the number of spheres required to cover the object, the algorithm adjusts either the upper or lower bound to narrow the interval. After each iteration the size of the interval has been halved and so the minimum value of e can be found quickly.

Selecting The Set of Expanded Spheres

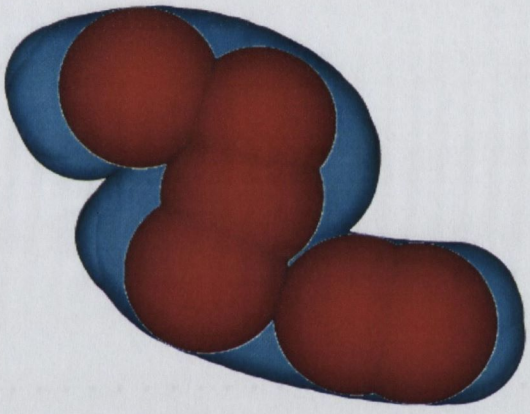
The job of selecting the minimum number of expanded spheres that cover an object is a complicated one. As the set of spheres from which the reduced set is drawn is potentially quite large, it would be very expensive to try every combination of spheres. Instead of looking for this global optimum we can try to find a good minimal set of spheres. A minimal set will be a set from which none of the spheres can be removed without exposing part of the surface.

A greedy algorithm allows us to choose the set of spheres without having to evaluate a large number of combinations. In this algorithm the set of currently selected spheres is maintained. Successive spheres are chosen from the remaining spheres until the desired region of the object is completely covered. In order to decide which sphere to add, each

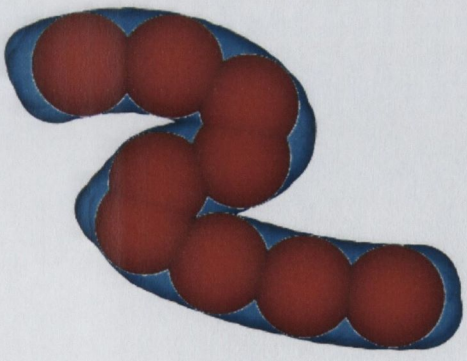


(a) Model

(b) e too small

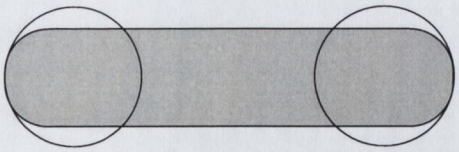


(c) e too large

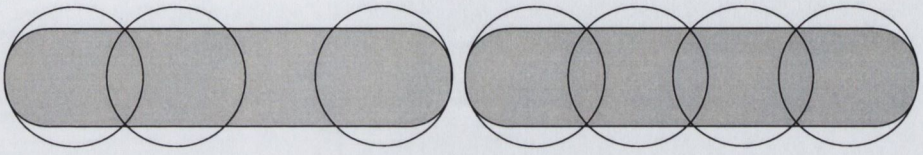


(d) e just right

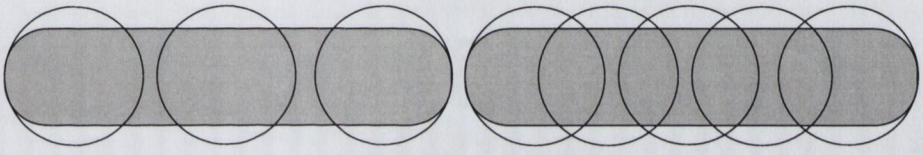
Figure 5.15: Varying the stand-off distance to create an approximation with a given number of spheres (expanded medial spheres are in blue, selected spheres overlayed in red).



(a) First Two Spheres



(b) Third Sphere (Good Choice)



(c) Third Sphere (Bad Choice)

Figure 5.16: When selecting a set of spheres to cover the object, a bad choice of spheres may result in gaps being formed, which will require extra spheres to fill them.

sphere must be ranked according to its potential to keep the set of spheres small. The first obvious choice would be to rank each candidate by the area of previously uncovered surface contained within it. This will allow the algorithm to cover the largest amount of the object at each iteration. As with all greedy algorithms, this aims to make the biggest gain at each stage but does not guarantee to find the global optimum.

However, this heuristic suffers from one major drawback. Consider how the algorithm will choose spheres to approximate a cylinder with rounded ends. As shown in Figure 5.16(a), the first two spheres will be chosen to cover the ends. The problem arises when choosing the third sphere. There exists a large number of candidate spheres, along the remaining section of the surface, that all have the same ranking. Thus, it is possible to choose a sphere next to one of the first two spheres or one that is towards the middle. The first option, illustrated in Figure 5.16(b), will require one additional sphere to cover the object. However, the second option, shown in Figure 5.16(c), will require two more spheres.

The aim is to select as few spheres as possible and to cover the entire object using the specified stand-off distance. Another way of approaching this is to rank the spheres by the number of other spheres it makes redundant. Ranking the spheres in this way will tend to select spheres that cover complicated areas of the object first, especially when the

set of spheres has been constructed using adaptive sampling. It will also tend to choose spheres that cover areas near previously selected spheres. The previously selected spheres will partly overlap a number of remaining spheres, which will be made redundant. These spheres will increase the rank of the spheres that overlap them. This will in turn favour the configuration featured in Figure 5.16(b).

The "Expand" algorithm, presented as Algorithm 6, is a greedy algorithm in all senses of the word. It aims to cover as much of the object as possible at each stage. Both the heuristics presented tend to produce large spheres initially and smaller ones later on. For a cube, this type of scheme will result in a large sphere in the middle with smaller ones at the corners. This is similar to the "Merge" algorithm. As the stand-off distance is increased, the center sphere grows until it reaches a point where the corner spheres become redundant, see Figure 5.17. The corner spheres also grow and they eventually make the center sphere redundant. If the algorithm chooses the largest sphere at each state, the center sphere will be chosen before the corners. So instead of choosing the corner spheres, which are sufficient to cover the object, the large central one will be chosen first and the corner spheres will still be required to complete the representation. When the stand-off distance is increased to the point where the corner spheres are no longer required, the algorithm will only choose the one large central sphere. This will prevent the algorithm from progressing as it will continue to produce a single sphere and the approximation will never improve. A simple way to combat this problem is to repeat the selection without allowing it to choose the highest ranked sphere previously chosen. This will allow the algorithm to choose the larger set of spheres, which will divide the object into more regions for further refinement.

5.4 Eliminating the Medial Axis

The "Expand" algorithm, for reducing the set of medial spheres into a set suitable for use in a sphere-tree, aimed to reduce the error in the approximation by distributing the error evenly between all the spheres. The idea of stand-off distances can be used to generate similar approximations without the construction of the medial axis approximation.

Generating such a sphere can be expressed as a constrained optimisation problem. Starting with one of the spheres already generated, a new sphere can be created using an optimisation algorithm to choose the position of the sphere that makes it cover as much of the uncovered surface as possible for the desired stand-off distance. Algorithm 7 details the SPAWN algorithm[1], which grows a new sphere from each of the existing spheres and chooses to keep the one that covers the most points. When no new sphere can be created a sphere is constructed in the region of one of the uncovered points. The algorithm uses an optimisation step to maximise the amount of the surface covered by each new sphere.

[1]The algorithm has been named SPAWN as each sphere grows from its predecessor.

Algorithm 6 Select spheres using Expand.

Input : Medial spheres, S. Points to cover, P.
Maximum allowable error, $maxErr$.
Output : Selected spheres, T.

EXPAND*(T, S, P, maxErr)*

for all $s \in S$ **do** {*expand spheres*}
  $c \leftarrow$ center of sphere s
  $p \leftarrow$ point on the surface which is closest to c
  $s_r \leftarrow maxErr - \|p - c\|$
end for

for all $s \in S$ **do**
  $Points[s] \leftarrow \{\}$ {*list points in sphere*}
  for all $p \in P$ **do**
    if s contains p **then**
      $Points[s] \leftarrow Points[s] \cup \{p\}$
    end if
  end for
end for

while some of P is not covered **do** {*select spheres*}
  $bestS \leftarrow NIL$
  $bestCount \leftarrow 0$
  for $s \in S, s \notin T$ **do**
    $count \leftarrow \|Points[s]\|$
    if $count > bestCount$ **then**
      $bestS \leftarrow s$ {*higher ranked sphere*}
      $bestCount \leftarrow count$
    end if
  end for

  $T \leftarrow T \cup \{bestS\}$ {*add bestS to the set*}

  for all $p \in Points[bestS]$ **do**
    for all $s \in S$ **do**
      if $p \in Points[s]$ **then**
        $Points[s] \leftarrow Points[s] - \{p\}$ {*update points lists*}
      end if
    end for
  end for
end while

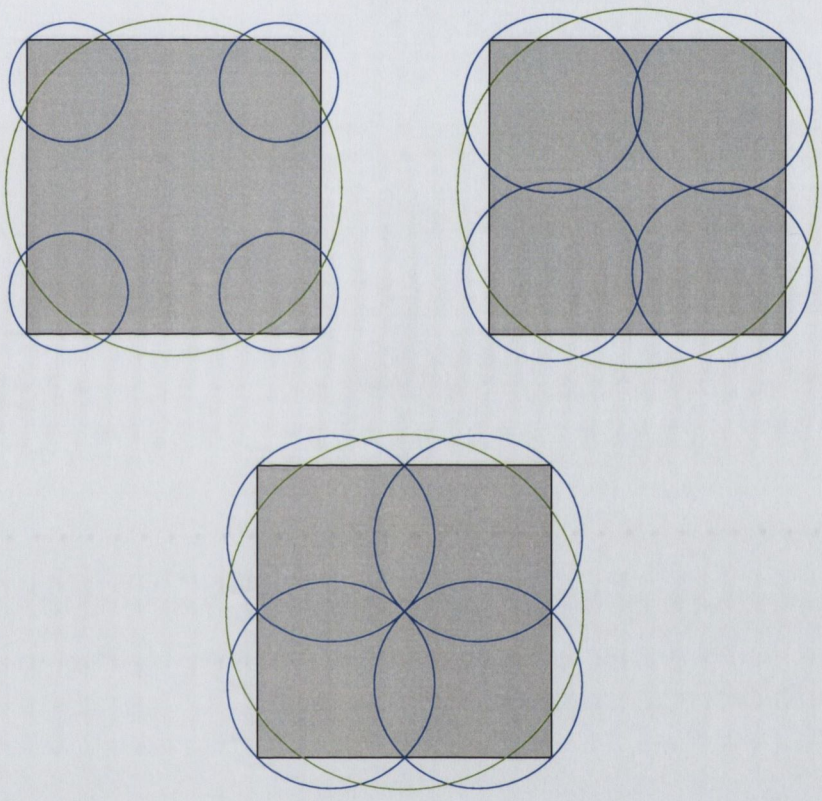


Figure 5.17: As the stand-off distance increases, the center sphere eventually makes corner spheres redundant. However they also grow and make the center sphere redundant.

As the optimiser manipulates the center of the sphere, the objective function computes the radius of the sphere and evaluates how much of the surface it covers. As with the "Expand" algorithm, this constructs a set of spheres that all have a desired stand-off distance and hence a search is needed if a specific number of spheres is required.

5.5 Conclusions

This chapter has focused on improving the medial axis method for generating sets of spheres to approximate rigid objects. This method used the medial axis as a basis for the construction of a base set of spheres from which the sphere-tree can be constructed. The biggest difficulty with this method is the generation of a reasonable set of surface points for constructing the Voronoi diagram that approximates the medial axis. The emphasis was shifted from constructing an accurate medial axis to forming a set of spheres that tightly approximate the object. An adaptive sampling technique was presented, this ensures that the set of spheres approximates the object to a given level of error by improving the worst areas of the approximation first. While this does not necessarily give a good medial axis, it provides a good set of spheres from which to construct the sphere-tree. The adaptive sampling algorithm also uses extra spheres, which lie on non-medial vertices of the Voronoi diagram, to ensure that the surface of the object is entirely covered. Thus the adaptive algorithm does not require any initial samples to be generated as the object will be fully contained within the initial Voronoi diagram. This will provide enough coverage for the adaptive sampling algorithm to operate.

We also examined methods for reducing the number of spheres required to cover the object. This is necessary for constructing successively tighter sets of spheres in the sphere-tree. Hubbard's algorithm used a merging strategy that was unable to distribute the introduced error among the remaining spheres. Thus the resulting spheres could potentially contain poorly fitting spheres. A number of alternative schemes were presented. These aimed to improve the set of spheres generated from the set of medial spheres by allowing the error to be distributed more evenly across the object. The Expand algorithm produces sets of spheres that all have the same stand-off distance. This results in spheres that have the same error if the body is convex and have very similar errors otherwise. This is a desirable property for collision response as lumpy approximations may lead to odd looking collisions. This algorithm uses a heuristic to choose a set of spheres to cover the object, with the aim of minimising the number of spheres required. If a specific number of spheres is required, a search algorithm is needed to find the stand-off distance that results in the specified number. Finally this idea was used to develop an optimisation based alternative to using the medial axis. This algorithm operates much the same as the Expand algorithm but doesn't require an approximation of the medial axis to be constructed.

Algorithm 7 SPAWN sphere generation

Input : Set of points to cover, P. Maximum allowable error, $maxErr$.
Output : Set of spheres, S.

SPAWN*(S, P, maxErr)*

```
while there are points in P do
  p ← any point from P                                   {make initial sphere}
  s ← sphere that covers most points in P, with stand-off
      distance maxErr, using p as an initial guess       {make sphere}
  S ← S ∪ {s}

  repeat
    bestS ← NIL                                          {spawn new spheres}
    bestCount ← 0
    for all s ∈ S do
      s1 ← sphere that covers most points in P, with stand-off
           distance maxErr, using sc as an initial guess

      count ← number of points from P that are within s1
      if count > bestCount then
        bestS ← s1                                       {new best sphere}
        bestCount ← count
      end if
    end for

    if bestCount ≥ 1 then
      S ← S ∪ {s}                                        {add sphere to set}
      for all p ∈ P, p inside s do
        P ← P − {p}                                      {point is now covered}
      end for
    end if
  until bestCount == 0
end while
```

Input : Center of the sphere to evaluate, c. Spheres generated so far, S.
Points to cover, P. Maximum allowable error, $maxErr$.
Output : Measure of how much new surface the sphere covers, *count*.

SPAWNOBJECTIVE*(c, S, P, maxErr)*

```
p ← closest surface point to c
sc ← c                                                   {make sphere}
sr ← maxErr − ||p − c||

count ← number of points from P inside s                 {evaluate metric}
return count
```

Chapter 6

Improved Sphere-Tree Construction

Chapters 3 - 5 critically analysed the Octree and Medial Axis methods for the construction of sphere-trees. A number of improved algorithms were presented, each one identifying weaknesses in the existing algorithms and aiming to make various levels of improvement. The resulting algorithms are summarised as follows :

- **Grid :** this method of sphere generation is based on the generalisation of the octree method, detailed in Section 3.1. The algorithm places spheres in a grid-like arrangement but is allowed to choose the orientation, position, size and dimensions of the grid to best suit the object's geometry, see Section 3.3.

- **Merge :** similarly to Hubbard's original algorithm, the set of spheres from the medial axis approximation is reduced down to the specified number by successively merging pairs of spheres. A number of improvements have been made. Instead of using Ritter's approximate bounding sphere algorithm a more accurate algorithm is used. Also, special attention is given to merges that actually reduce the error in the approximation and to ensuring that each sphere always has neighbours with which to merge, see Section 5.3.1.

- **Burst :** the set of spheres produced from the medial axis is reduced, to contain the specified number of spheres, by successively removing (bursting) spheres. When a sphere is removed, the surrounding spheres must expand to cover the areas that have been left uncovered. This allows neighbouring spheres to collectively absorb the error introduced during the reduction. As with the merge algorithm, special attention is paid to sphere removals that actually reduce the error in the approximation, see Section 5.3.2.

- **Expand :** the medial spheres are expanded so that they all have the same stand-off distance from the surface. The set of spheres is then reduced to eliminate spheres

that are no longer required to maintain coverage of the object. This allows ALL the remaining spheres to collectively absorb the error in the approximation, see Section 5.3.3.

- **Spawn :** this is similar to the expand method except that the spheres are generated using an optimisation algorithm, which aims to cover as much of the object as it can with each sphere (for a given stand-off distance). This allows the algorithm more freedom in choosing spheres than the expand algorithm and does away with the need to construct the medial axis approximation, see Section 5.4.

Each of these algorithms were discussed in the context of approximating the object (or an area of it represented by a set of sample points) using a single set of spheres. The algorithms do not deal directly with the construction of a hierarchical representation of the object.

The job of constructing a hierarchy of spheres to represent the object can be expressed as an independent generic algorithm that relies on the presented algorithms to approximate the required areas of the object. The controlling sphere-tree generation algorithm dictates how the sphere-tree is constructed and how the object is divided at each level. The overall process can thus be broken up into a number of phases; the sphere-tree construction algorithm; the sphere generation algorithms (which are used by the sphere-tree construction algorithm) and; the sphere-set optimisation algorithms, which can be used to improve the fit of a set of spheres (generated by the sphere generation algorithms) prior to their inclusion in the hierarchy. This chapter deals with the higher level algorithms and presents generic sphere-tree construction and optimisation algorithms. A method for determining the appropriate number of children to assign each node is also presented.

6.1 Generic Sphere-Tree Construction

This algorithm represents a generic way of constructing sphere-trees using any of the previously presented algorithms, which fit a set of spheres to a section of the given object. The root of the hierarchy is always the smallest sphere that can enclose the object, which can be approximated using Ritter's algorithm [89] or computed more exactly using algorithms by Gärtner [29], Weltz [108] or White [109]. For successive levels, the algorithm controls the generation of a hierarchical representation of the object by partitioning it into a number of sections, each of which is covered by one of the spheres in the generated set. Each of these partitioned sections of the object will then be recursively approximated by a smaller sphere-tree, which will become one of the branches of the main tree. The first set of spheres must cover the entire object while the rest must cover sub-sections of the object. The individual sphere generation algorithm dictates how this is achieved.

For algorithms relying on the medial axis, the adaptive algorithm can be used to update the medial axis approximation so that it can provide a tight fitting set of spheres.

These sphere generation algorithms maintain a Voronoi diagram, which approximates the medial axis of the object. When the sphere generator is asked to approximate a region of the object, the Voronoi diagram is updated so that it contains a set of spheres that approximates the region to a desired level. This allows the medial axis to be updated so that the set of spheres, which will form the basis of the approximation, is of sufficient quality to allow a tight approximation. For this we require that the set of spheres contains some multiple of the target number of spheres and that the worst sphere in the set has an error that is a fraction (typically $\frac{1}{2}$ to $\frac{1}{4}$) of the parent sphere's error.

As the sets of spheres can be generated with a number of algorithms, we need to be able to determine the regions of the object covered by each sphere in an arbitrary set of spheres. The simplest method to achieve this would be to simply use any part of the object (or its surface) that is contained within the parent sphere. However, the spheres generated by the sphere generation algorithms could contain large areas of overlap inside the object. This is particularly true when trying to achieve tight fitting sets of spheres. Thus there will be large areas of the object that are shared between sets of children spheres. This would be very wasteful as the same area will be covered many times, which would further contribute to the overlap.

A more desirable situation is to divide the object into sub-regions with as little overlap as possible. This is achieved by dividing any overlapping regions between the spheres. In a region covered by a number of spheres, each part of the object need only be covered by one set of children spheres. It is not crucial to produce approximations with solid interiors, therefore we require that only the surface of the object be completely covered. It is advantageous to have the interior of the object filled with spheres as this reduces the chances of tunnelling but it is not essential. Some algorithms, particularly those based on the medial axis, usually fill the interior of the object quite well.

The surface of the object is represented by an arbitrarily large set of sample points. Thus to segment the object into regions we simple choose the sub-set of points that represents the surface within that area. For surface points covered by a single sphere the situation is simple. Each point must be covered by at least one of the children spheres. When a pair of spheres overlap, the intersection of the spheres is divided in two using a plane. The points inside the overlap are assigned to one of the spheres based on which side of the dividing plane they lie. An initial division plane can be computed as the plane of intersection of the two spheres. The following equation represents this plane for two spheres S_1 and S_2 located at (x_1, y_1, z_1) and (x_2, y_2, z_2) with radii r_1 and r_2 respectively :

$$\begin{aligned} 2x(x_2 - x_1) \;+\; & 2y(y_2 - y_1) \;+\; 2z(z_2 - z_1) \;+ \\ & [r_2^2 - (x_2^2 + y_2^2 + z_2^2)] \;-\; [r_1^2 - (x_1^2 + y_1^2 + z_1^2)] = 0 \end{aligned}$$

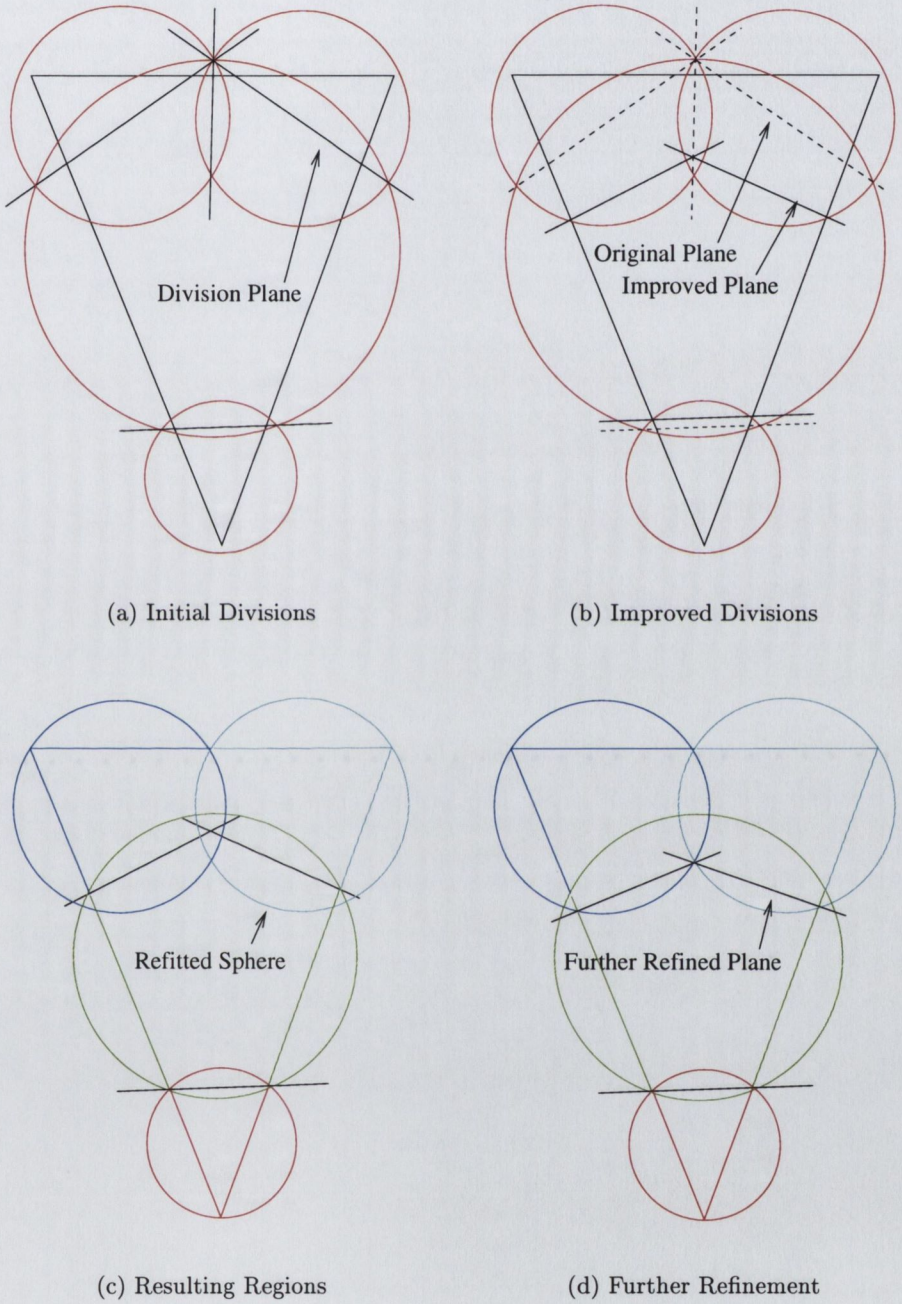
Figure 6.1 shows how a triangular object is divided into four regions. The dividing planes, shown in Figure 6.1(a), are constructed so that the plane passes through the points of intersection of the circles (spheres in 3D). As the sets of spheres can be generated by one of a number of different algorithms, they are essentially of arbitrary configuration. Each set of spheres can be made up of spheres of varying size, both small and big. As illustrated, the region associated with the large central sphere is much bigger than the regions associated with the other spheres. Thus, the children of this sphere will potentially give a looser fit than the other sets of children. As the two top spheres are quite loose fitting it is possible to move their dividing planes closer to the center of the larger sphere as shown in Figure 6.1(b). It is very desirable that the regions divide the object as evenly as possible without affecting the fit of the spheres. The dividing plane between a pair of spheres is moved so as to get more of the shared points to be assigned to the smaller spheres.

Once the object has been divided up into a number of regions, a new set of spheres can be created. These new spheres are now only required to cover the parts of the surface that are not covered by any other spheres. A minimum error sphere can be fitted around each set of points to produce a tighter approximation, as shown in Figures 6.1(c). This process can then be repeated to further reduce the area to be covered by the large sphere, see Figure 6.1(d). Algorithm 8 presents the generic sphere-tree construction algorithm in its recursive form.

6.2 Sphere Set Optimisation

The generic sphere-tree construction algorithm, described above, sub-divided the object in a recursive fashion, approximating each area of the object with a set of spheres. The algorithm can utilise any of the sphere generation algorithms presented in Chapters 3 - 5 for this approximation. Each of the algorithms aimed to create a set of spheres to approximate the required region of the object as accurately as possible. As there is potentially a large number of combinations to be considered, the algorithms use heuristics to reduce the computational costs. While these heuristics generally do lead to good solutions they can often lead to sphere sets that still contain some residual error. The tightness of fit varies between algorithms and results from the way in which the spheres are generated.

For example, Hubbard's successive merging algorithm (Section 4.3) uses a greedy algorithm to choose which pair of spheres to combine. This algorithm does not consider the consequences of its operation on the final result - it is only concerned with choosing the best pair of spheres to combine for any given iteration. The expand method grows all the spheres to a given stand-off distance and then selects a minimal set of spheres i.e. one in which there are no redundant spheres. As the algorithm progresses, new spheres are selected so as to try to cover the most previously uncovered surface. However, this does not guarantee to produce the globally optimal set of spheres. Also, as many of the



(a) Initial Divisions

(b) Improved Divisions

(c) Resulting Regions

(d) Further Refinement

Figure 6.1: Dividing the object into distinct regions using dividing planes.

Algorithm 8 Generic Sphere Tree Construction.

Input : Set of points representing object, P.
Tree depth, d. Branching factor, b.
Output : Sphere-tree covering the object, T.

```
ConstructTree(T, P, d, b)
  T_root ← BoundingSphere(P)                    {minimum bounding sphere}
  MakeChildren(T_root, P, d, b, 1)              {make sub-trees}
```

Input : Sphere-tree node for which to create children, N.
Region of the object to cover, P. Tree depth, d.
Branching Factor, b. Current *level* of recursion.
Output : Sub-tree which has node N as its root

```
MakeChildren(N, P, d, b, level)
  S ← Set of spheres with at most b spheres, covering the surface
      defined by the set points P (using the chosen sphere set
      generation algorithm)

  OptimiseSpheres(S, P)                         {optimise if you like}

  for all s ∈ S do
    H ← {}
    for all t ∈ S, t ≠ s do
      if t overlaps s then
        p ← plane between s & t                 {dividing planes}
        H ← H ∪ {p}
      end if
    end for

    Q ← {}                                      {sub set of points}
    for all p ∈ P do
      if s contains p and p is in the region defined by H then
        Q ← Q ∪ {p}
      end if
    end for

    s' ← minimum error sphere around Q
    N_children ← N_children ∪ {s'}              {add sphere to tree}

    if level < d then
      makeChildren(s, Q, d, b, level+1)         {recursively make sub-tree}
    end if
  end for
```

algorithms are based on the original medial axis method, they rely on the set of spheres constructed from the medial axis approximation. Thus their final accuracy is dictated by the accuracy of the medial axis and the number of vertices within it. The adaptive medial axis algorithm allows larger sets of spheres to be constructed as desired but some residual error can still be present.

As a final stage of improving the tightness of fit, a general purpose optimisation algorithm can be used. Such an algorithm would be free to manipulate every sphere in any way it can so as to achieve a better fitting set of spheres. Figure 6.2(a) shows a pair of spheres that approximate a section of an object. Sphere A obviously fits the surface tighter than B. As the larger sphere has a worse fit than the smaller one it is desirable to make sphere A slightly looser so that the accuracy of sphere B can be improved, as illustrated in Figure 6.2(b) & Figure 6.2(c).

It is desirable that the maximum error be minimised within any set of spheres, but if we cannot decrease this error we would like to decrease any other errors, thus the optimisation function should consider both the *worst* error and the *RMS* error. This is achieved with the following metric :

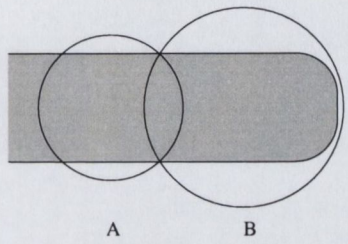
$$E = MAX_{i=0}^{N} Error(S_i) \quad + \quad weight * \sqrt{\sum_{i=0}^{N} \frac{Error(S_i)}{N}} \tag{6.1}$$

where :

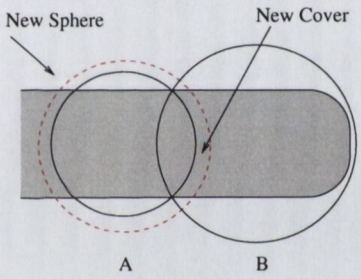
E	is the error associated with the set of spheres,
S_i	is sphere i from the set of spheres S,
$Error(x)$	is a measure of how well x fits the surface,
N	is the number of spheres in the set,
$weight$	is the relative weighting between the maximum error and the RMS error.

There are many different functions for evaluating the fit of a sphere. Section 4.3 details a method that measures the maximum distance from a set of points, on the surface of the object, to a sphere. This is of course an approximation, as the real error should be the maximum distance from the sphere to the surface. Section 5.1 uses a metric that accurately computes the fit of a sphere to the surface of a convex object. It can also serve as an over-approximation of the error for non-convex surfaces. When computing his results, Hubbard uses a more computationally intensive method that finds the point on the object that is closest to each of a number of sample points. These are points that are distributed across the section of the sphere that lies outside the object [49]. Hubbard generates his sample points using a dodecahedron, which produces 12 sample points. A more general sampling scheme could use an isohedron[1], which will have $\frac{n}{2} + 2$ vertices, where n is the

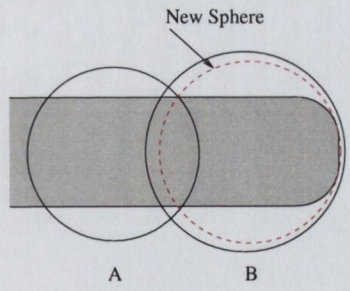
[1] An Isohedron is a convex polytope in which each face is an equilateral triangle of unit area.



(a) Sub-Optimal



(b) Adjusted Sphere



(c) Improved Neighbour

Figure 6.2: Adjusting one of the spheres covering the object can allow the other spheres to cover a different area of the object and thus decrease the worst error of the approximation.

number of faces. The isohedron can be generated by recursively sub-dividing the faces of an equilateral octahedron (or any shape with equilateral faces). This allows larger sets of evenly spaced points to be generated on the surface of the sphere.

If the optimisation algorithm were allowed to manipulate all the spheres simultaneously, there would be a large number of variables to consider (4 variables per sphere). This would make it difficult for the algorithm to find a good solution so a more structured approach is necessary. The approach used is an iterative one. Each iteration of the algorithm takes each sphere in turn and tries to adjust its position and radius so as to improve the overall error in the approximation. The spheres can be considered in round-robin order or in order of increasing/decreasing error. We consider the spheres in order of decreasing error as we are most interested in improving the worst spheres first. Algorithm 9 presents the top level algorithm, which manipulates the spheres in order of decreasing error. The objective function used to update and evaluate the sphere set is given as Algorithm 10.

Algorithm 9 Sphere Optimisation.

Input : Set of spheres, S. Points to be covered, P.
Output : Updated set of spheres, S.

```
OPTIMISESPHERES(S, P)
  while the sphere set is still improving do
    done ← {}
    for n=1 to ||S|| do
      worstErr ← 0
      worstS ← NIL                          {sphere with highest error}
      for all s ∈ S, s ∉ done do
        error ← error associated with sphere s
        if error > worstErr then
          worstS ← s
          worstErr ← error
        end if
      end for

      done ← done ∪ {worstS}                {flag as already done}

      S ← optimise spheres in set S by manipulating
          worstS to minimise OptFunc        {optimise spheres}
    end for
  end while
```

Algorithm 10 Optimisation Objective Function.

Input : Set of spheres, S. Points to be covered, P.
Sphere to manipulate, *manSph*.
New sphere, *repSph*.
Weighting factor between MAX and RMS errors, *weight*
Output : returns error metric for new configuration.

OPTFUNC*(S, manSph, repSph, P, weight)*

for all $s \in S$ **do** {*points in spheres*}
  $Points[s] \leftarrow$ set of points, from P, inside s
end for

for all $p \in Points[manSph]$, p not inside *repSph* **do** {*ensure coverage*}
  $closestS \leftarrow$ closest sphere to p
  $Points[closestS] \leftarrow Points[closestS] + \{p\}$
end for

for all $p \in P$, p inside *repSph* **do** {*free covered points*}
  for all $s \in S$ **do**
    $Points[s] \leftarrow Points[s] - \{p\}$
  end for
end for

$T \leftarrow S - \{manSph\}$

$worstErr \leftarrow Error(repSph)$
$sumSqErr \leftarrow worstErr^2$ {*evaluate metric*}
for all $s \in T$ **do**
  $newS \leftarrow$ FITSPHERE(Points[s])
  $e \leftarrow$ ERROR(newS)
  $sumSqErr \leftarrow sumSqErr + e^2$ {*accumulate term*}
  if $e > worstErr$ **then**
    $worstErr \leftarrow e$
  end if
end for

return $worstErr * weight + \sqrt{\frac{sumSqErr}{\|S\|}}$

6.3 Balancing Work vs. Error

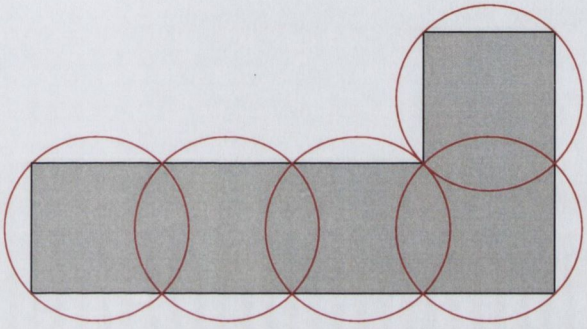
The generic sphere-tree construction algorithm, presented in Section 6.1, is capable of using a number of different algorithms to construct the sphere sets for the sphere-tree. The algorithm for constructing these hierarchies uses the chosen sphere generation algorithm to approximate a region of the object with a specified number of spheres. This number, N_c, is the maximum branching factor of the hierarchy. Thus each non-terminal node in the tree will have up to N_c child nodes. As the value of N_c increases, so does the number of spheres that will need to be tested if their parent is involved in an overlap.

The value of N_c is given as a parameter to the sphere-tree construction algorithm, and is entirely a matter of choice. None of the algorithms presented try to determine a suitable value for N_c. The number of spheres required to approximate a region of the object will largely depend its geometry. Take for example the $2D$ shape featured in Figure 6.3, which has been approximated with 5 and 8 circles. While using more (i.e. 8) circles to approximate the object does reduce the error in some parts, it actually increases the worst error and thus makes the approximation worse. There is also much greater overlap between the spheres which means there is a higher likelihood that the sphere-tree traversal algorithm will have to traverse multiple sub-trees. Thus, using a larger number of spheres does not necessarily give a better approximation and it certainly does increase the work load for the collision detection algorithm.

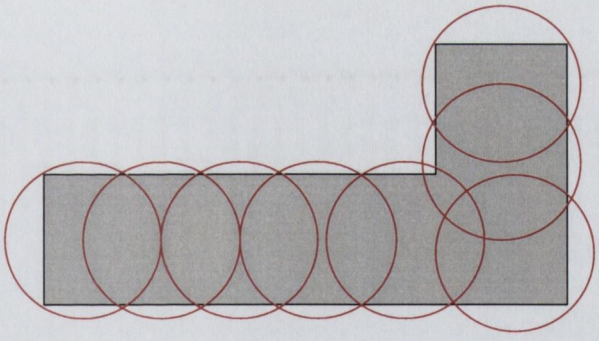
It is questionable whether we wish to always use the maximum allowable number of spheres to approximate the object. If there is only a small increase in the tightness of the set of spheres then it is not worth increasing the computational demands on the collision handling system as there will only be a small gain in accuracy (if any at all). Therefore, having generated the maximum number of spheres it may often be desirable to further reduce the number of spheres used if it only introduces a small amount of error.

It is therefore necessary to be able to determine if the error introduced by the removal of some of the spheres is acceptable. It is easy to compute the reduction in computation required for the collision handling system. For example, if there is one less sphere, then the computation required for that set of siblings will have decreased by $\frac{1}{N_o}$, where N_o is the original number of spheres. This generalises to $\frac{m}{N_o}$ when m spheres are removed. The allowable increase in error can either be a function of the work saved ($\frac{m}{A*N_o}$) or a constant (say 5%). If the allowable increase in error is a function of the amount of work saved, it is desirable that it become more difficult to remove each successive sphere so as to prevent the algorithm throwing away too many spheres. Thus, a penalty is associated with each sphere removed. The following equation captures this generally as :

$$Allow(N) = Error(N_o) * (1 + \frac{N_o - N}{A * N_o} * v^{N_o - N - 1} + B) \tag{6.2}$$



(a) 5 spheres



(b) 8 spheres

Figure 6.3: Illustration of a shape that does not require the full number of spheres to achieve a good approximation.

where :

Allow(N)	is the maximum allowable error for N spheres,
N_o	is the original number of spheres,
Error(N_o)	is the error present with N_o spheres,
v & B	are scalars in the range $\{0..1\}$,
A	is a positive scalar.

The value of v dictates how difficult it is to remove successive spheres from the set. If we choose a value such that $0 < v < 1$ then it will become increasingly more difficult to remove each successive sphere. For example using a value of $v = 0.95$ and $N_o = 8$ then one sphere can be removed provided the resulting error is less than $1 + \frac{1}{8} * 0.95 = 1.11875$ times the error when using all 8 spheres. When removing a second sphere this number becomes $1 + \frac{2}{8} * 0.95^2 = 1.225625$, which is less than twice the increase in the allowable error. The lower the value of v the harder it becomes to remove each successive sphere. The choice of these parameters is up to the developers who are using the sphere-trees[2]. Algorithm 11 shows an algorithm that can be used, by the generic sphere-tree construction algorithm, to discard unnecessary spheres from the set prior to its inclusion in the sphere-tree.

6.4 Conclusions

In previous chapters, a number of improvements to the existing methods of creating sphere-trees were discussed. These improvements led to a number of new algorithms for approximating an object (or a section of an object) using a limited number of spheres. This chapter looked at a generic way of using these algorithms to construct sphere-trees. Attention was paid to the amount of overlap that could exist in these sets of spheres and the object was sub-divided in such a way as to minimise the redundancy introduced into the sphere-tree and to make the divided regions similarly sized. As many of the algorithms still have potential for generating sphere sets that contain some residual error, a generic sphere set optimisation algorithm was presented. This manipulated the sets of spheres with the aim of further improving the fit, and can be applied to each sets of spheres to minimise their error prior to incorporating them into the sphere-tree. Finally, a more complicated optimiser was presented. This algorithm not only aims to minimise the amount of error in a set of spheres but also tries to eliminate unnecessary spheres so as to reduce the computational demands of the narrow phase traversal.

[2]When constructing sphere-trees for this thesis, the following values were generally used : $A = 5, v = 0.75, B = 0.0$. This allows an increase of about 2.5% when removing one sphere from a set of 8 and 4.375% for the second sphere. Another commonly used set of parameters is $A = \inf, v = 1, B = 0.05$, which allows spheres to be removed as long there is less than a 5% increase in error, i.e. 105% of the original error.

Algorithm 11 Remove spheres that contribute little to the approximation.

Input : Set of spheres, S. Points to be covered, P.
Parameters for Equation 6.2, A, B & v.
Output : Updated set of spheres, s.

BALANCESPHERES$(S,\ P,\ A,\ B,\ v)$
  $S \leftarrow$ OPTIMISESPHERES(S, P) {*initial optimise*}
  $StartErr$ = maximum error associated with spheres in S
  $N_o = \|S\|$ {*initial size*}

  for $N = (N_o - 1)$ to 1 **do**
    $allowedErr \leftarrow StartErr * (1 + \frac{N_o - N}{A * N_o} * v^{N_o - N - 1} + B)$

    $T \leftarrow S$ reduced, with burst algorithm, to have N spheres
    $Err \leftarrow$ maximum error associated with spheres in T

    if $Err \geq allowedErr$ **then** {*try to get error below allowedErr*}
      $T \leftarrow$ OPTIMISESPHERES(T, P)
      $Err \leftarrow$ maximum error associated with spheres in T
    end if

    if $Err \geq allowedErr$ **then**
      break {*done removing spheres*}
    else
      $S \leftarrow T$ {*update sphere set*}
    end if
  end for

  if $\|S\| < N_o$ **then**
    $S \leftarrow$ OPTIMISESPHERES(S, P) {*final optimise*}
  end if

Chapter 7

Evaluation

Chapters 3 and 4 presented two existing algorithms for the construction of sphere-trees. These methods were examined and a number of improvements were proposed. These developments aimed to improve the tightness and consistency of the approximations generated. Chapter 6 presented a generic sphere-tree construction algorithm that used these methods to approximate areas of the object when constructing the hierarchies. Also presented was a generic optimisation algorithm for further improving the tightness of the spheres created.

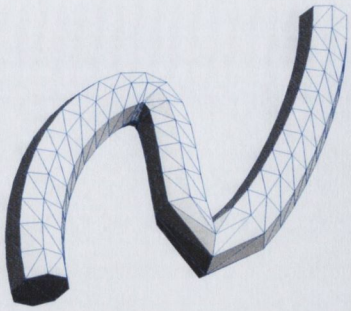
Sphere-tree construction algorithms can be thought of as falling into two broad categories. The first category consists of algorithms, such as the octree based algorithms, that are primarily concerned with the spatial localisation properties of the sphere-trees, i.e. their ability to narrow in on the areas of contact between two objects. The second class are the object approximation algorithms, which aim to approximate the object geometry closely so as to reduce the number of false positives, and hence the amount of wasted computation in the sphere-tree traversal. For interruptible collision detection, close approximation of geometry is particularly important as the spheres are used to approximate the points of contact and to compute the response.

This chapter compares the various algorithms for both object approximation and sphere-tree construction. The algorithms are compared using a number of simple geometric shapes including a cube, an ellipsoid, a cylinder, a torus, a cone, an "S" shaped object created using NURBS surfaces and a block with square cross sections. A number of commonly used complex models have also been used, including the Bunny[1], the Cow and the Dragon[2]. These meshes have been simplified, to contain about 1500 triangles, using Garland's *QSlim* software and can be seen in Figure 7.1.

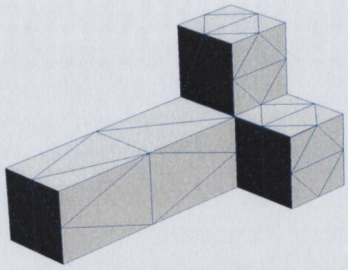
The initial analysis is concerned with the geometric properties of the sphere-trees resulting from the various algorithms. Later analysis considers the use of the hierarchies in an interruptible collision handling system.

[1]Data from http://graphics.stanford.edu/data/3Dscanrep/

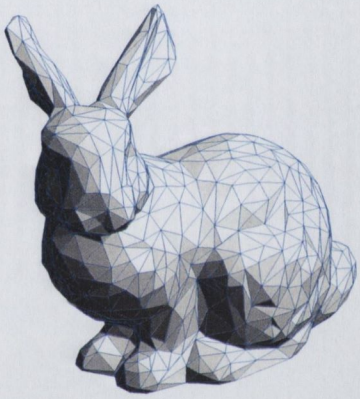
[2]Data from http://graphics.cs.uiuc.edu/~garland/research/quadrics.html



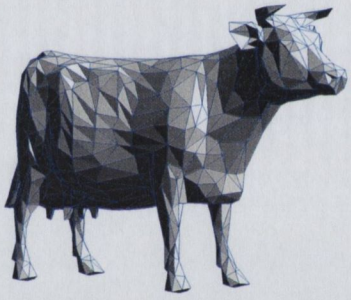
(a) S-shape



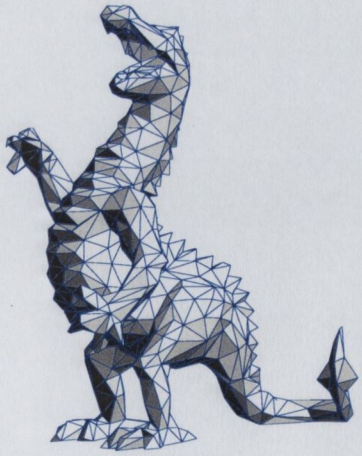
(b) A Block



(c) Bunny



(d) Cow



(e) Dragon

Figure 7.1: Some of the models used for testing the algorithms.

7.1 Geometric Approximations

Chapter 5 considered a number of improvements that can be made to the medial axis method for sphere-tree construction, originally presented in Chapter 4. A sub-problem within the sphere-tree construction algorithm is to approximate a region of an object with a number of spheres. The algorithm uses an approximation of the medial axis as a guide to where spheres should be placed. Having constructed the medial axis, a large number of spheres were constructed so they were centered on the medial axis and touched the surface at a number of points. An iterative merging algorithm was utilised to reduce this set of medial spheres down to the number required for the sphere-tree.

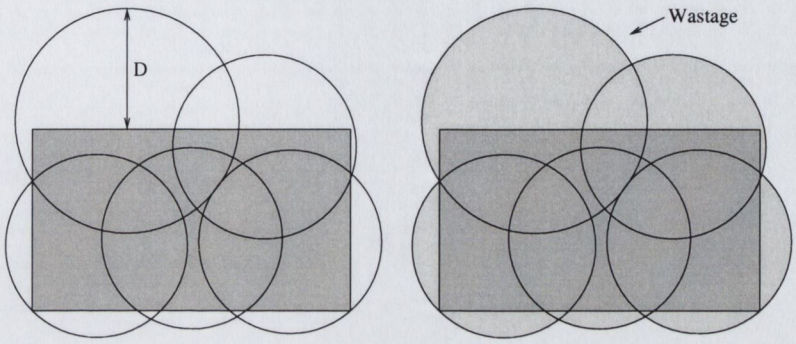
This section first considers the adaptive medial axis construction algorithm, presented in Section 5.1, comparing it to a regular sampling algorithm based on Hubbard's. The second stage of the algorithm, the sphere reduction/selection stage, is next considered and finally the sphere-trees produced by the algorithms are compared in terms of geometric quality of fit.

7.1.1 Strategy

There are a number of factors that must be considered when approximating an object with spheres. As stated in Section 2.3, the spheres should approximate the object's surface to a high degree of accuracy and should cover the entire object. The tightness of fit can be measured in terms of the distance from the surface of the spheres to the actual surface of the object or in terms of the volume within the spheres that is not occupied by the object, see Figure 7.2. For the purposes of approximating objects closely, the maximum distance from the surfaces of the spheres to the object is the most important factor as this represents the largest gap that can be present in the approximated collision. The amount of the object not covered by the approximation can be either the volume of the object that is not contained within a sphere or the amount of surface area that is not covered by spheres, see Figure 7.3. It is critical that the surface of the object is completely covered so as not to miss any collisions. Although filling the interior of the object is not critical it can be beneficial as it reduces the chances of missing collisions between time-steps, as discussed in Section 2.1.

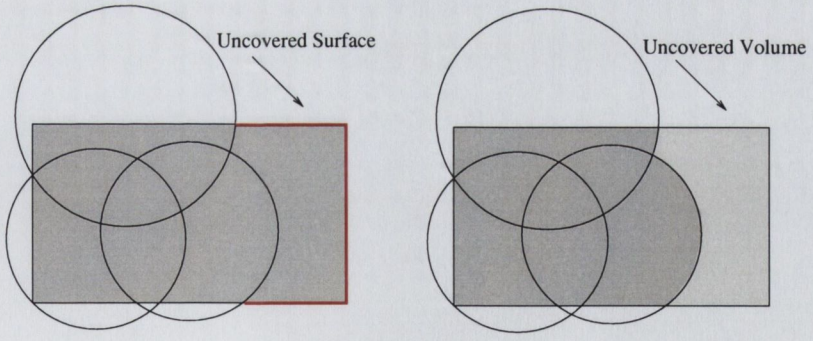
Measuring The Wasted and Uncovered Volumes

Measuring the volume of the spheres that is outside the object and the volume of the object not covered by spheres are both essentially integration problems. These regions can be expressed as the boolean difference between the sphere set and the model. The region of wastage can be expressed as $S-O$ and the uncovered region as $O-S$, where O is the object being approximated and S is the union of the spheres. Monte Carlo integration techniques allow us to determine these volumes when they cannot be determined analytically. The



(a) Maximum Distance (b) Wasted Sphere Volume

Figure 7.2: The error in an approximation can be measured as the maximum distance from the surface of the spheres to the object or as the volume of the wasted portions of the spheres.



(a) Surface Area (b) Uncovered Volume

Figure 7.3: The amount of the object not covered by spheres can be measured using either volume or surface area

Hit or Miss integration algorithm expresses the solution in terms of a number of random points sampled from a bounding region, the domain of integration, see Figure 7.4(a). The wasted volume, V_w and the uncovered volume V_u can be determined as follows :

$$V_w = \frac{N_w}{N_t} * V_b \tag{7.1}$$

$$V_u = \frac{N_u}{N_t} * V_b \tag{7.2}$$

where :

V_w	is the volume of the sphere wastage,
V_u	is the volume of the uncovered regions of the object,
N_w	is the number of samples that fall in a sphere but not in the object,
N_u	is the number of samples that fall in the object but not in a sphere,
N_t	is the total number of samples tested,
V_b	is the volume of the boundary region.

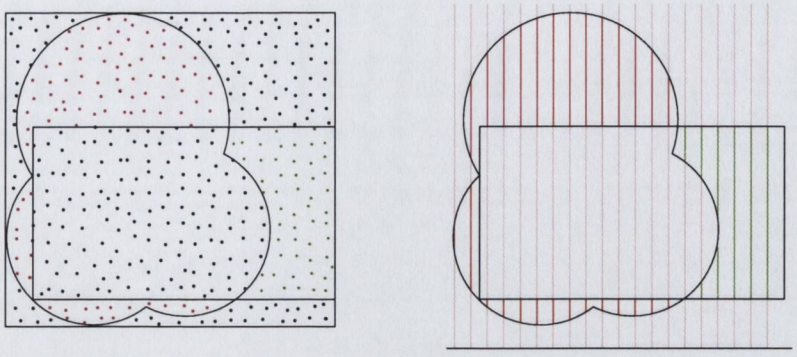
An alternative to the hit or miss method is the *Mean Sample* method, illustrated in Figure 7.4(b). In this method we distribute samples across two dimensions of the integration domain, say the XY plane, and project rays along the third dimension. The height associated with each sample is the length of the ray that intersects the region we are interested in. By averaging a large number of these samples we compute the height of the cuboid that represents a volume equal to that which we are trying to determine. Thus the volume can be calculated as :

$$V = L_a * A_d \tag{7.3}$$

where :

V	is the volume to be integrated,
L_a	is the length of the ray associated with the mean sample,
A_d	is the area of the domain of integration i.e. the XY face of the bounding box.

As with any Monte Carlo integration, both these methods are subject to a statistical error. The mean sample method is more statistically sound than hit or miss as it converges to the correct answer quicker and hence requires fewer samples. In order to improve the convergence of the results, we use the *Miser* integration algorithm which first distributes samples randomly and then focuses its effort in regions of high variance [85].



(a) Hit Or Miss (b) Mean Sample

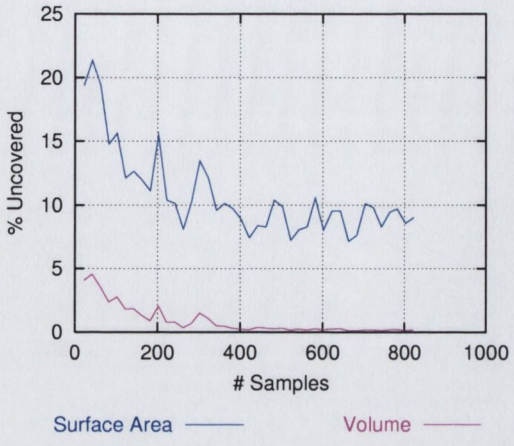
Figure 7.4: Integrating the wasted and uncovered volumes using Monte Carlo techniques.

7.1.2 Medial Axis Construction

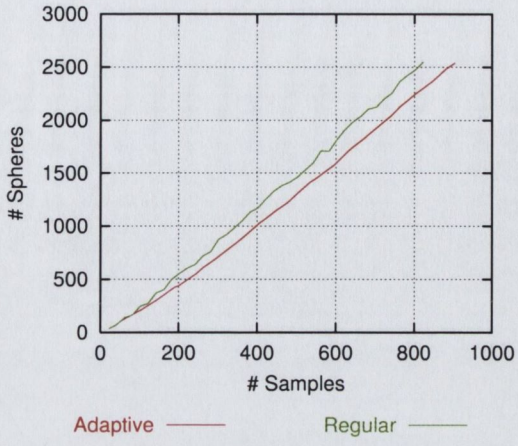
Section 5.1 detailed a number of problems associated with approximating the medial axis. The algorithm constructs the Voronoi diagram for a set of points distributed across the surface of the object. The internal vertices of the Voronoi diagram were then used to approximate the medial axis, upon which spheres were to be centered. The question of how many samples should be used, and where they should be placed, is an important one. The problem of choosing surface samples sometimes leads to the medial axis being poorly approximated. Hubbard used the notion of gap crossing cells to address some of the problems that can be experienced. However, this scheme does not ensure that the spheres generated from the medial axis will fit the surface to a desired level or will even cover the entire object.

Thus, an adaptive sampling scheme was developed. This allows extra samples to be added to the surface so as to generate spheres whose error is bounded to a desired value. In order to ensure that the entire object is covered, the algorithm adds extra spheres using Voronoi vertices that do not lie on the medial axis. The addition of these extra spheres, which ensure coverage of the surface, is very necessary in order to generate a faithful approximation. As discussed in Section 5.2, the adaptive sampling algorithm allows for this quite nicely. New spheres can be chosen so as to minimise the error introduced into the approximation or so that they will be quickly replaced by the adaptive algorithm. In the experiments conducted for this thesis the latter performed marginally better and therefore has been chosen for the purposes of evaluation.

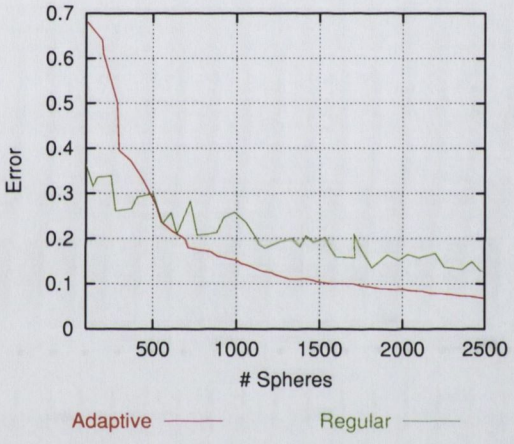
Figures 7.5 - 7.8 compare the adaptive sampling scheme with Hubbard's relaxation based algorithm. The first graph (a) shows the amount of the object that is covered by the set of medial spheres, results are shown in terms of surface area and object volume. The graphs only show results for Hubbard's algorithm as the adaptive algorithm covers the



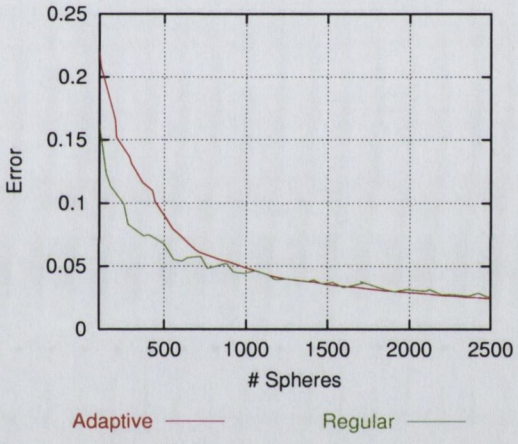
(a) Coverage (Regular Sampling)



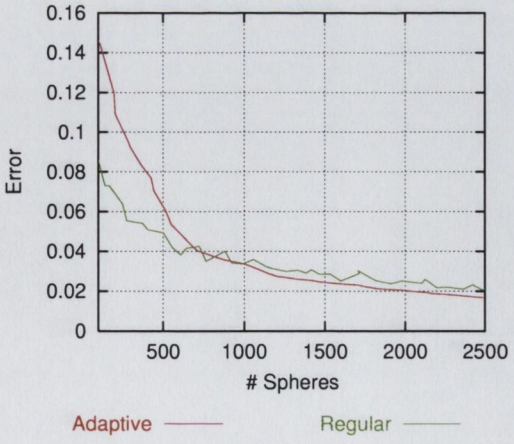
(b) Spheres per Sample



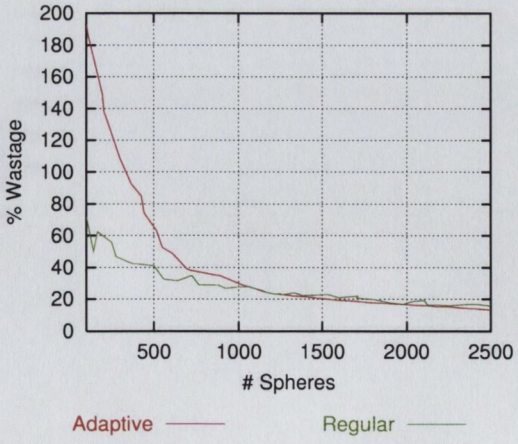
(c) Max Distance



(d) RMS Distance

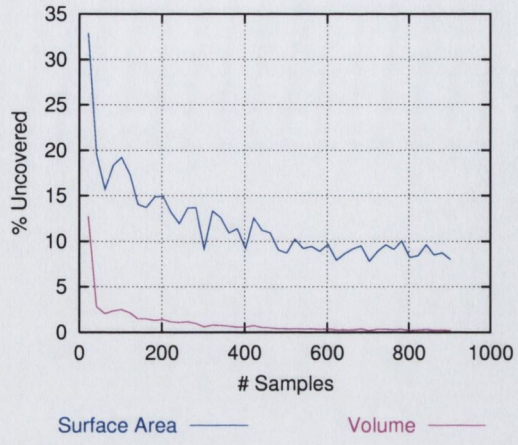


(e) Variance

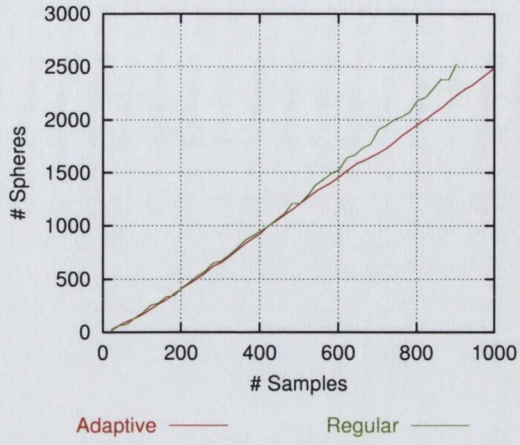


(f) Wasted Volume

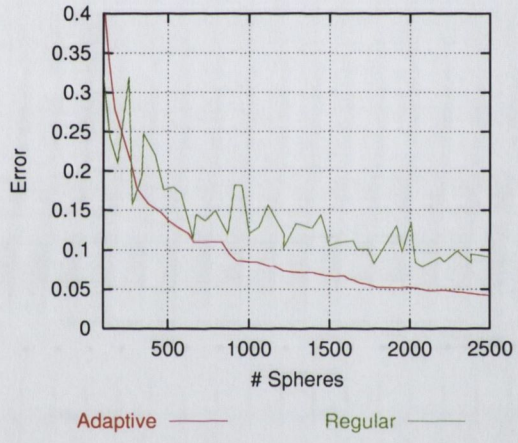
Figure 7.5: Comparison of Regular vs. Adaptive sampling for the construction of the medial set of the Bunny.



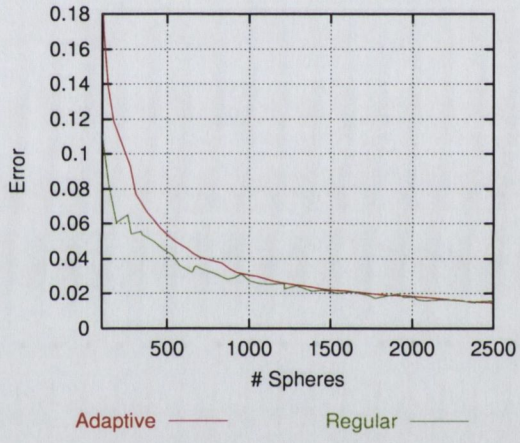
(a) Coverage (Regular Sampling)



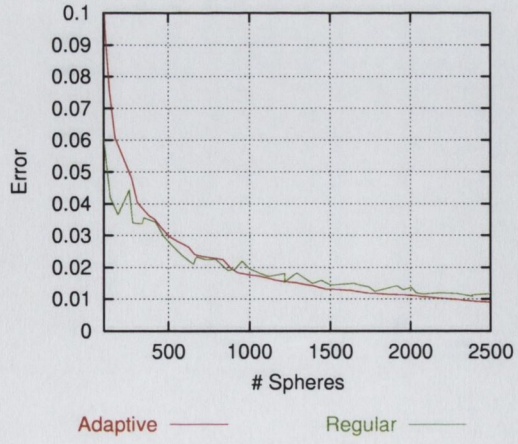
(b) Spheres per Sample



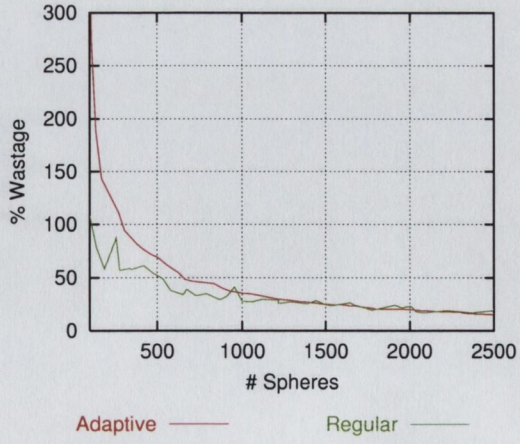
(c) Max Distance



(d) RMS Distance

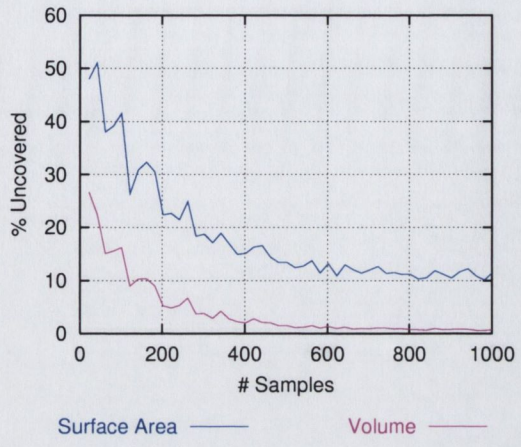


(e) Variance

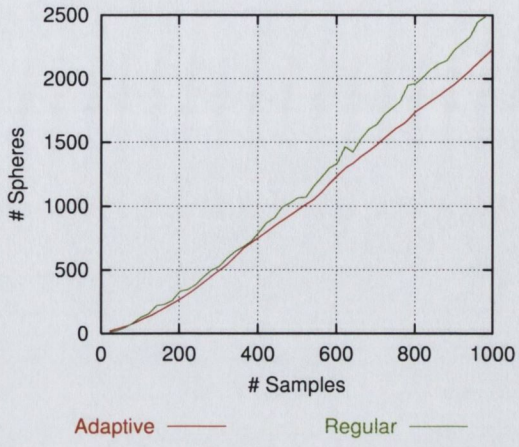


(f) Wasted Volume

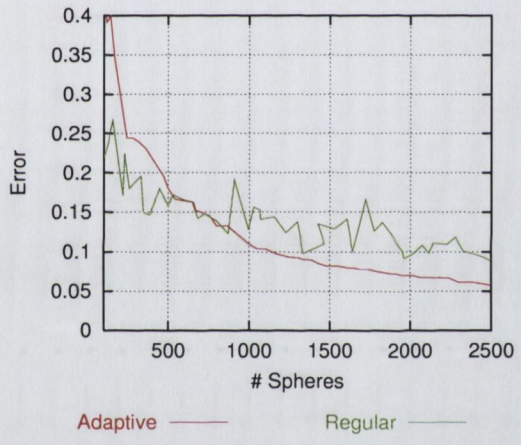
Figure 7.6: Comparison of Regular vs. Adaptive sampling for the construction of the medial set of the Cow.



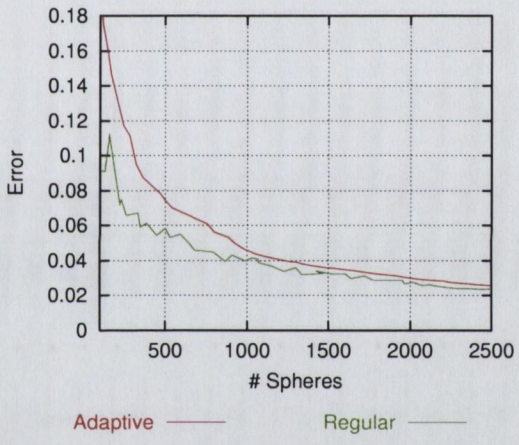
(a) Coverage (Regular Sampling)



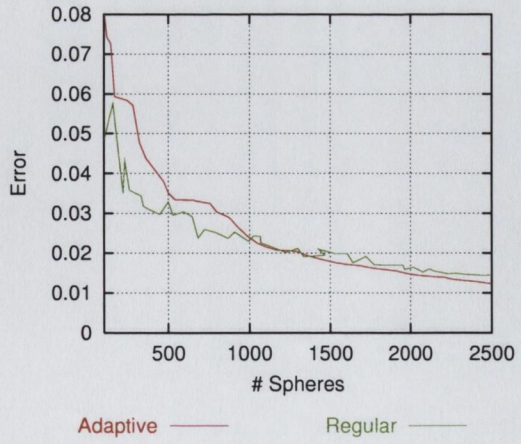
(b) Spheres per Sample



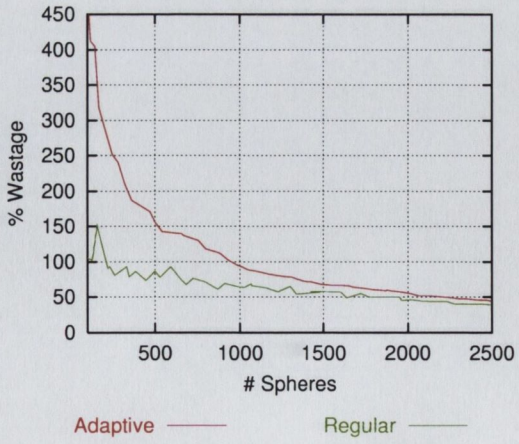
(c) Max Distance



(d) RMS Distance

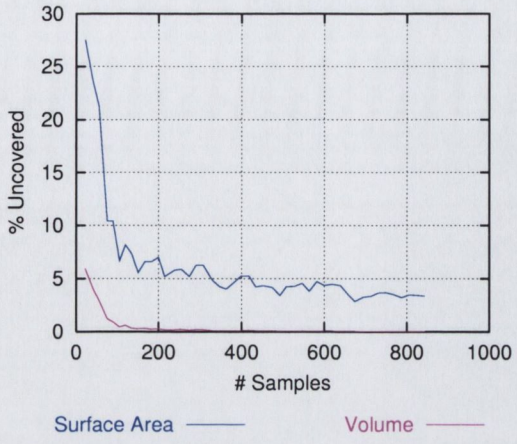


(e) Variance

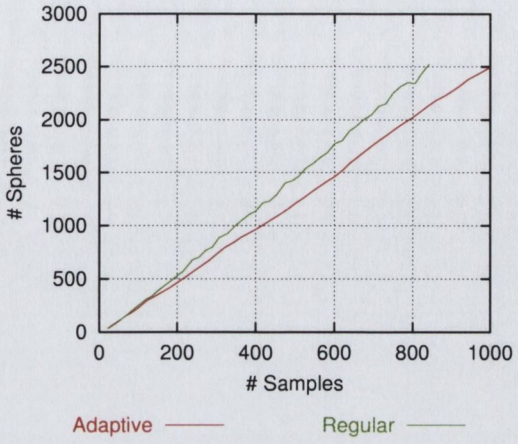


(f) Wasted Volume

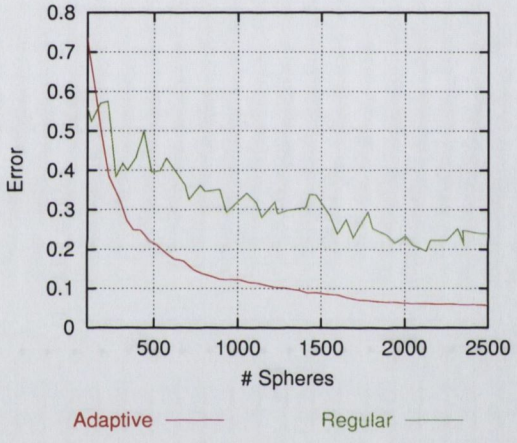
Figure 7.7: Comparison of Regular vs. Adaptive sampling for the construction of the medial set of the Dragon.



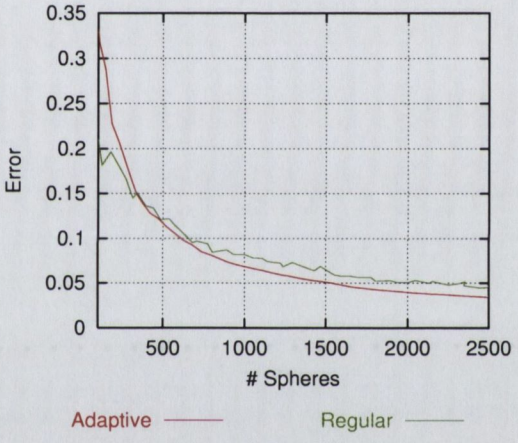
(a) Coverage (Regular Sampling)



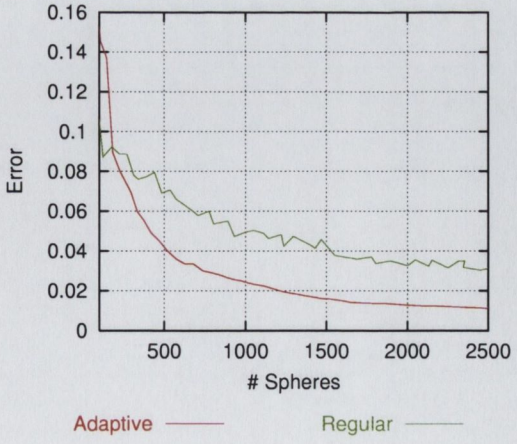
(b) Spheres per Sample



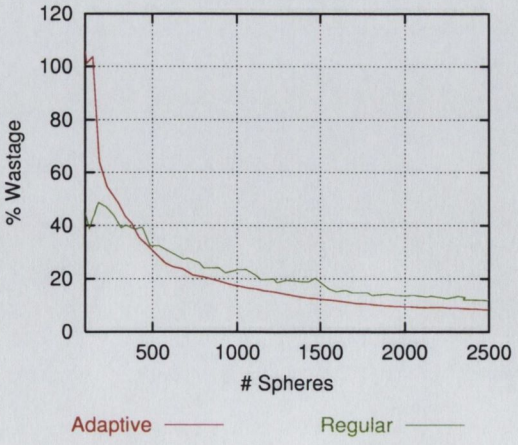
(c) Max Distance



(d) RMS Distance



(e) Variance



(f) Wasted Volume

Figure 7.8: Comparison of Regular vs. Adaptive sampling for the construction of the medial set of the cube.

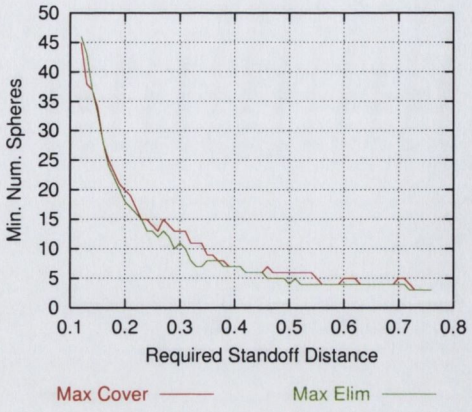
entire object. For the more complex models, the regular sampling scheme often provided rather poor coverage of the object. As the number of samples increases the coverage generally improves, but while the volume of the uncovered regions is reduced to nearly zero there is still a large amount of the surface left uncovered, as seen in Figures 7.5(a) - 7.8(a). This is very undesirable as it can lead to collisions being missed and the objects inter-penetrating. The adaptive algorithm ensures that the surface, represented by an arbitrarily large set of points, is completely covered. Thus gaps can be avoided as long as the set of sample points is sufficiently dense. The second graph (b) of each set shows the number of medial spheres generated for a given number of samples. It is interesting to note that for both algorithms this relationship is pretty much linear.

The third and fourth graphs (c & d) in each set compare the algorithms in terms of the distance from the surface of the spheres to the surface of the object. This basically measures how far the spheres protrude past the surface. The adaptive algorithm generally exhibits a lower worst error than using regular sampling, although for some models it starts out higher, as seen in Figures 7.5(c) & 7.7(c). However, as the adaptive sampling algorithm aims to improve the worst fitting sphere at each iteration the approximation improves quite quickly. The adaptive algorithm also often exhibits a lower variance in the distances from the spheres to the surface, graph (e). This is due to the algorithm choosing to improve the worst fitting sphere at each iteration. Thus spheres with large error are replaced while those with lower values are left alone. The sixth graph (f) shows the volume of the sphere set that does not cover part of the object. This was measured using the mean sample integration technique described in Section 7.1.1. Again, for the more complicated models the adaptive algorithm performs much the same as regular sampling. However for simpler models the adaptive algorithm has much less wastage.

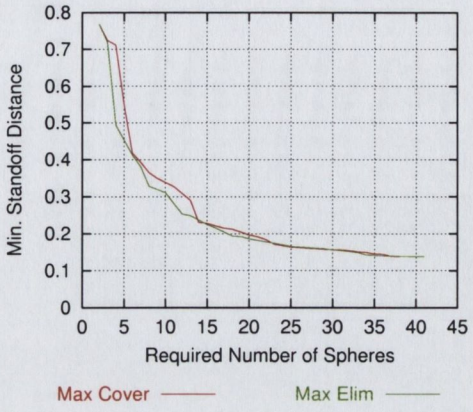
7.1.3 Sphere Selection

In Section 5.3.3, a novel algorithm for sphere set reduction was presented. The algorithm first expanded all the spheres so that they protruded past the surface of the object by a given amount. The algorithm then selected a sub-set of the spheres, which covered the surface. Two heuristics were presented for the selection of the spheres; the first was to select the sphere that covered the most previously uncovered surface area (we'll call this "Max Cover"); the second was to choose the sphere that eliminated the largest number of the remaining spheres (we'll call this "Max Elim").

Figures 7.9 - 7.14 compare these two heuristics for a number of geometric models. The first graph (a) of each set shows the number of spheres chosen to approximate the object with a given stand-off distance. The second (b) shows the minimum stand-off distance, found using an adaptive search algorithm, required to select a certain number of spheres. Table 7.1 shows the results of an "analysis of variance" (ANOVA) comparison to test the hypothesis that the mean number of spheres selected with each algorithm is the same.

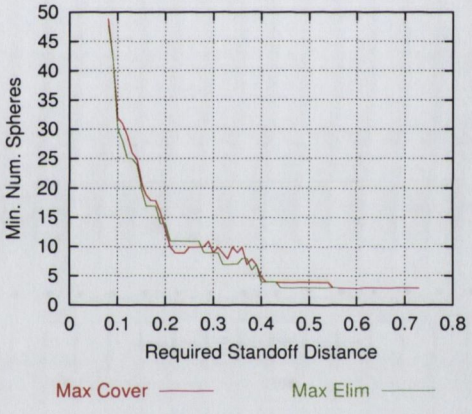


(a) Required Stand-off Distance

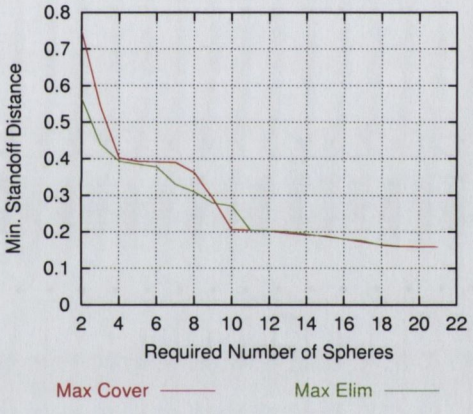


(b) Required Number of Spheres

Figure 7.9: Comparison of sphere selection heuristics for the Bunny.

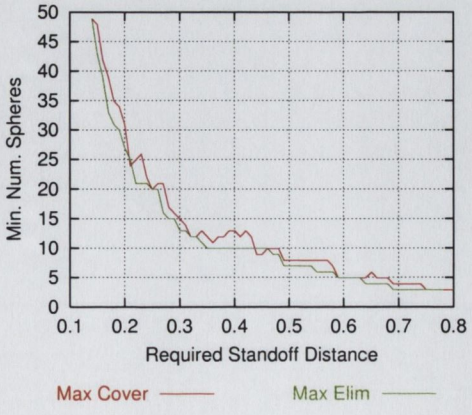


(a) Required Stand-off Distance

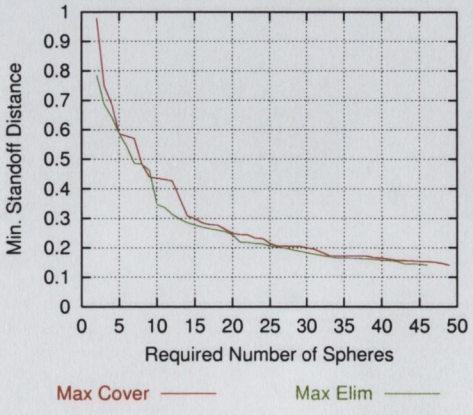


(b) Required Number of Spheres

Figure 7.10: Comparison of sphere selection heuristics for the Cow.

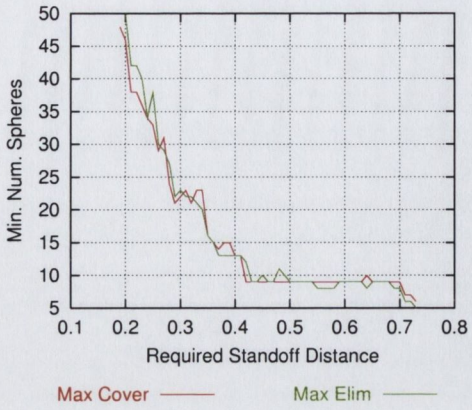


(a) Required Stand-off Distance

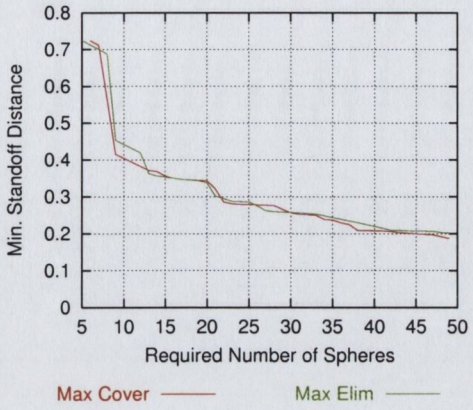


(b) Required Number of Spheres

Figure 7.11: Comparison of sphere selection heuristics for the Dragon.

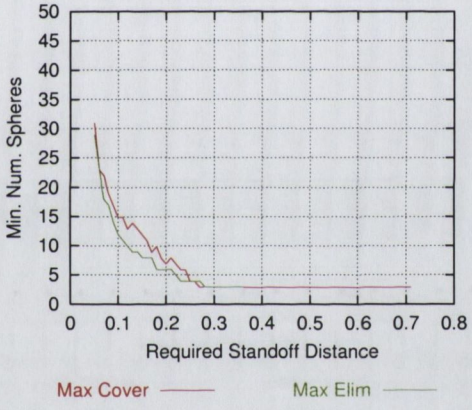


(a) Required Stand-off Distance

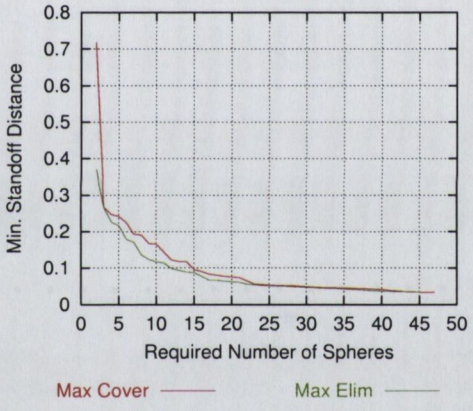


(b) Required Number of Spheres

Figure 7.12: Comparison of sphere selection heuristics for the cube.

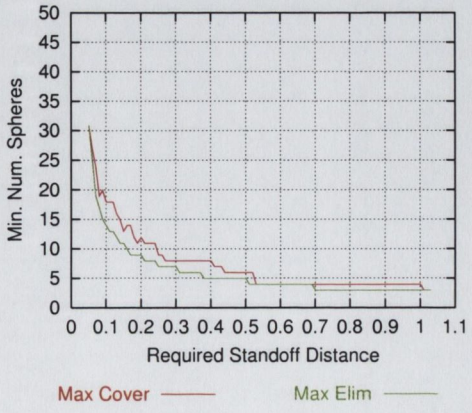


(a) Required Stand-off Distance

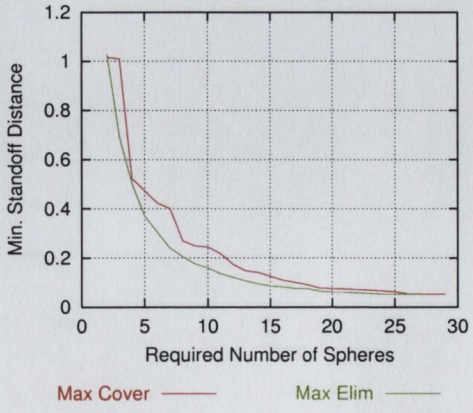


(b) Required Number of Spheres

Figure 7.13: Comparison of sphere selection heuristics for the ellipsoid.



(a) Required Stand-off Distance



(b) Required Number of Spheres

Figure 7.14: Comparison of sphere selection heuristics for the torus.

Model	F	P-value	F crit
Bunny	0.2519	0.6165	3.9151
Cone	0.0120	0.9129	4.0098
Cow	0.6091	0.4367	3.9258
Cube	0.0102	0.9196	3.9306
Cyl	0.6950	0.4075	3.9909
Dragon	0.5972	0.4410	3.9151
Ellipsoid	1.6255	0.2054	3.9412
Block	0.0446	0.8330	3.9333
S-Shape	1.3958	0.2396	3.9151
Torus	3.3848	0.0681	3.9151

Table 7.1: ANOVA comparison of sphere selection heuristics.

From the graphs it can be seen that the Max Elim heuristic often performs marginally better than Max Cover. However in the ANOVA tests the P-Values are quite high for most models. This indicates that, with $\alpha = 0.05$ (95% confidence), the average performance of the two heuristics is very similar. The exception is the torus, where there is a significant difference. As evident from Figure 7.14, the Max Elim heuristic performs significantly better in this situation.

Generally the Max Elim heuristic yields slightly better results than the Max Cover heuristic, requiring fewer spheres to cover the object and a lower stand-off distance to achieve the required number of spheres. The cube, Figure 7.12, is quite a difficult shape to represent using this algorithm, with neither heuristic being noticeably better than the other.

7.1.4 Sphere Reduction

Having constructed a large set of spheres, which approximate the geometry of the object, the sphere-tree construction algorithm needs to reduce the number of spheres down to the number required for the sphere-tree. Hubbard's algorithm successively merged pairs of spheres together to achieve this. When merging two spheres, Hubbard used Ritter's algorithm to form a new bounding sphere. Ritter's algorithm does not provide the minimum volume bounding sphere and can often produce quite loose spheres. Hubbard also measured the error of the spheres in terms of the distance from the surface to the spheres.

A number of improvements were detailed in Section 7.1.4. These were the use of White's minimum volume bounding sphere algorithm [109], the measurement of fit as distance from the sphere to the surface, and the special consideration of merges that reduce the worst error in the approximation. This improved algorithm is presented as "Merge". To further improve the sphere reduction process, the merging strategy was replaced with one, presented as "Burst", that removes a sphere and fills in the hole using the surrounding spheres. A novel algorithm, which expands the medial spheres and eliminates the ones

that are redundant, was also presented as "Expand".

Figures 7.15 - 7.18 compare these reduction algorithms using both the distance based metric and the wasted volume. Each algorithm generates the required number of spheres from a set of 500 medial spheres generated using the adaptive sampling algorithm.

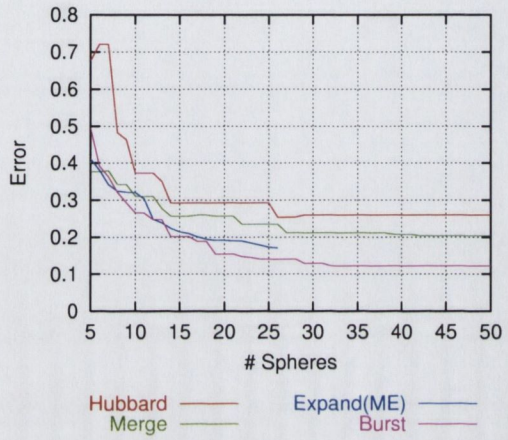
As would be expected, the use of a minimum bounding sphere algorithm and a more accurate sphere fit metric allows the new merge algorithm to provide a reasonable improvement over Hubbard's merging strategy. This algorithm certainly reduces the worst error for all the models. The RMS error and the variance in the approximation is also reduced. The expand algorithm, using the "Max Elim" heuristic, generates sets of spheres with very low error variance. For convex shapes this will be zero, however for non-convex objects some variation is experienced. This is a result of the equation used for computing the sphere's radius, Equation 5.3, which over-estimates the error (and hence under-estimates the sphere's radius) for non-convex bodies. The expand algorithm generally exhibits a lower worst error than either Hubbard's or the merge algorithm. This is due to the way the algorithm tries to distribute the error evenly between all the spheres in the resulting set. The burst algorithm further improves the fit of the reduced sets of spheres. Although the "expand" algorithm tries to distribute the error as evenly as possible it can have difficulty for non-convex regions of the objects as it never tries to reposition the spheres. Also, the expand algorithm has difficulty producing larger sets of spheres. In this situation it will produce the largest set of spheres it can, but these cases are omitted from the graphs as the number of spheres in the set does not match the numbers on the horizontal axis.

7.1.5 Sphere-Tree Construction

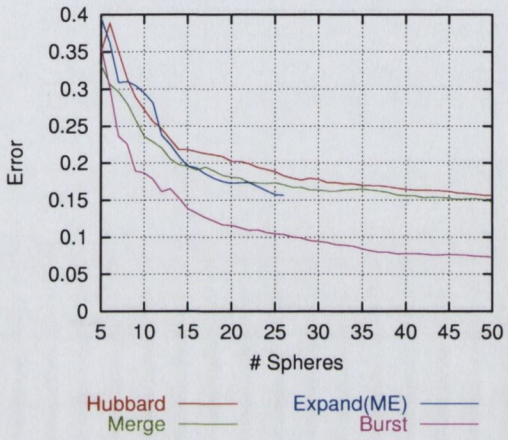
The top level sphere-tree construction algorithm, described in Section 6.1, controls the decomposition of the object into sub-regions. Each of these regions is approximated using a set of spheres. The controlling algorithm aims to divide the object into distinct regions so as to minimise duplication within the hierarchy.

Figures 7.19 - 7.25 and Tables 7.2 - 7.5 compare the geometric fit achieved using the various sphere reduction algorithms. Also shown are results for the "GRID" algorithm and those obtained using the optimiser detailed in Section 6.2. Two additional algorithms are also presented. The first, labelled "Hybrid", is a post-processing algorithm that produces a new sphere-tree from the one generated using the grid algorithm. Each node of the sphere-tree contains the minimum volume bounding sphere, as produced by the grid algorithm, and a minimum error sphere covering the same region. The algorithm labelled "Combined" is simply a sphere reduction algorithm that tries both the "Merge" and "Expand" algorithms and chooses the one with the lowest error. This allows the algorithm to fit tight sets of spheres where the expand algorithm is able to operate more effectively or revert to the merge algorithm, which is more generic in nature.

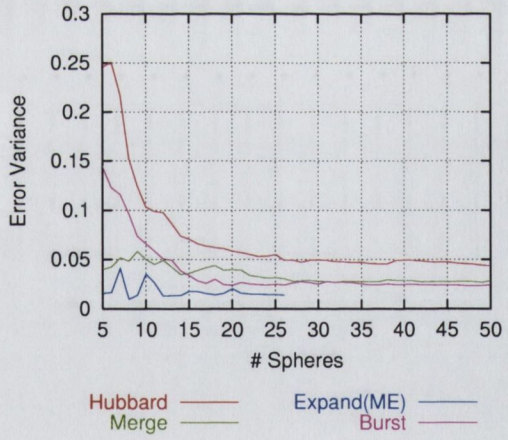
All tests were conducted with a tree branching factor of 8. This number was used as



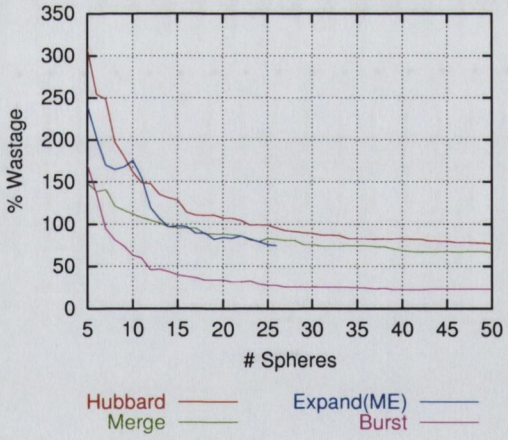
(a) Max Distance



(b) RMS Distance

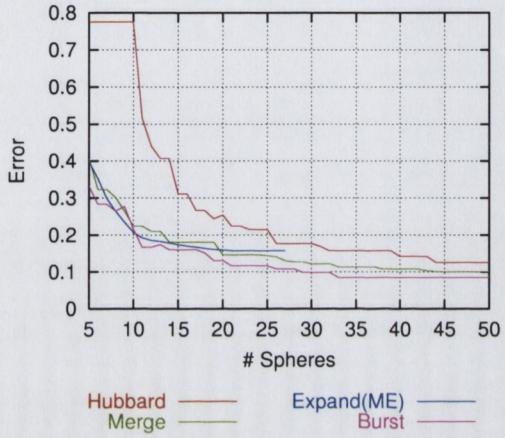


(c) Variance

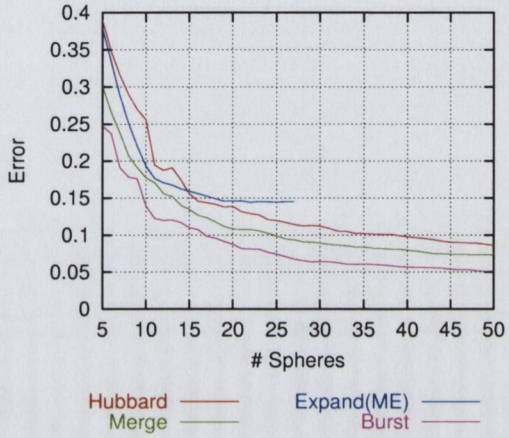


(d) Wasted Volume

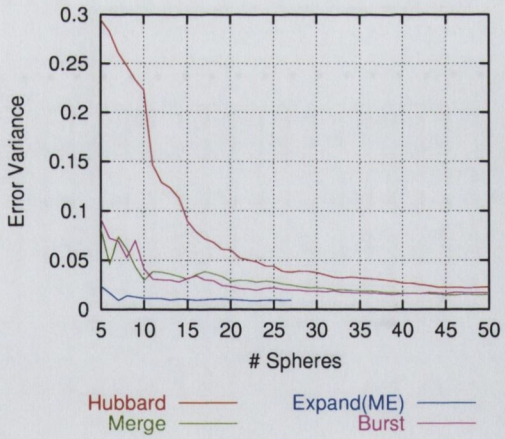
Figure 7.15: Comparison of sphere reduction techniques for the Bunny.



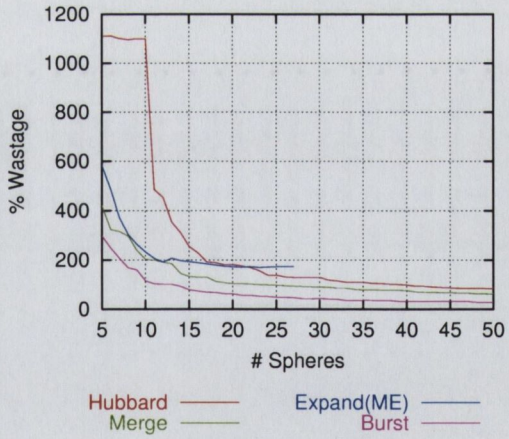
(a) Max Distance



(b) RMS Distance

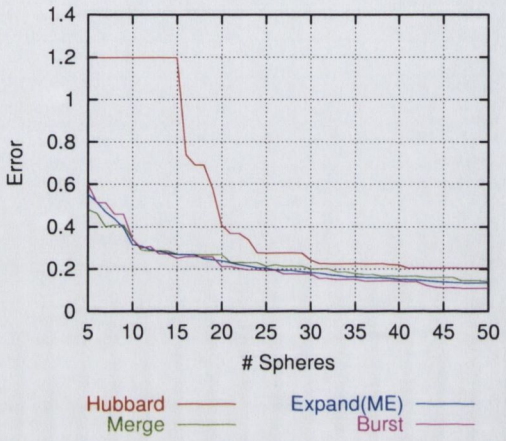


(c) Variance

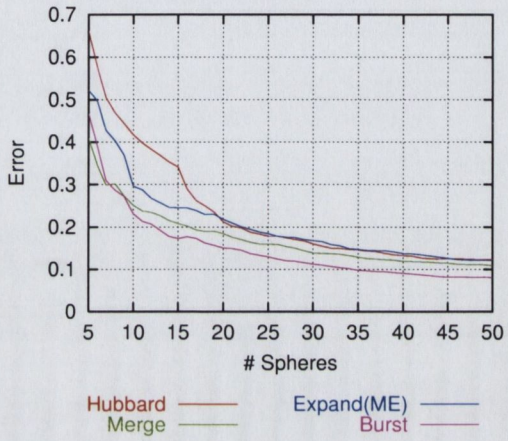


(d) Wasted Volume

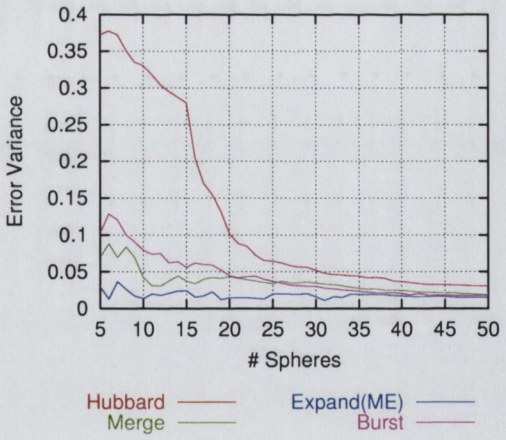
Figure 7.16: Comparison of sphere reduction techniques for the Cow.



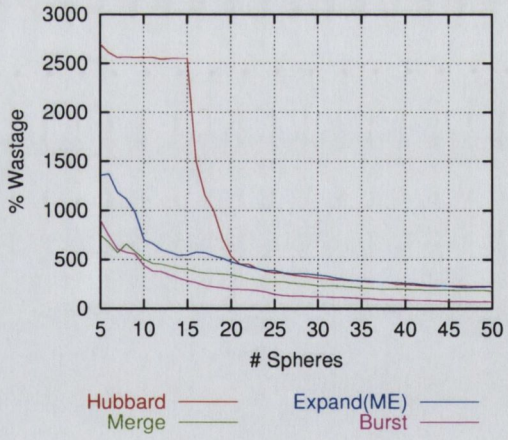
(a) Max Distance



(b) RMS Distance

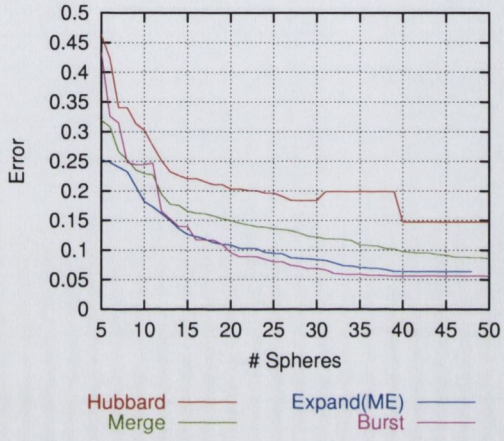


(c) Variance

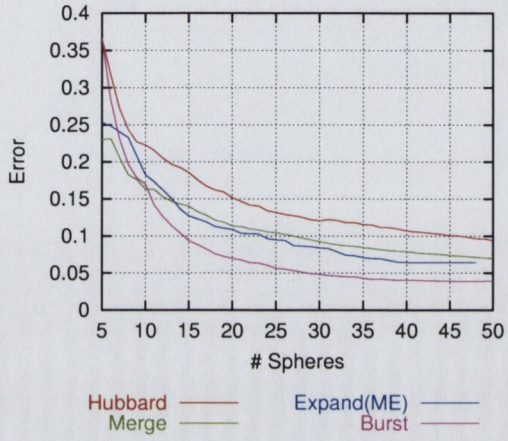


(d) Wasted Volume

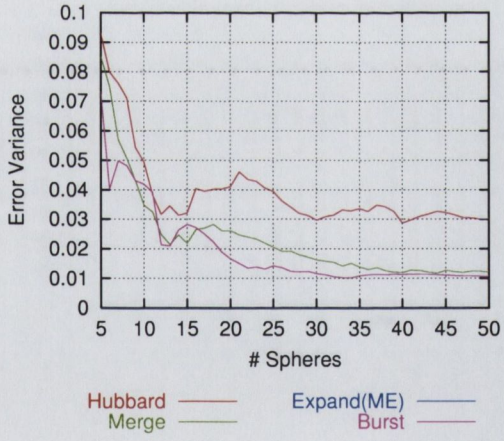
Figure 7.17: Comparison of sphere reduction techniques for the Dragon.



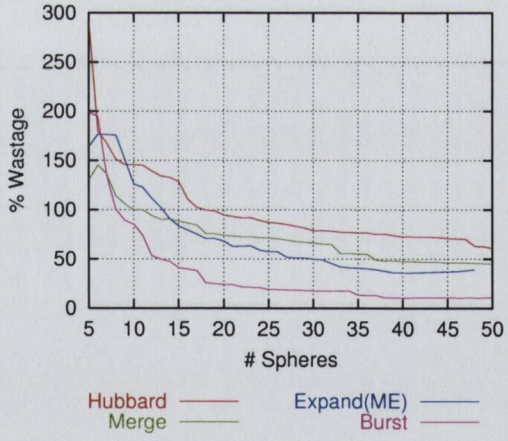
(a) Max Distance



(b) RMS Distance



(c) Variance



(d) Wasted Volume

Figure 7.18: Comparison of sphere reduction techniques for the cone.

it provides a reasonable level of sub-division without incurring too much computational cost within the traversal algorithm. It also represents the largest number of children that the Octree method can generate for each node. All the algorithms used a set of 5000 points to represent the surface of the object. Hubbard's algorithm used a medial axis approximation containing *circa* 2500 spheres. For the adaptive medial axis algorithms the initial medial axis contained 500 spheres and the approximation was dynamically refined so that each region had $\frac{1}{2}$ the error of the parent sphere and at least 100 spheres, i.e. the merge/burst/expand algorithms started with 100 spheres from which the 8 were to be produced. The grid, hybrid and optimiser algorithms all used Nelder and Mead's Downhill Simplex Method [84] to find local optima within their objective functions.

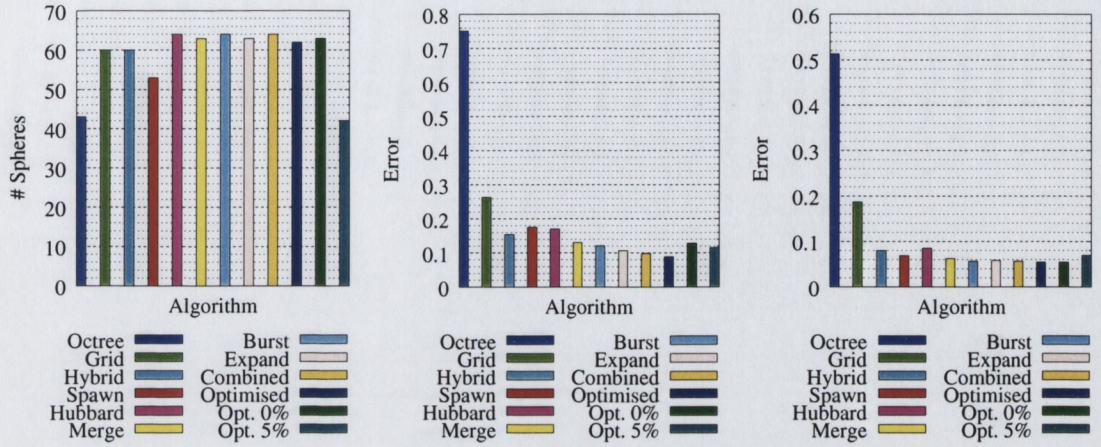
The first graph, in each set, compares the number of spheres generated with the various algorithms[3]. It is clear from the graphs that the algorithms do not always produce a complete sphere-tree, i.e. a sphere-tree that has the maximum allowable nodes for the given branching factor. The octree method always creates 8 spheres and discards the empty ones. While the grid, expand and spawn algorithms do aim to use as many spheres as allowed at each stage, they often do not achieve this - extra spheres will only ever be used if it provides a gain in fit. The sphere optimiser throws away spheres as long as an acceptable degree of fit remains. In these tests the algorithm was allowed to discard spheres provided the resulting worst error was less than a certain percentage of the initial worst error. The results for 100% and 105% are presented.

There is quite a subtle reason for merge/burst algorithms not generating the maximum allowable number of spheres. The sphere sets are generated by performing an iterative reduction of the set of medial spheres. The adaptive medial axis algorithm is used to ensure that each region of the object is approximated by about 10 times the desired number of spheres. However, the sets of spheres created using these algorithms often contain redundant spheres. These spheres contribute nothing to the approximation as the areas of the surface they cover are also covered by other spheres. Thus these spheres are discarded and their descendents are not computed. If this is an undesirable situation the algorithm can be modified so as to check for redundant spheres at each iteration and stop when the number of remaining (non-redundant) spheres reaches the required number.

The grid and hybrid algorithms consistently provide significant improvements over the octree algorithm, from which they were derived. Both the worst and RMS errors have been reduced. For complex model such as the Bunny (Figure 7.20) and the Dragon (Figure 7.22) the grid algorithm exhibits errors as low as $\frac{1}{4}^{th}$ that of the octree, while the minimum error spheres fitted over the same regions have as little as $\frac{1}{7}^{th}$ the worst error. It is also clearly visible that the grid based algorithms use more of the allowable number of spheres, which gives them the freedom to achieve a much tighter fit.

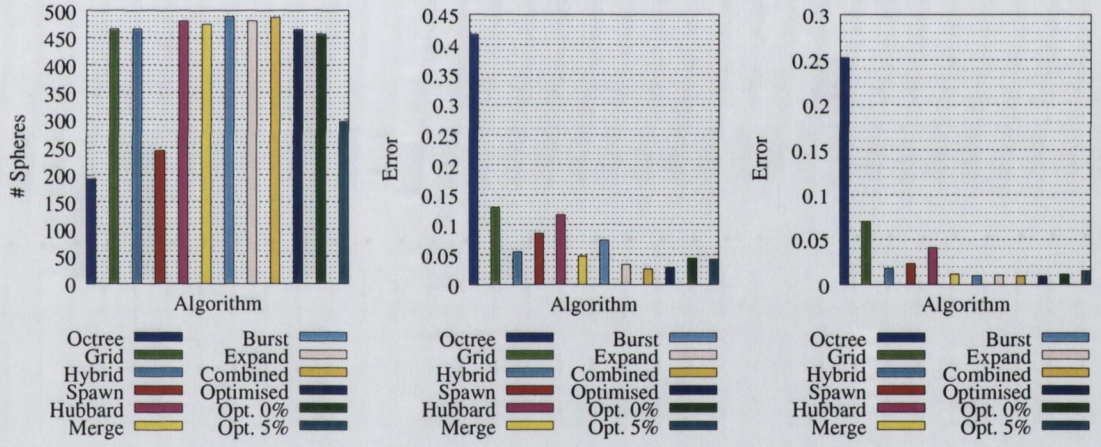
The medial axis algorithms approximate the object to a higher degree than the octree

[3]Different levels of the sphere-trees are presented separately for clarity.



(a) # Spheres (b) Worst (c) RMS

Figure 7.19: Comparison of sphere-trees for the Bunny. (Level 2)

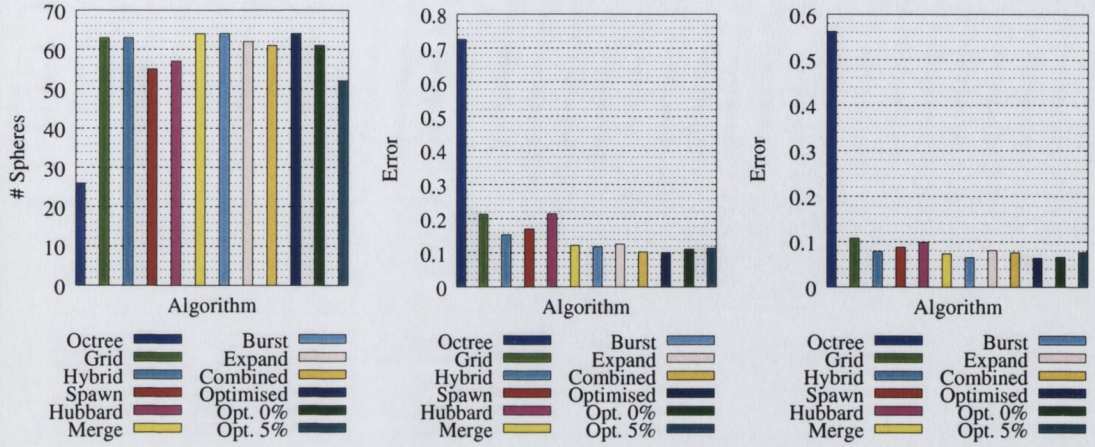


(a) # Spheres (b) Worst (c) RMS

Figure 7.20: Comparison of sphere-trees for the Bunny. (Level 3)

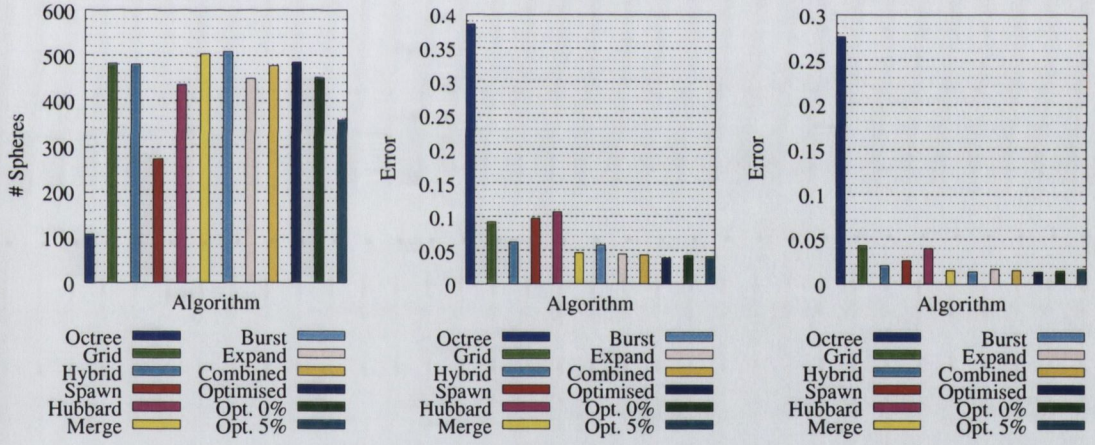
	Level 1		Level 2		Level 3	
	Max	RMS	Max	RMS	Max	RMS
$\frac{Hubbard}{Merge}$	1.56	1.32	1.30	1.36	2.47	3.44
$\frac{Hubbard}{Expand}$	1.61	1.25	1.59	1.46	3.44	4.03
$\frac{Hubbard}{Combined}$	1.60	1.24	1.72	1.49	4.48	4.24
$\frac{Octree}{Grid}$	1.80	2.44	2.85	2.75	3.23	3.60
$\frac{Octree}{Hybrid}$	2.98	4.33	4.89	6.40	7.58	14.00

Table 7.2: Improvements in fit of sphere-trees constructed for the Bunny.



(a) # Spheres (b) Worst (c) RMS

Figure 7.21: Comparison of sphere-trees for the Dragon. (Level 2)

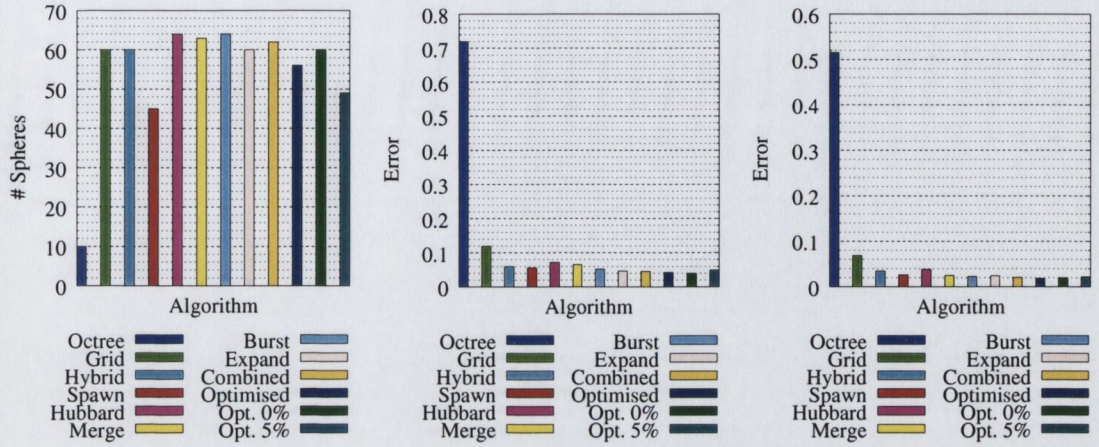


(a) # Spheres (b) Worst (c) RMS

Figure 7.22: Comparison of sphere-trees for the Dragon. (Level 3)

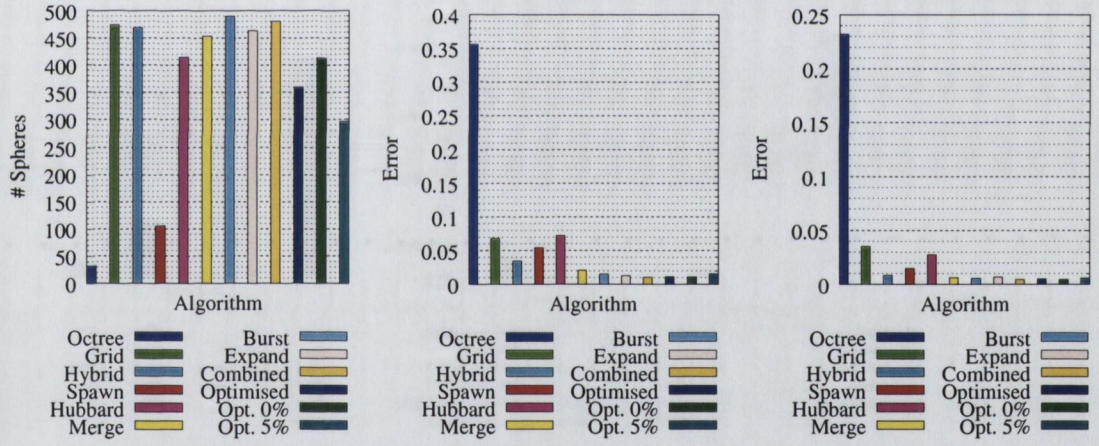
	Level 1		Level 2		Level 3	
	Max	RMS	Max	RMS	Max	RMS
$\frac{Hubbard}{Merge}$	1.59	1.35	1.75	1.34	2.30	2.58
$\frac{Hubbard}{Expand}$	1.50	1.21	1.70	1.22	2.40	2.38
$\frac{Hubbard}{Combined}$	1.58	1.35	2.08	1.30	2.49	2.60
$\frac{Octree}{Grid}$	3.31	3.86	3.40	5.20	4.18	6.43
$\frac{Octree}{Hybrid}$	3.97	4.44	4.75	7.16	6.18	13.43

Table 7.3: Improvements in fit of sphere-trees constructed for the Dragon.



(a) # Spheres (b) Worst (c) RMS

Figure 7.23: Comparison of sphere-trees for the S-shape. (Level 2)

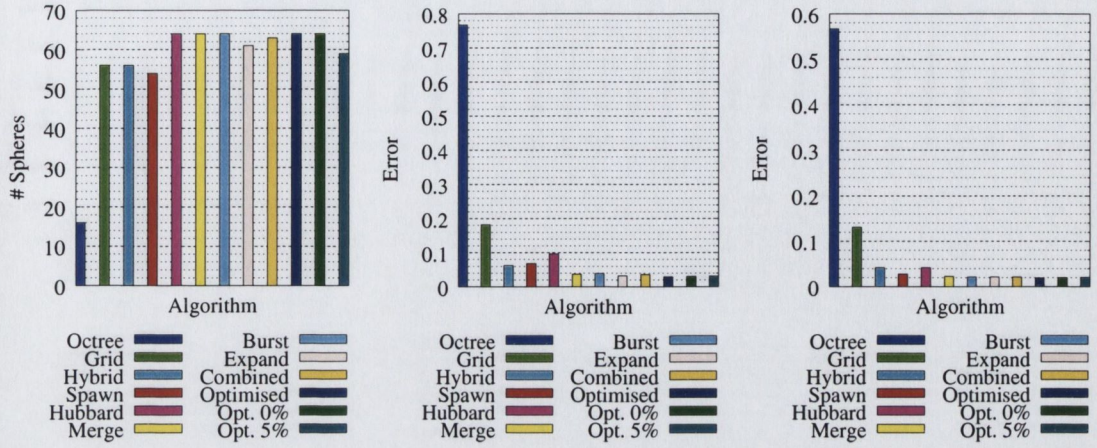


(a) # Spheres (b) Worst (c) RMS

Figure 7.24: Comparison of sphere-trees for the S-shape. (Level 3)

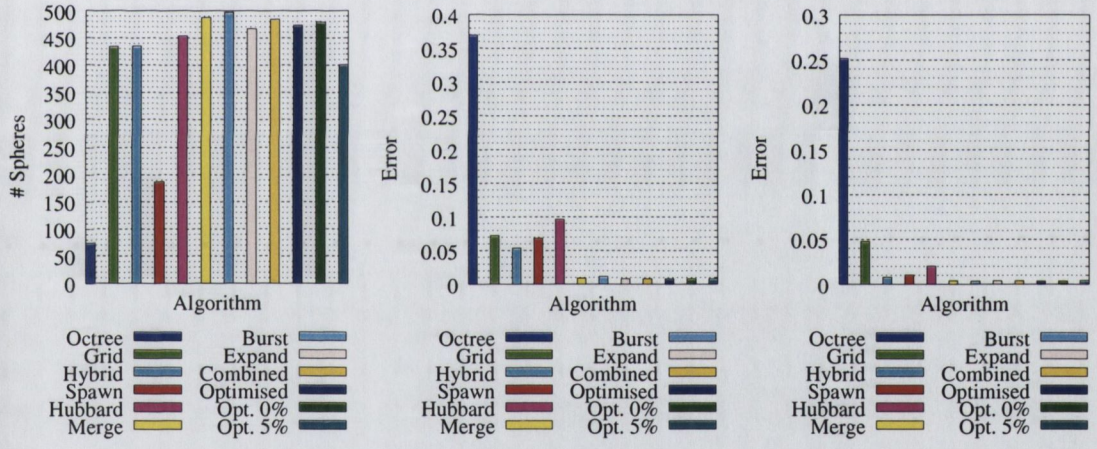
	Level 1		Level 2		Level 3	
	Max	RMS	Max	RMS	Max	RMS
$\frac{Hubbard}{Merge}$	1.35	1.34	1.11	1.53	3.43	4.30
$\frac{Hubbard}{Expand}$	1.73	1.42	1.56	1.59	5.74	3.99
$\frac{Hubbard}{Combined}$	1.58	1.39	1.60	1.86	6.86	6.07
$\frac{Octree}{Grid}$	6.84	7.01	6.02	7.50	5.23	6.57
$\frac{Octree}{Hybrid}$	8.57	7.95	11.96	14.63	10.23	27.80

Table 7.4: Improvements in fit of sphere-trees constructed for the S-Shape.



(a) # Spheres (b) Worst (c) RMS

Figure 7.25: Comparison of sphere-trees for the Ellipsoid. (Level 2)



(a) # Spheres (b) Worst (c) RMS

Figure 7.26: Comparison of sphere-trees for the Ellipsoid. (Level 3)

	Level 1		Level 2		Level 3	
	Max	RMS	Max	RMS	Max	RMS
$\frac{Hubbard}{Merge}$	1.21	1.19	2.59	1.83	10.31	5.18
$\frac{Hubbard}{Expand}$	1.64	1.40	2.93	1.94	11.76	4.98
$\frac{Hubbard}{Combined}$	1.54	1.35	2.65	1.94	11.65	5.16
$\frac{Octree}{Grid}$	3.84	4.96	4.21	4.33	5.12	5.26
$\frac{Octree}{Hybrid}$	4.83	5.63	12.30	13.43	6.88	31.87

Table 7.5: Improvements in fit of sphere-trees constructed for the Ellipsoid.

based algorithm. This can be attributed to the explicit use that is made of the object geometry and the flexibility in the placement and size of the spheres produced. The new merge algorithm shows significant improvements over the previous algorithm. For both the bunny and the dragon, the new merge has a worst error that is less than $\frac{1}{2}$ that of Hubbard's algorithm. In fact, for both the bunny and dragon, the worst case for the new algorithm's level 2 spheres (Figures 7.19(b) & 7.21(b)) is the about the same as the level 3 spheres made with the old merge algorithm (Figures 7.20(b) & 7.22(b)). Thus the new algorithm produces the same tightness of fit using around $\frac{1}{8}^{th}$ the number of spheres (*circa* 64 and 512 spheres respectively). The burst and expand algorithms produce similar results to the new merge algorithm. The burst algorithm produces its best results for the top levels of the spheres trees. This can be attributed to the way the sphere-tree construction algorithm divides the object. At the lower levels, the regions to be approximated tend to be largely to one side of the object which prevents the burst algorithm from producing spheres which span the object. This results in the spheres being very much towards the surface of the object and having a higher level of error. For the Bunny and the S-shape the expand algorithm provides further improvement over the new merge algorithm. There are however regions in which the expand algorithm can behave quite poorly, thus allowing the combined algorithm to choose between "merge" and "expand" for each node allows it to achieve the tightest fit. The worst case error for the combined algorithm being as low as $\frac{1}{2}$ that of the new merge algorithm (Table 7.4).

Sphere selection based techniques, such as expand and spawn, aimed to distribute the error evenly between the spheres in a given set. This allows for approximations that exhibited a low variance, in terms of error. However, for sphere-tree construction, each set of spheres is produced independently which means that the variance across a level of the sphere-tree will not be kept small. For the lowest level of both the Bunny (Figure 7.20) and the Dragon (Figure 7.22) the spawn algorithm achieves a similar error to the Hubbard's algorithm using as few as half the number of spheres. As the spawn algorithm does not explicitly use the medial axis approximation, it is unable to achieve the same level of fit as the medial axis based algorithms.

The sphere set optimisation algorithm was also tested. This algorithm aims to improve the arrangement of the spheres so as to decrease the error in the approximation. When the algorithm was allowed to throw away as many spheres as it could without increasing the worst error by more than 5%, the approximations contain pretty much the same amount of error as the unoptimised ones, while throwing away as much as $\frac{2}{5}$ of the spheres, see Figures 7.20 & 7.24.

7.2 Simulation

In order to evaluate the sphere-trees generated with the algorithms presented in Chapters 3 and 5, a number of simulations were built. During these simulations, the objects

were positioned and oriented randomly about a sphere and were given a random velocity towards its center. Each sphere-tree approximation of the object is evaluated during the simulation. At each time-step the colliding pairs are created by testing the bounding spheres of the objects. The sphere-trees for the potentially colliding objects are then traversed using the algorithm presented in Section 2.7.1. Separate traversals are conducted using the sphere-trees constructed using the Octree method, Hubbard's medial axis method etc. The motions of the objects and their response to collisions has been computed using Havok's dynamics system[4]. This uses an exact collision detection algorithm, and a dynamics model that includes complex friction. Using an exact collision detection algorithm avoids bias towards or against a particular sphere-tree algorithm and represents a situation where the sphere-trees are traversed to a very deep level before interruption.

The sphere-tree traversals are interrupted at regular intervals and the accuracy of the approximated collisions is measured. The error associated with each pair of colliding spheres is computed as the sum of the spheres' errors. As the error associated with a sphere is the largest distance from the surface of the sphere to the surface of the object (Hausdorff distance) this provides an upper-bound for the true separation between the objects. At each interruption the worst, best and RMS errors, in the approximated collisions, are computed. Each of the new algorithms are compared to a reference sphere-tree, i.e. the medial axis based algorithms are compared to the sphere-tree constructed with Hubbard's algorithm and the octree-like sphere-trees are compared to those constructed using the octree method. For each interruption time, the average improvements is computed. This is expressed as a fraction of the reference tree's error. For example, for each frame (at a given interruption interval), the merge sphere-tree is evaluated by computing $\frac{Error_{Merge}}{Error_{Hubbard}}$. These values are then averaged over all the frames of the simulation. In order to provide a sufficiently large set of frames, the simulation is run a number of times with a new random position (and orientation) for each object.

Figure 7.27 - 7.34 present results for simulations containing 20 objects. Sphere-trees constructed with the Merge and Expand algorithms and using the optimiser are compared to those constructed using Hubbard's algorithm. The GRID and hybrid sphere-trees are compared to the sphere-trees constructed with the Octree method. To allow these algorithms to be compared to Hubbard's algorithm, the sphere-trees made with Hubbard's algorithm are also evaluated relative to the Octree. The horizontal axis of each of the graphs shows the interruption interval. This is expressed in terms of the number of primitive operations performed, i.e. sphere updates and overlap tests. Amount of work done before interruption is computed using Equation 2.1, with $C_u = 2$ and $C_v = 1$. These values are used as they represent the relative number of floating point operations performed in updating a sphere's position (21 floating point operations) and in testing two spheres

[4]An evaluation version of Havok's software is available from http://www.havok.com.

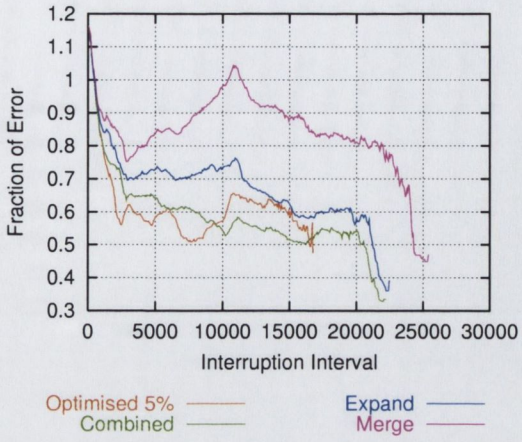
for overlap (10 floating point operations). The first three graphs (a), (b) & (c) in each set show the fraction of error present in the approximations. The fourth graph (d) shows the relative number of colliding pairs produced by the different algorithms. The fifth graph (e) shows the relative number of frames that still had unresolved collisions at the various interruption points, i.e. as compared to the reference sphere-tree.

Each of the medial axis based algorithms show a definite reduction in the worst error. For the Bunny and the Dragon (Figures 7.27(a) & 7.28(a)) the combined algorithm quickly falls to as little as 50% of the worst error present in the reference sphere-tree. For the simpler shapes, this value is as low as 20%. For the unoptimised sphere-trees the amount of error continues to decrease, to as low as 30% for the complicated models and 10% for the simpler ones. The sphere-trees that have been optimised show their greatest gain when interrupted early on. When the optimisation algorithm chooses to discard some of the spheres it makes the sphere-trees more efficient to traverse in the early stages but ultimately affects the final accuracy of the approximations. In terms of best error, the new medial axis based sphere-trees show a slight increase. The algorithms treat the worst case scenario as being more important than the average/best case and so are designed to minimise the worst error in the approximation.

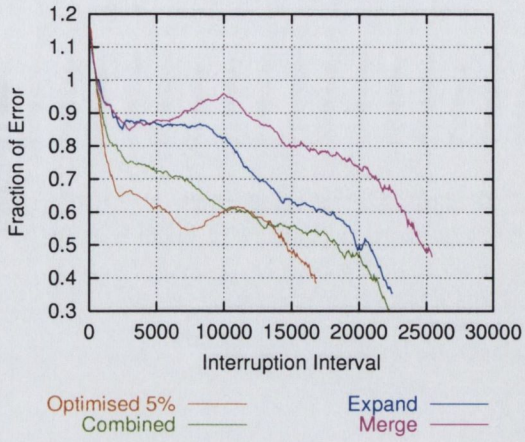
The algorithms also show significant reductions in the numbers of pairs of colliding spheres that result from the traversal. This provides a reduction in the amount of work that will need to be done by the later stages of the collision handling system, i.e. contact modelling and collision response. For both the Bunny (Figure 7.27(d)) and the Dragon (Figure 7.28(d)) the number of colliding spheres is as low as 20%.

The fifth graph (e) shows the relative number of frames that still had unresolved collisions at the various interruption intervals. All the improved medial axis methods show a reduction in the numbers operations required to resolve the frames to the lowest level of the sphere-tree (which are all 3 levels deep with circa 512 leaves). For all the models, when interrupted after 15000 operations, the optimised sphere-trees have less than $\frac{1}{2}$ the number partly resolved frames as those created with the reference algorithm. The Bunny seems particularly good in this respect with as few as $\frac{1}{20}^{th}$ the number of frames requiring 15,000 or more operations to resolve the collisions (Figure 7.27(e)).

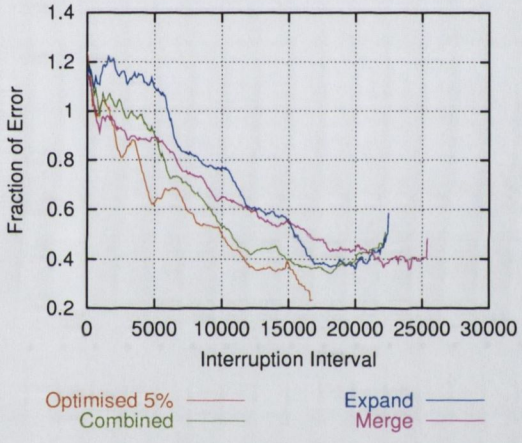
The GRID and Hybrid algorithms also show vast improvements over the Octree based algorithm. For all the models, the worst error quickly drops down to 10% of the error present in the Octree based approximation. For the Hybrid sphere-tree it is possible to traverse the set of minimum volume spheres (labelled MV) or the minimum error spheres (labelled ME). The minimum error spheres are always used to approximate the final set of contacts and there does not seem to be a significant difference between the two sets of spheres. Again graph (e) shows the number of frames that still had unresolved collisions at the point of interruption. Even though the grid and hybrid algorithms tend to produce sphere-trees containing a lot more spheres than the octree, the number of partially resolved frames, at each interruption time, is still reduced. For both the Bunny



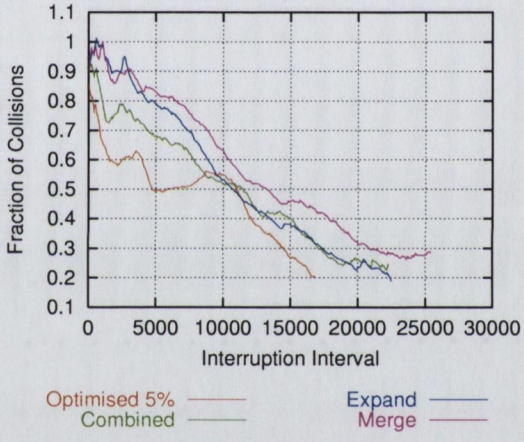
(a) Worst Error



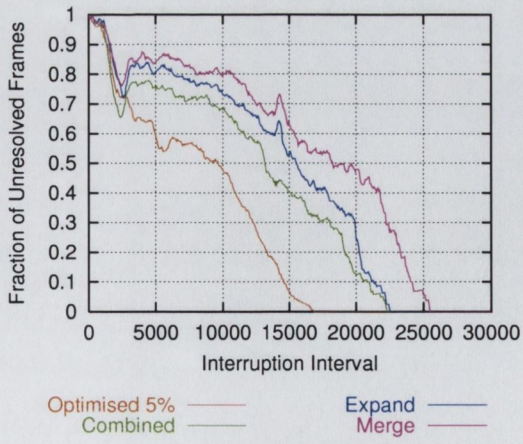
(b) RMS Error



(c) Best Error

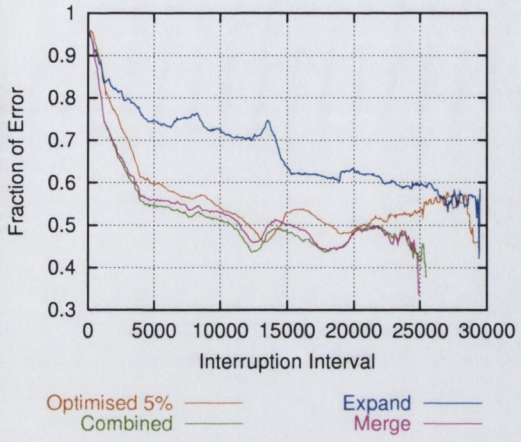


(d) Resulting Contacts

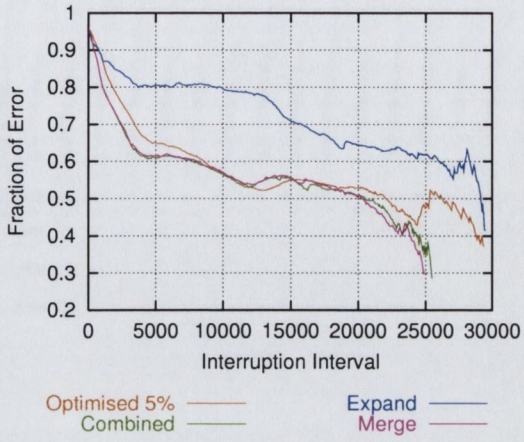


(e) Frames Evaluated

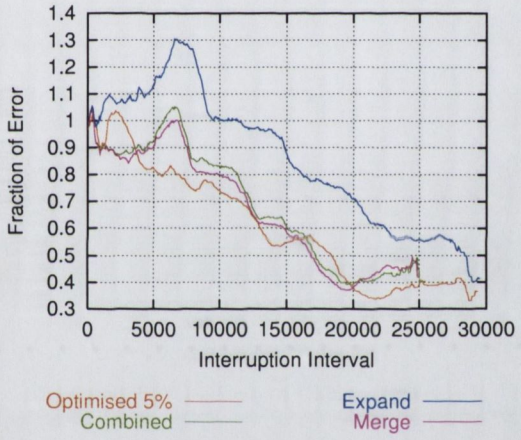
Figure 7.27: Comparison of medial axis based sphere-trees at various interruption times for The Bunny (20 objects).



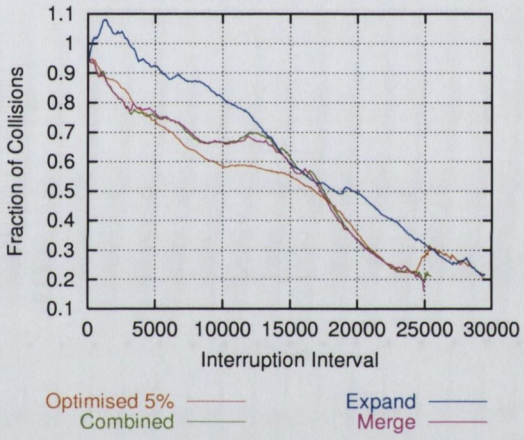
(a) Worst Error



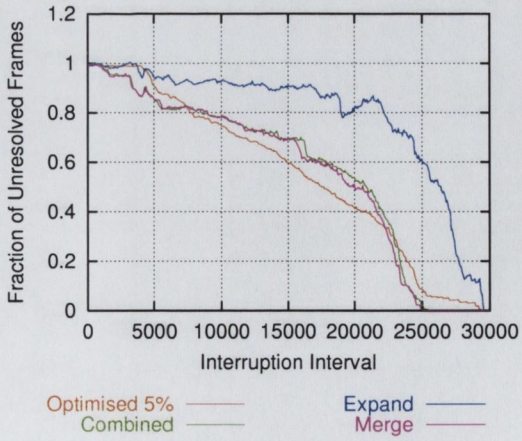
(b) RMS Error



(c) Best Error

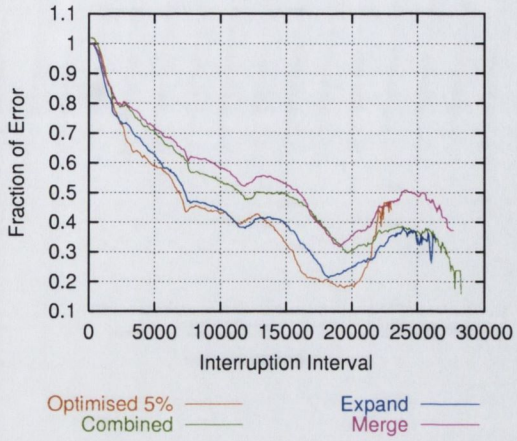


(d) Resulting Contacts

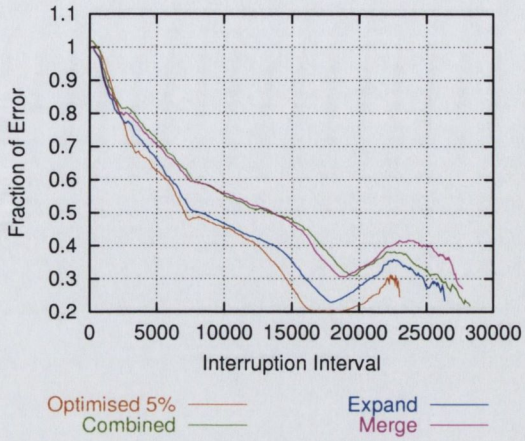


(e) Frames Evaluated

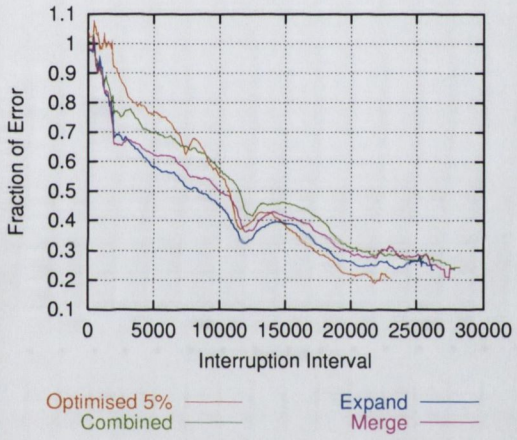
Figure 7.28: Comparison of medial axis based sphere-trees at various interruption times for The Dragon (20 objects).



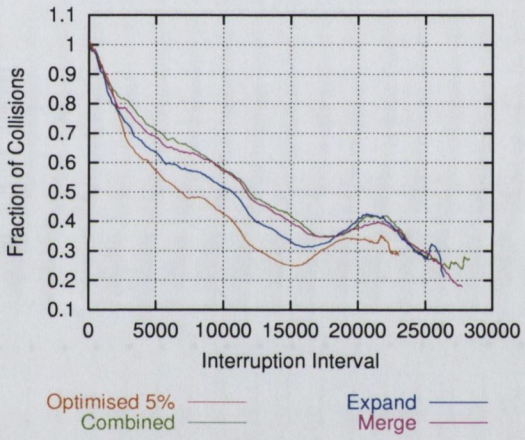
(a) Worst Error



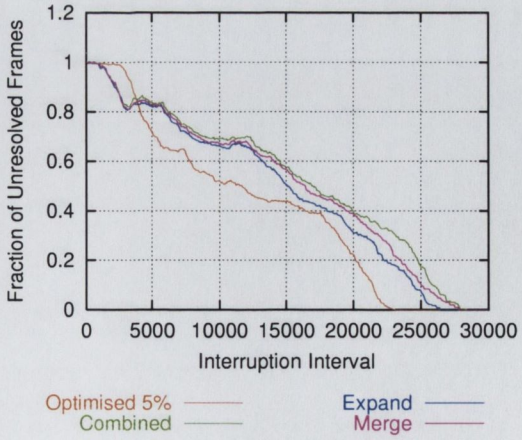
(b) RMS Error



(c) Best Error

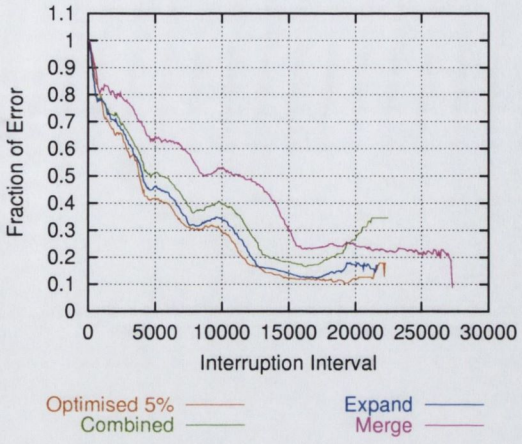


(d) Resulting Contacts

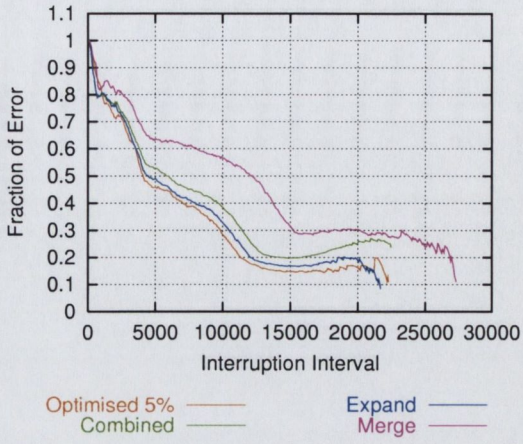


(e) Frames Evaluated

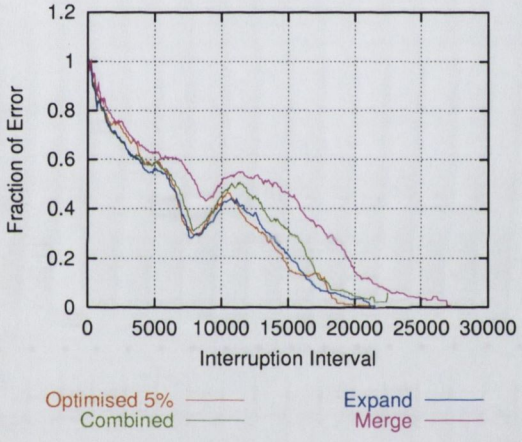
Figure 7.29: Comparison of medial axis based sphere-trees at various interruption times for the S-shape (20 objects).



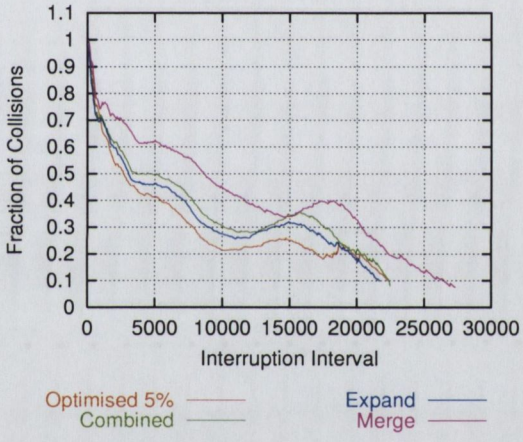
(a) Worst Error



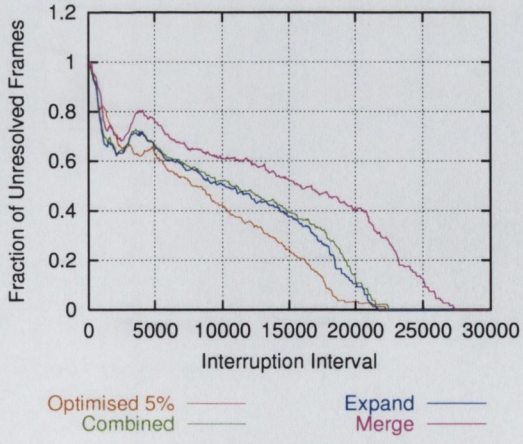
(b) RMS Error



(c) Best Error



(d) Resulting Contacts



(e) Frames Evaluated

Figure 7.30: Comparison of medial axis based sphere-trees at various interruption times for the ellipsoid (20 objects).

and Dragon (Figures 7.31(e) and 7.32(e)) the number of frames that still have unresolved collisions after 15,000 operations has been reduced to about $\frac{1}{4}$.

7.3 Conclusion

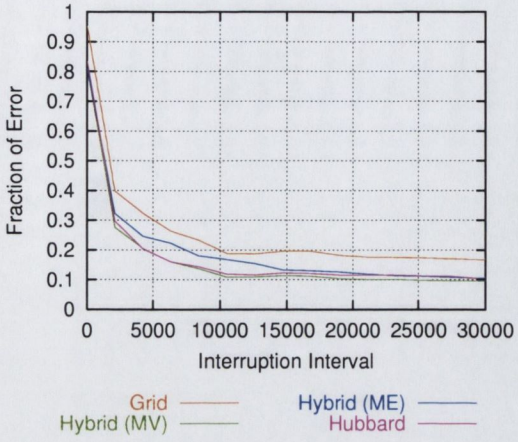
This chapter has taken a critical look at the work contained within this thesis. Both existing and new algorithms were compared. This analysis first looked at the algorithms for constructing the medial axis approximation, looking at both the regular and adaptive sampling algorithms. The results indicate that the adaptive algorithm often starts out with a poorer approximation but quickly improves. A hybrid algorithm could be adopted, with the initial approximation being constructed using regular sampling and then improved using the adaptive algorithm, which would also fill in uncovered areas.

The sphere-tree generation algorithm is composed of a number of sub-problems, each of which aims to approximate a given region of the object with a small number of spheres. The sphere reduction algorithms, which perform this task, were compared. The improvements to Hubbard's merging strategy show an improvement in the tightness of the approximations generated. The expand algorithm often shows a further reduction in the worst error and produces approximations that exhibit very small variances in tightness. For constructing sphere approximations, the burst algorithm shows significant improvements in tightness of fit over all the other algorithms.

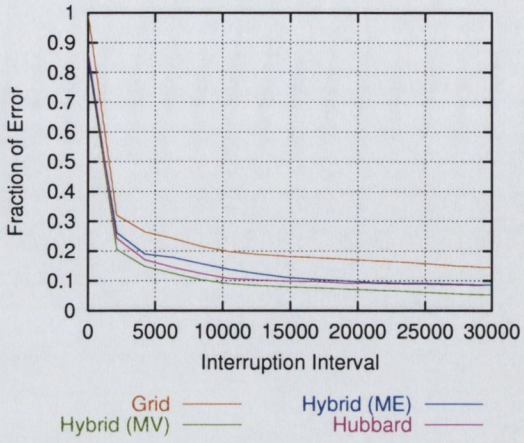
The algorithms were next used to construct sphere-trees for various models. Significant improvements were demonstrated for both the octree based and medial axis based algorithms. For the more complicated models, such as the Bunny, the worst error in the second level (64 spheres) is about the same as the third level (512 spheres) constructed with Hubbard's algorithm. The use of the optimisation algorithms, presented in Section 6.2, allows the construction of sphere-trees with about the same level of error using $\frac{1}{3}^{rd}$ less spheres. The GRID and Hybrid algorithms also showed significant improvements over the Octree method.

These sphere-trees were also evaluated for use in an interruptible collision detection system. The sphere-trees generated with the improved medial axis algorithms result in collisions with as little as 20% of the error present in those constructed with Hubbard's algorithm. Thus, they provide more accurate collisions and, as they fit the object tighter, result in less false positives. This produces much fewer pairs of spheres to be processed for collision response. For complicated models, such as the Bunny and the Dragon, as little as 20% the number of sphere pairs were produced. Also, the number partly resolved frames present after any given interruption interval has been significantly reduced. This results from the frames being resolved more quickly due to a decrease in the number of false positives experienced during the traversal.

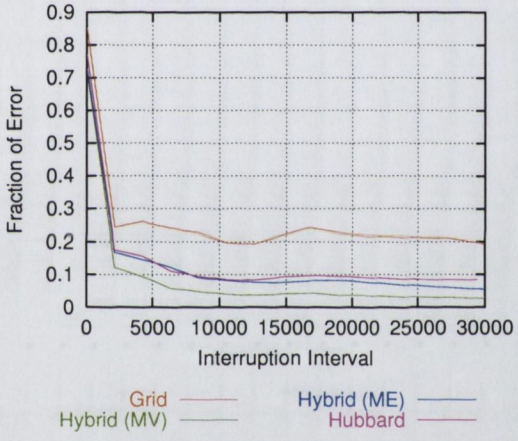
Similar results were presented for the Octree based algorithms. The Hybrid algorithm, which contains both minimum volume sphere and minimum error spheres, was introduced.



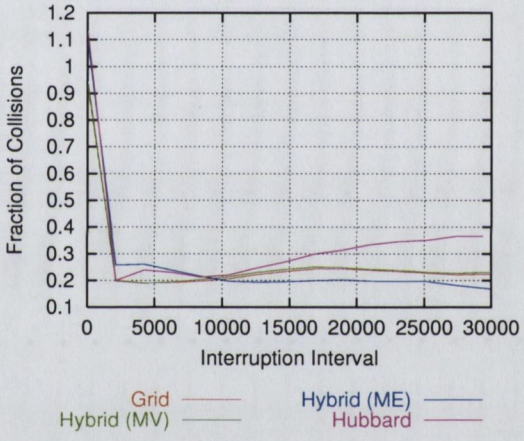
(a) Worst Error



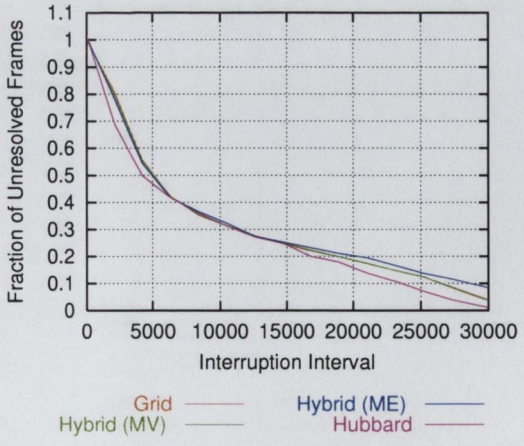
(b) RMS Error



(c) Best Error

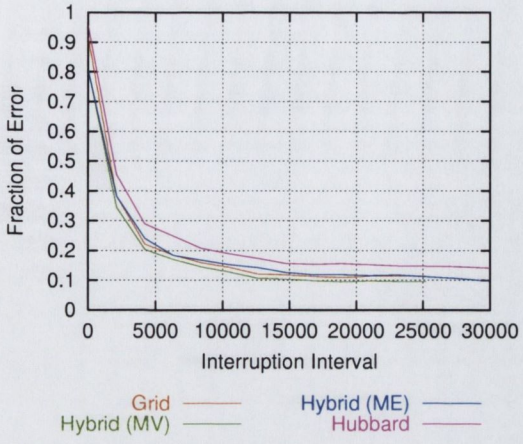


(d) Resulting Contacts

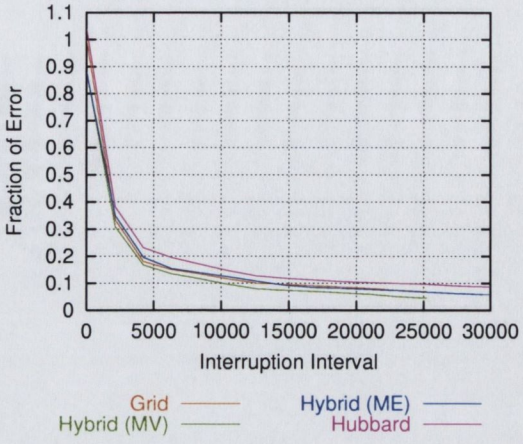


(e) Frames Evaluated

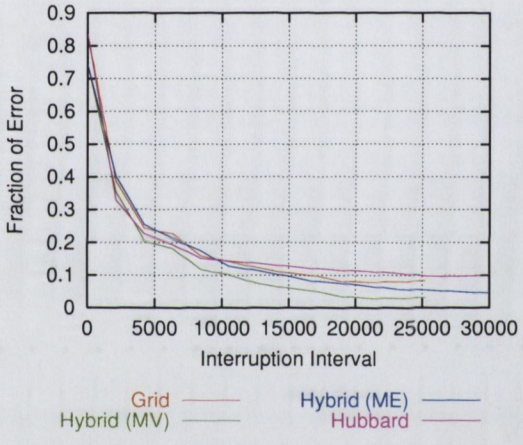
Figure 7.31: Comparison of octree based sphere-trees at various interruption times for The Bunny (20 objects).



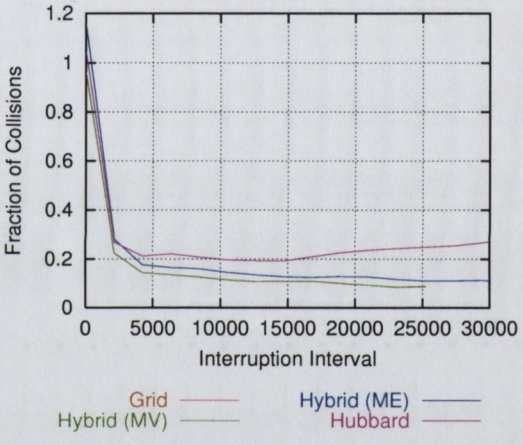
(a) Worst Error



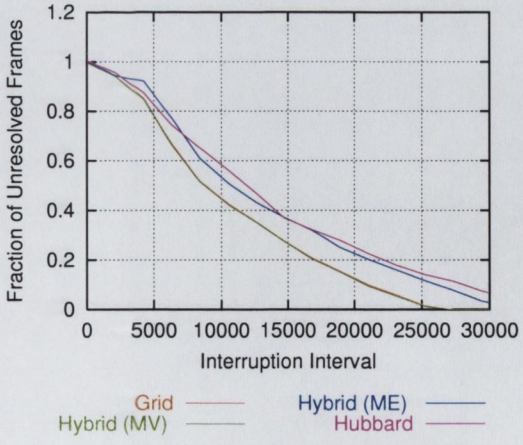
(b) RMS Error



(c) Best Error

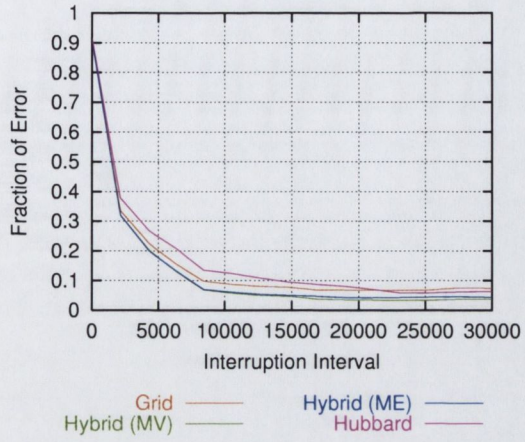


(d) Resulting Contacts

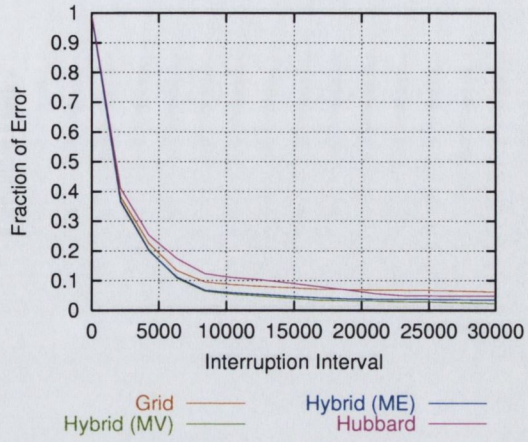


(e) Frames Evaluated

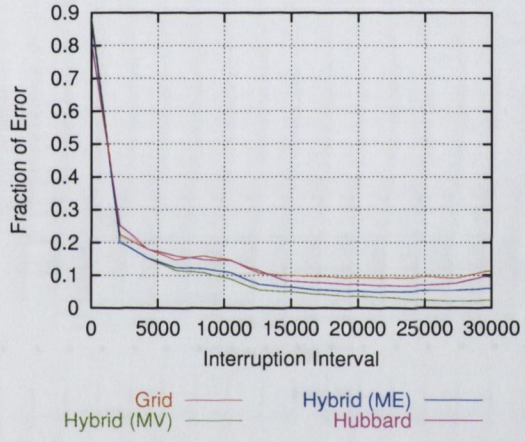
Figure 7.32: Comparison of octree based sphere-trees at various interruption times for The Dragon (20 objects).



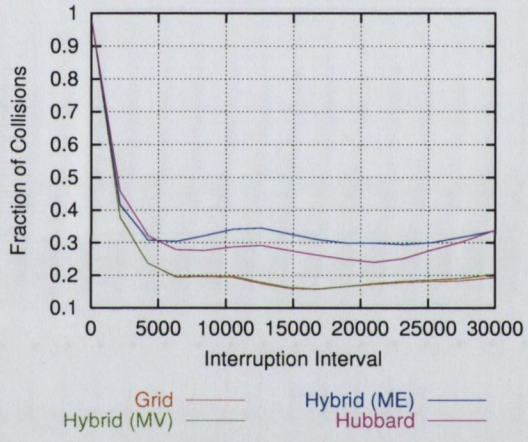
(a) Worst Error



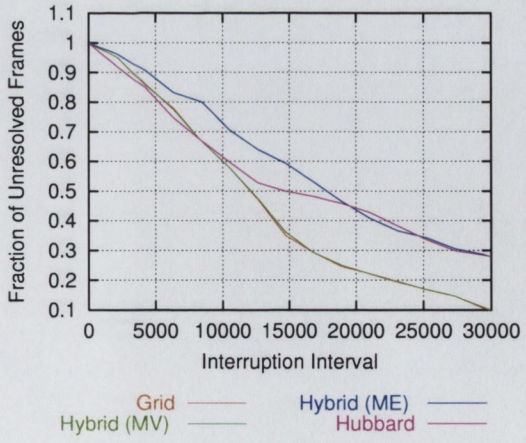
(b) RMS Error



(c) Best Error

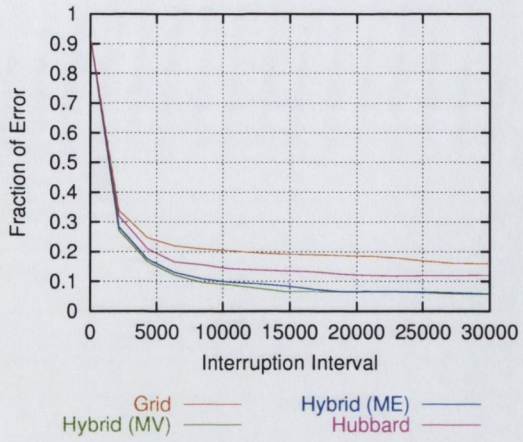


(d) Resulting Contacts

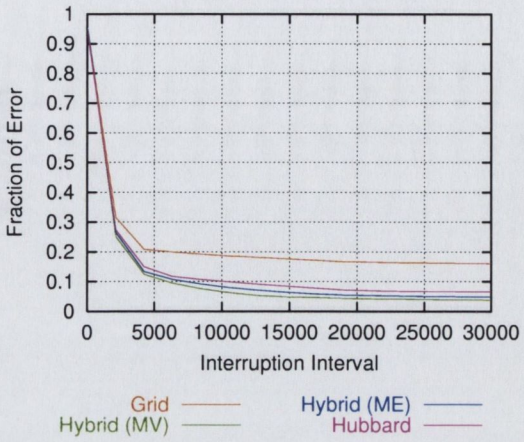


(e) Frames Evaluated

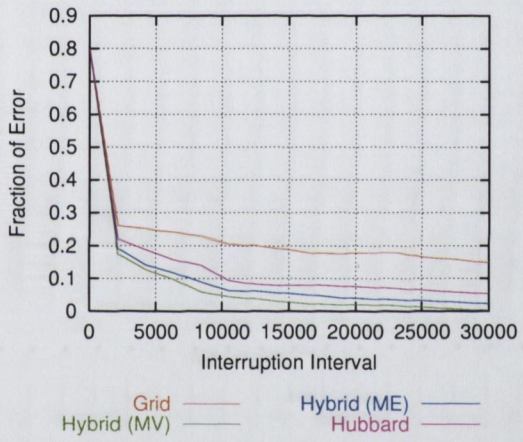
Figure 7.33: Comparison of octree based sphere-trees at various interruption times for the S-shape (20 objects).



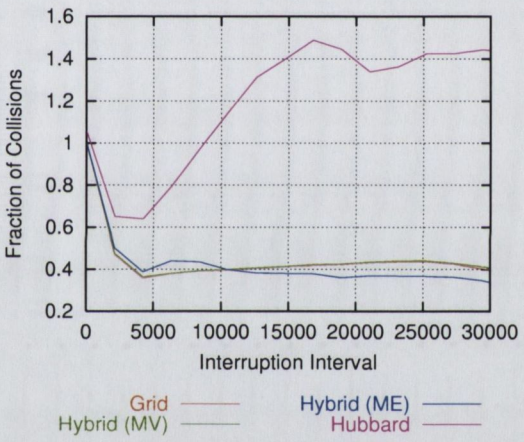
(a) Worst Error



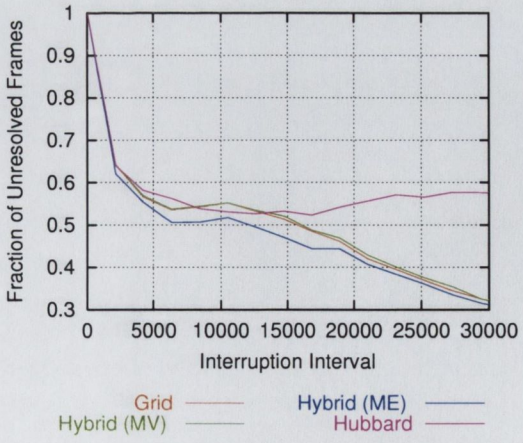
(b) RMS Error



(c) Best Error



(d) Resulting Contacts



(e) Frames Evaluated

Figure 7.34: Comparison of octree based sphere-trees at various interruption times for the ellipsoid (20 objects).

The octree based algorithms showed significant improvements over the original octree algorithm. For complicated models, such as the Dragon, the hybrid sphere-trees had as little as 10% of the error of those constructed with the original algorithm. The number of resulting colliding pairs was also reduced to as low as 10% the number resulting from the octree algorithm.

Chapter 8

Conclusions and Future Work

This chapter will bring together the material covered in this thesis. Conclusions will be presented on the work done in evaluating sphere-tree construction algorithms and their use in collision detection. The contributions of this thesis will be highlighted and a number of topics for further research will be presented.

8.1 Assessment

Collision detection is a major bottleneck for interactive systems. While some tasks require highly accurate collision detection others can use approximate techniques. Many researchers have utilised Bounding Volume Hierarchies, reviewed in Section 2.3, for accelerating collision queries.

Spheres offer distinct advantages for interruptible collision detection algorithms, described in Section 2.4, which trade accuracy for efficiency to provide consistent interactive frame-rates. As spheres are rotationally invariant they can be updated very efficiently during simulation. They also lead to very efficient overlap tests. As the interruptible collision detection algorithm may never fully resolve the collisions, the spheres themselves are often used to approximate the collision response.

Sphere-tree construction has been viewed from two different perspectives: Spatial subdivision techniques, such as the Octree method, and object approximation techniques, such as the Medial Axis method.

The work conducted can be summarised as follows :

- Chapter 3 took a critical look at the octree based method for sphere-tree construction. This is by far the simplest algorithm considered. A number of improvements were developed, which allow more freedom in how spheres are placed. The resulting algorithm was named the GRID algorithm.

- Chapters 4 and 5 analysed the second class of algorithms, the object approximation algorithms. An adaptive medial axis approximation method, which allows a

consistent and complete set of spheres to be constructed, was presented. This algorithm also allows the approximation to be constructed on demand so that it can be updated to provide a sufficiently tight set of spheres from which to construct each section of the sphere-tree. A number of alternative sphere set reduction techniques were presented. These techniques allow for the construction of the different approximations that make up a sphere-tree. The Expand algorithm was specifically designed for generating approximations that have a high level of consistency. An optimisation based approximation algorithm was also presented. This produces similar types of approximations as the Expand algorithm without requiring that a medial axis approximation be constructed.

- Chapter 6 presented a number of high level algorithms used in the sphere-tree construction process. A generic high level sphere-tree construction algorithm uses the various approximation algorithms to construct the sphere-trees. This algorithm controls how the object is partitioned so as to minimise the amount of redundancy in the hierarchy. A sphere-tree optimisation algorithm was also presented. This two phase algorithm optimises the sets of spheres and determines whether each sphere contributes anything to the approximation. Spheres that contribute little to the approximation are removed so as to reduce the cost of traversing the hierarchy.

- Chapter 7 details a series of experiments. These tests evaluated each stage of the approximation process. The two classes of algorithm are compared by using the resulting sphere-trees in a collision detection system. The conclusions drawn from these tests are as follows:

    1. The use of the adaptive medial axis approximation algorithm provides noticeable improvements. This algorithm allows a medial axis approximation to be constructed which guarantees to cover the surface. While the initial approximation is sometimes not as tight fitting as previous algorithms, the approximation quickly improves. The resulting sets of spheres produce tighter fitting approximations and more evenly distribute the error between them. See Section 7.1.2.
    2. The use of more accurate bounding sphere algorithms and sphere fit metrics provides benefits when reducing the number of spheres in an approximation. This allows the initial approximation to be reduced into various smaller approximations, while still maintaining a high degree of accuracy. Also, the approximations often exhibit a higher level of consistency. The Expand algorithm produces approximations with near zero variance across the entire region being approximated. See Section 7.1.4.
    3. Using the improved approximation techniques for the construction of sphere-trees produces considerable benefit for non-trivial objects. Even for complicated models, the second level of the sphere-trees has a similar error to the third level

of the trees constructed using existing techniques, i.e. the same tightness of fit with almost an order of magnitude less spheres. Further gain is achieved by using the optimisation algorithm, presented in Section 6.2. By allowing the algorithm to tolerate a small increase in error, the number of spheres in the approximation can be significantly reduced. See Section 7.1.5.

4. The sphere-trees, generated with the new algorithms, do show significant benefits to the process of collision detection. Having interrupted the sphere-tree traversal after a specified number of overlap tests, sphere-trees generated with the GRID algorithm show as low as 20% the error observed for the Octree based trees. The sphere-trees generated with the optimised medial axis method show as little as 20% the error of Hubbard's algorithm when interruption occurs. Combining both classes of sphere-tree construction algorithms, to produce a hybrid sphere-tree, produced further improvements over the GRID algorithm. The sphere-trees produced with the improved algorithms also reduce the number of colliding pairs that result from sphere-tree traversal. This shows a vast decrease in the number of false positives reported and will reduce the cost associated with computing the collision response. Dynamically choosing the branching factor of the hierarchy shows most improvement when interruption occurs early on. This is due to the object being segmented into fewer regions, which ultimately affects the accuracy of the resulting spheres-tree. See Section 7.2.

8.2 Contributions

A number of contributions have been made in this thesis:

- An adaptive medial approximation technique that allows the medial axis to be constructed on demand and focuses on improving the approximation in areas where the spheres created from the medial axis ill-fit the object.
- Improved sphere reduction techniques that reduce the spheres generated from the medial axis into a manageable set while maintaining a high degree of fit and consistency.
- A generic sphere-tree construction algorithm that decomposes the problem into smaller object approximation problems, which can then be solved using the algorithms presented. When dividing the object into sub-regions, the algorithm minimises the amount of overlap between neighbouring regions so as to eliminate redundancy within the sphere-trees.
- A sphere-tree optimisation algorithm that further improves the degree of fit in the approximations.

- Additional approximation algorithms based on generalising the octree sub-division to produce tighter approximations, by allowing more freedom in the way spheres are placed, while maintaining its good sub-division characteristics.
- A purely optimisation based approximation algorithm that yields similar results to those obtained using the medial axis but does not require the overhead of constructing the medial axis approximation.
- The introduction of a hybrid sphere-tree structure that contains both minimum error and minimum volume spheres at each node. These sphere-trees provide for both spatial localisation and object approximation.

8.3 Future Work

There are a number of interesting areas of research that can build upon this work. A number of the algorithms presented may be applicable to areas other than collision detection. The adaptive sampling algorithm, combined with the Expand algorithm, can provide a method for approximating rigid objects with very low variances. It is often very desirable to be able to approximate objects with a high level of consistency. Also this work has uncovered a number of related topics that merit investigation.

8.3.1 Ensuring Object Coverage

Throughout this thesis, objects have been represented as a set of surface points. This was done as it is much more efficient to check that all the surface points are covered than it is to work directly with the underlying polygons. This can however cause small areas of the object to be left uncovered. As these areas will be between the sample points, using more sample points will result in the smaller (and less frequent) gaps. One interesting alternative would be to use groups of points which are treated as atomic units. These points would represent triangles (or polygons) covering the entire surface. In order for a group to be considered "covered", all its points much be contained within the same sphere.

8.3.2 Combining Different Collision Detection Strategies

Buildings often contain large flat areas and are not in motion, therefore spheres are not very attractive for approximation purposes. Thus, it would be interesting to model these parts of the simulation with structures that are suitable for performing collision detection but are also more in line with their shape. One example of this is where we have used a grid of AABBs to model a height field, see Figure 8.1. This provides very efficient collision detection as the spheres are simply projected onto the ground plane to determine which AABBs to test.

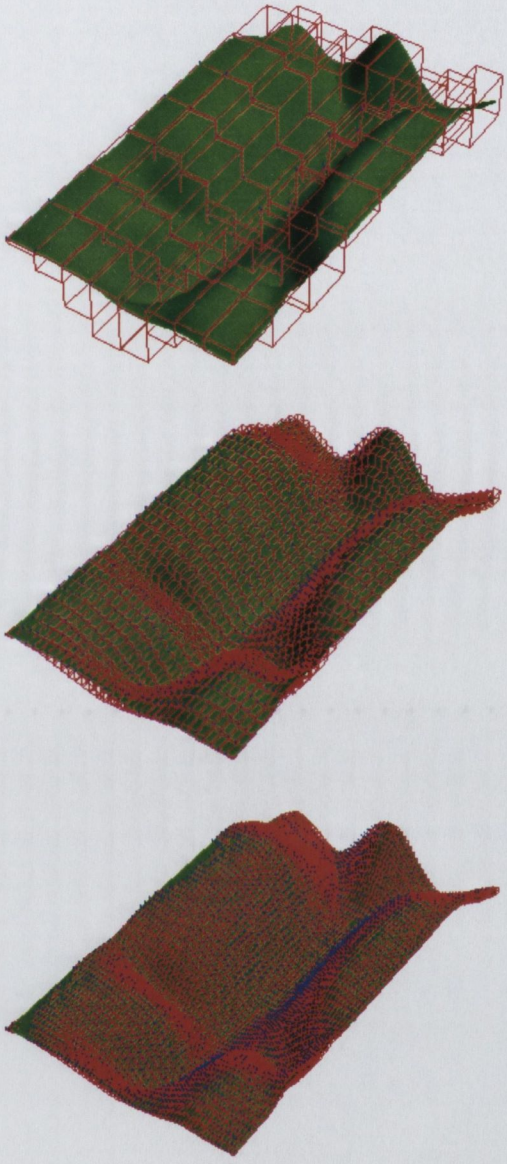


Figure 8.1: Examples of a terrain modelled by various sized AABBs for efficient collision detection.

8.3.3 Unified LOD Rendering and Collision Handling

In [92, 93], Rusinkiewicz and Levoy present a point based visualisation system called Q-Splat. In this system, sphere-trees are used to perform level-of-detail rendering and visibility culling. The sphere-tree is traversed in a depth-first manner. When the projected size of a sphere becomes significantly small, it is rendered using a splatting technique. In order to maintain interactivity this process can be interrupted when the user tries to rotate the object. Interactive frame rates are maintained by increasing the "splatting threshold", i.e. the size at which the spheres are considered small enough to splat. A very interesting area of research would be to use the same sphere-trees for collision and rendering. These sphere-trees would have to provide good spatial sub-division characteristics for visibility culling and closely approximate the object's geometry for collision detection.

8.3.4 Automatic Skinning of Models

Many of the algorithms in this thesis rely on being able to determine whether a point is inside the model being approximated. When approximating the medial axis, the vertices of the Voronoi diagram that are inside the object are used to create spheres. Also, when evaluating a sphere's fit to the surface, the sample points on the sphere need to be categorised as being inside or outside the object. Our inside/outside tests, discussed in Appendix A, assume that the surface of the object is closed. Each triangle must have exactly three neighbours and each edge must be shared by two triangles. Often, objects are modelled to produce a desired result when rendered. Modellers often construct their object using a number of meshes. For example when modelling an aeroplane, the wings and fuselage might be modelled with Bézier surfaces, while the finer trimmings are constructed by hand. Having multiple meshes is not a problem as the inside/outside test can check to see if the point is inside one of the meshes. However, meshes that are not closed, or have self-intersections result in incorrectly categorised points. This often causes problems when constructing sphere-trees for these objects. In order to approximate these objects, a closed surface needs to be constructed. The meshes used for the tests in this thesis were carefully checked before use. Further research into the construction of an outer skin for the models would make it much simpler to use generic models for collision detection.

8.3.5 Sphere-Trees for Deformable and Brittle Objects

The sphere-tree construction techniques presented in this thesis are primarily intended for use with rigid and articulated objects. The octree method is simple enough to be used to approximate deformable objects, i.e. the octree structure can be quickly updated as the objects deform. The GRID algorithm, presented in Section 3.3 may also be used for this purpose. Conceivably, this structure could be incrementally adjusted if the objects are deforming slowly. It would be interesting to investigate how this would perform in

a real-time system. This approach may also prove useful for brittle objects. When a fracture appears in the object, one of the branches of the sphere-tree could be broken off to produce an approximation of the fragment. It is unlikely that the medial axis based methods would be able to provide reasonable performance for these tasks as problems may be experienced due to the large spheres that are placed in the interior of the object. As these spheres often cross large portions of the model they may pose problems when the objects deform or shatter.

8.3.6 Hybrid Bounding Volume Hierarchies

Many researchers have used bounding volume hierarchies as a means of performing efficient collision detection. The bounding volumes used range in complexity. Swept sphere volumes, used by Larsen *et al.* [62], provide a number of different types of bounding volumes. The construction algorithm chooses which of the primitives to use for each region of the object. However, all the primitives available are based on spheres. Using a wider range of bounding volumes, such as those discussed in Section 2.3, could prove very interesting. Such a scheme would be able to choose the best primitive to use for each region. Although a potentially large number of intersection tests would be needed, the improved fit may compensate for this in certain situations.

8.3.7 When Spheres Are Bad Approximators

Finally, there are many types of geometry that are quite badly represented by spheres. At some point in the bounding volume hierarchy, many of the spheres will become small enough that they will only cover a single triangle/polygon in the original model. At this stage it would be more beneficial to test the actual polygon instead of a number of spheres. Once the sphere-tree traversal algorithm has reached this level of the approximation a simple change over can be made. This is subtly different from the exact phase algorithms, presented in Section 2.5, as only simple triangle-sphere and triangle-triangle tests would be required.

Appendix A

Surface Testing

There are a number of sections of this thesis that have relied on being able to determine whether a given point is inside the model being approximated. This appendix gives some information on how this can be achieved.

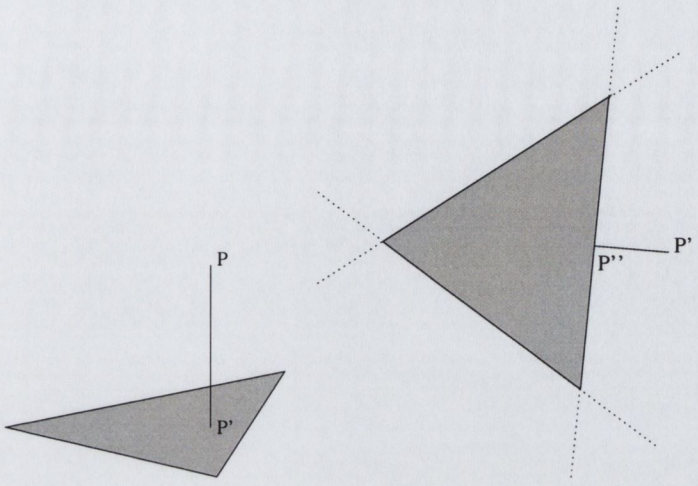
In his thesis, Hubbard presents an inside-outside test based on determining whether the given point is behind the closest part of the surface [49]. However, this test is conservative when identifying points as being internal. That is to say that it will only categorise the point as internal if it is definitely inside the object. There are, however, situations when points inside the object are labelled as external.

Using this test for determining the internal vertices of the medial axis does not pose a serious problem. It does, however, affect the measurement of fit mentioned in Section 6.2. This method operates on a number of points distributed across a sphere. For each of these points the error is measured as the distance to the surface. The actual error of a sphere is the maximum of the distances for the sample points that are outside the object. Thus, the mis-categorisation of points will affect this measure and invalidate the test results.

An overview of this algorithm and an alternative based on the crossings-test are given below. Both these tests require that the surface be represented by a polyhedral mesh and that the mesh be closed (2-manifold). Also, there is a requirement that the surface polygons do not cut through each other and enter the surface - for example a cup with a separate mesh for a handle can cause problems around where they join.

A.1 Closest Point Test

Hubbard's algorithm for categorising points as being inside or outside of the object proceeds as following. The closest point on the surface mesh is first determined. Using this point q the status of point p depends on whether the closest point lies on a face, an edge or a vertex. If the closest surface point q lies on a face then the point p can only be inside the object if the point p lies behind that face, i.e if the vector dot product between $\|p-q\|$ and the face normal, n, is negative. If q lies on an edge e then there are two cases that



(a) Inside (b) Outside

Figure A.1: Cases for when p projects into a triangle and for when p' projects onto an edge.

must be considered. If the two faces, which share e, form a concave fold then p must lie behind one of the faces to be inside, whereas if the two faces are convex then p must be behind both faces. If q is a vertex then p must lie behind all faces that share that vertex.

To find the point q on a triangulated polyhedron that is closest to a given point p the algorithm iterates through the triangles finding the closest point on each triangle. There are three cases that must be considered when trying to find this. The minimum distance between a plane and a point p is the perpendicular projection, p', of p onto the plane. If this projected point is inside the triangle then there exists no closer point on the triangle (see Figure A.1(a)). If p' lies outside an edge e of the triangle then the closest point between p and the edge e is the perpendicular projection, p'', of p' onto e. If p lies within the line segment then there can be no closer point on the triangle (see Figure A.1(b))[1]. If neither of the previous two cases are met then the closest point will be one of the vertices.

A SEADS grid (Spatially Enumerated Auxiliary Data Structure) is used to speed up the search for the closest triangle. The voxels of the SEADS grid contain lists of the triangles with which they intersect. Before checking the triangles within each voxel a quick distance check is performed. This eliminates the voxel if it does not contain any points closer than the minimum distance encountered so far. The voxels are considered in order of their distance from p so as to maximise the number of voxels that can be culled. Hubbard creates a list of voxels and sorts them according to their distance from p so as to terminate the search as quickly as possible. However, when performing inside/outside

[1]Hubbard incorrectly states that there can only be one edge onto which p' can perpendicularly project. The correct situation is that the only edge capable of producing the closest point is the one that makes p' lie outside the triangle.

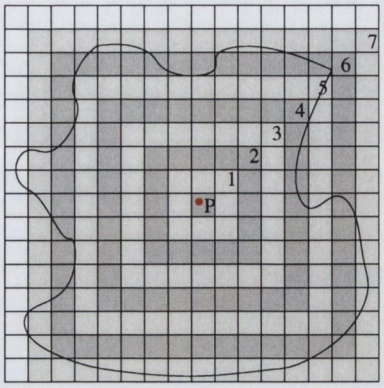


Figure A.2: Voxel traversal for finding the closest point on an object.

tests for this thesis, the sorting of the voxels accounted for a large amount of the work done. For a $15 * 15 * 15$ grid, the sorting (using quick sort) accounted for 80% of the total time for inside/outside tests.

An alternative strategy is to use a more complicated algorithm to consider the voxels in order. Instead of sorting a list of voxels, the algorithm implicitly considers the closest ones first and leaves the further ones until later. The search starts with the voxel that contains p. If p is not contained within a voxel then the test can return immediately as the voxels cover the entire object and therefore p has to be classified as being outside the object. The voxels are next considered to be a set of concentric rings, centered around the starting voxel as illustrated in Figure A.2. Each ring of voxels is considered in turn, starting with the one closest to p and moving outwards. This process continues until there are no more rings or the closest point on the ring is further away from p than the current estimate for its closest point. This doesn't necessarily consider the voxels in the same order as if they had been sorted by their distance from p. However, this scheme does allow a much finer SEADS grid to be used and does not incur the penalty of the sorting algorithm. Consequently, using this traversal algorithm has provided a significant speed-up (a factor of about 50) in the implementation used for this thesis.

A.2 Crossings Test

In the crossings test, based on the Jordan Curve Theorem, a ray is shot out from the query point in an arbitrary direction. The number of triangles that this ray intersects are counted. This number will be odd if and only if the point is inside the surface. The intersection test can be performed using any technique, such as that presented in [75]

There are a number of special cases that need to be considered. For instance, if the ray strikes an edge or a vertex then the number of triangles intersected may be incorrectly counted. This results from rounding errors in the arithmetic operations used to test the intersection. Thus, in order to ensure that the point be categorised correctly these cases

need to be considered. These cases are, of course, quite rare and for applications such as ray-tracing leaving out the special cases will occasionally result in an incorrectly coloured pixel (or part of a pixel). For testing sphere fit these small errors cause anomalies in the results. Therefore, the implementation used in this thesis fires a pair of rays and compares the answers. If the rays disagree another pair of rays are fired, which vote for the correct result.

Not all of the triangles need to be considered when counting the crossings. Any triangles that do not lie on the path of the ray can be ignored. This can be efficiently achieved by again using a SEADS grid. The voxels along the path of the ray are stepped through using a variation of the DDA algorithm [3]. Thus only the triangles that occupy these voxels need to be considered.

A.3 Speedup

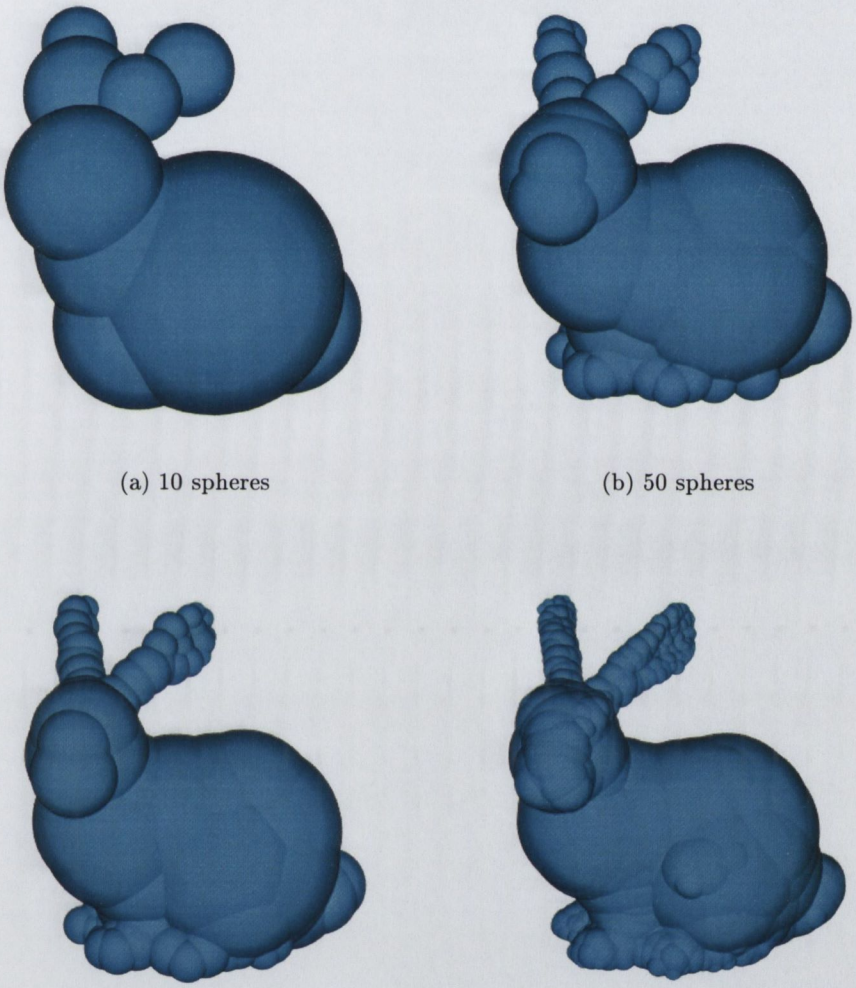
Whichever technique is being used to determine if a point is within an object, the test need not be used for every query. There are large areas of space that will be inside the object and others that will be outside. Again, a SEADS grid can be used to accelerate the process. Each of the voxels is categorised as being inside, outside or undetermined. The undetermined voxels are those that contain a section of the object's surface and, therefore, are only partly inside the object. If a voxel's state is not undetermined then it can be categorised with a single in-out test, i.e. testing the center point of the voxel.

Thus, when testing whether the point p is inside the object, a simple lookup is usually all that is required. If the point falls into a voxel that is categorised as in or out then no further work is needed. Only points that fall into voxels with the undetermined state need to be checked further, using the desired algorithm.

Appendix B

Examples

This appendix shows some examples of the sphere generation algorithms. The first set of pictures show the bunny, cow and dragon approximated using a single set of spheres. The second set show some sphere-trees generated for these models. Each of the sphere-trees has a branching factor of 8 and is generated using the adaptive medial axis methods discussed in this thesis.



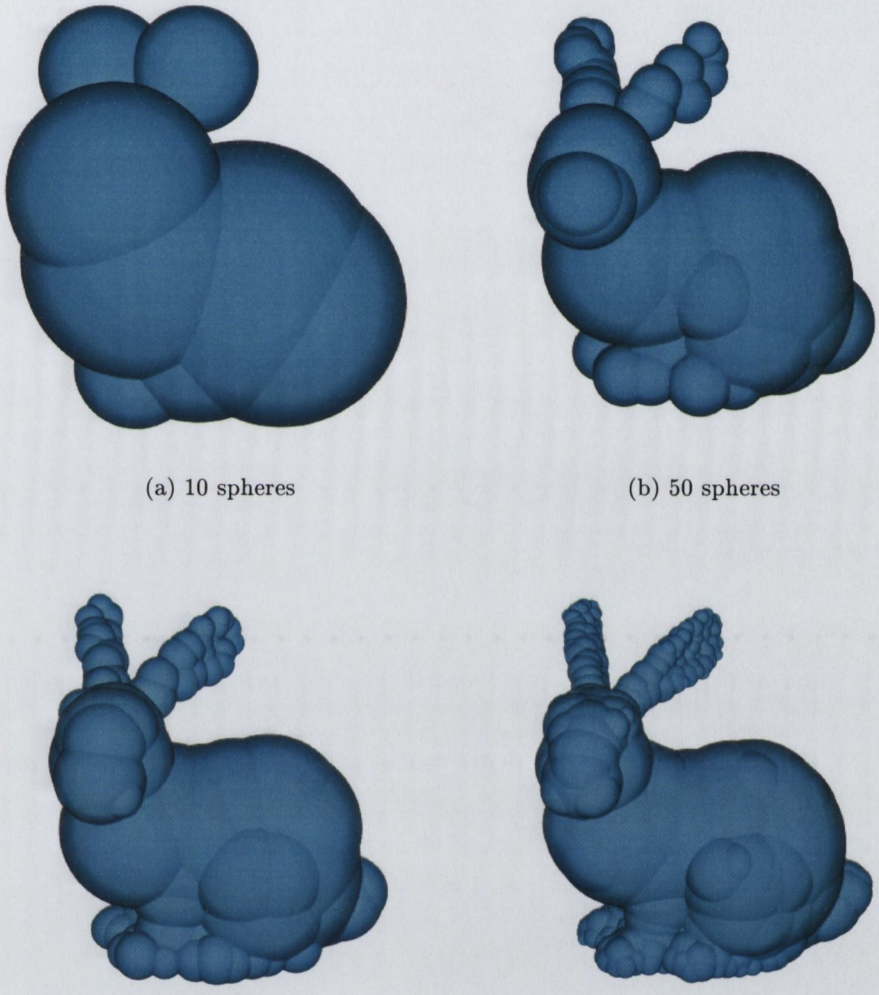
(a) 10 spheres

(b) 50 spheres

(c) 100 spheres

(d) 500 spheres

Figure B.1: Examples of the Bunny approximated with the Merge algorithm.



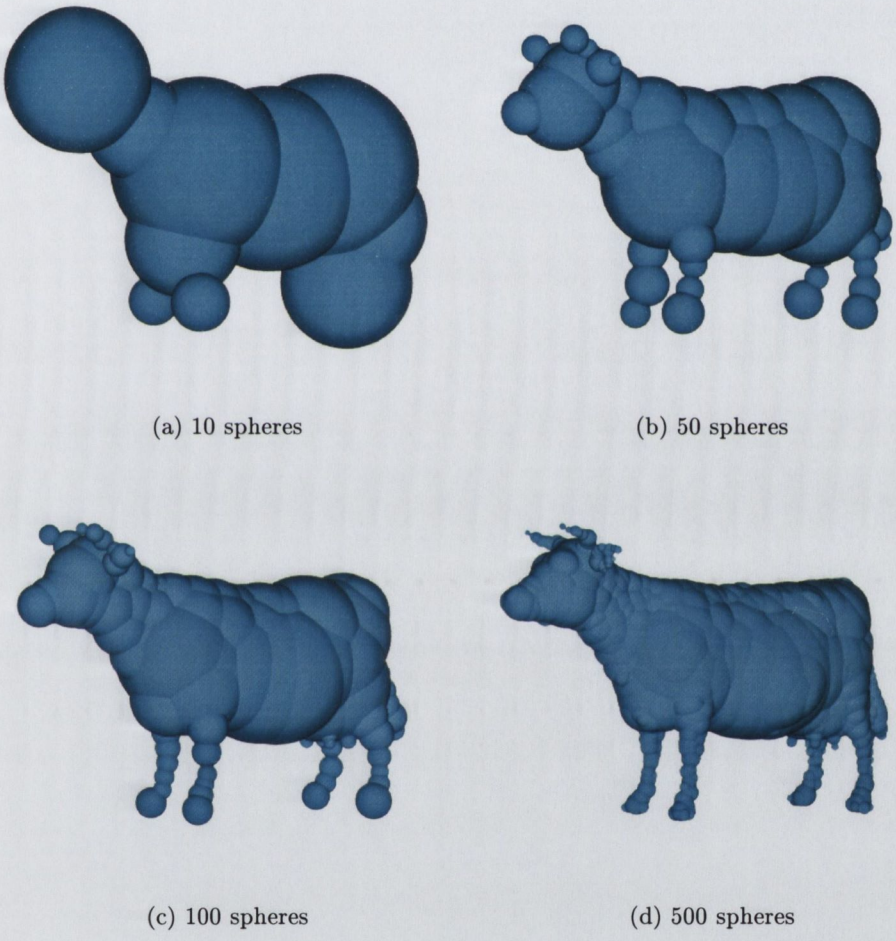
(a) 10 spheres

(b) 50 spheres

(c) 100 spheres

(d) 500 spheres

Figure B.2: Examples of the Bunny approximated with the Expand & Select algorithm.



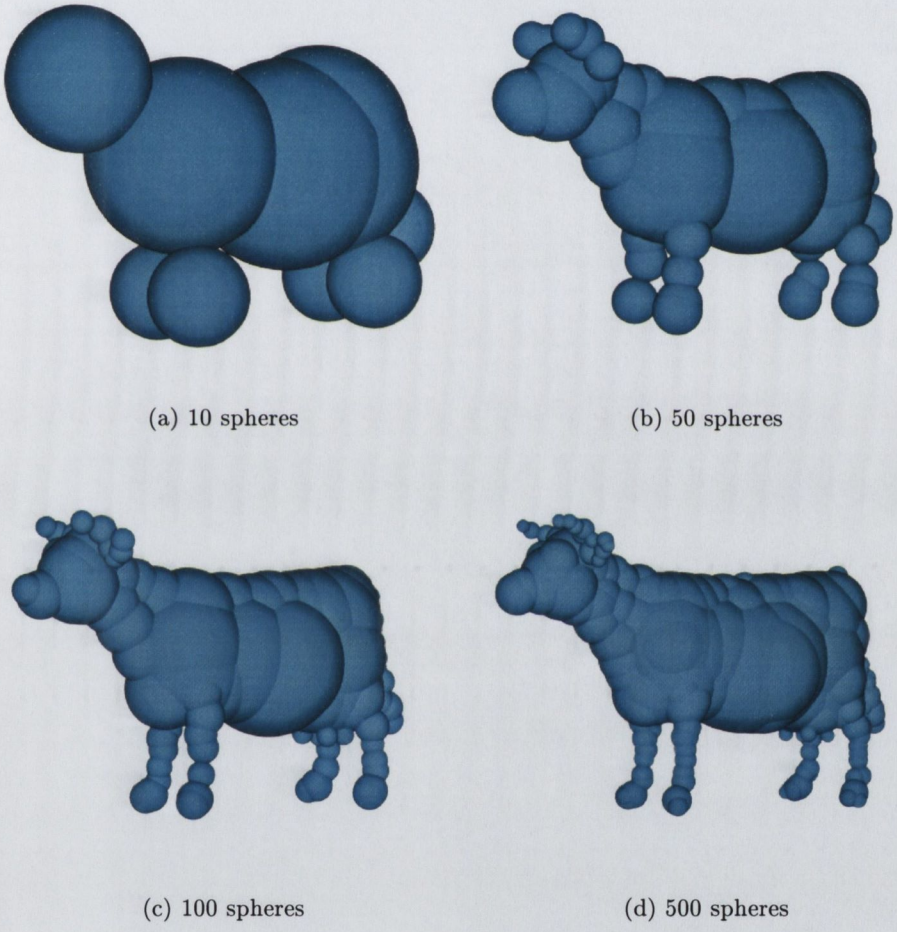
(a) 10 spheres

(b) 50 spheres

(c) 100 spheres

(d) 500 spheres

Figure B.3: Examples of the Cow approximated with the Merge algorithm.



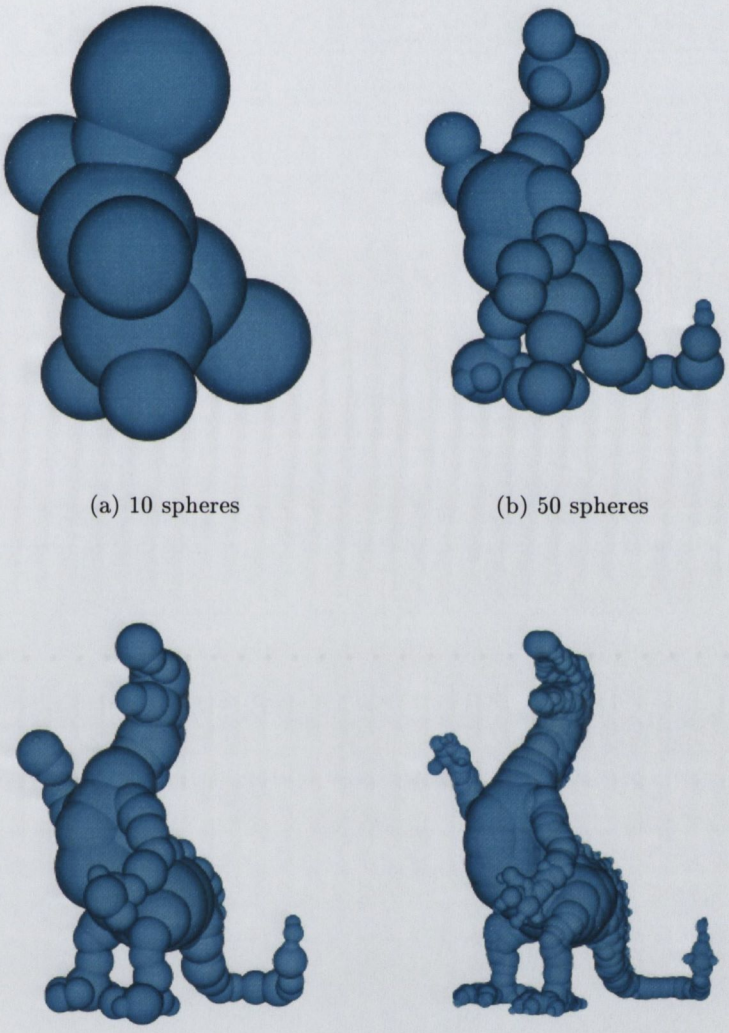
(a) 10 spheres

(b) 50 spheres

(c) 100 spheres

(d) 500 spheres

Figure B.4: Examples of the Cow approximated with the Expand & Select algorithm.



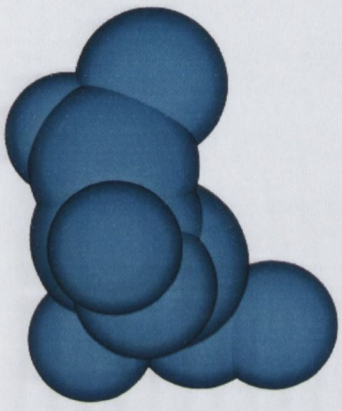
(a) 10 spheres

(b) 50 spheres

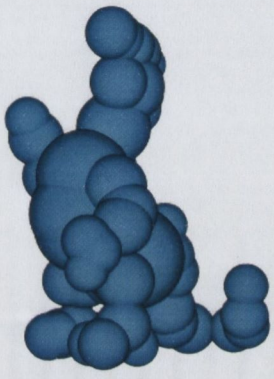
(c) 100 spheres

(d) 500 spheres

Figure B.5: Examples of the Dragon approximated with the Merge algorithm.



(a) 10 spheres

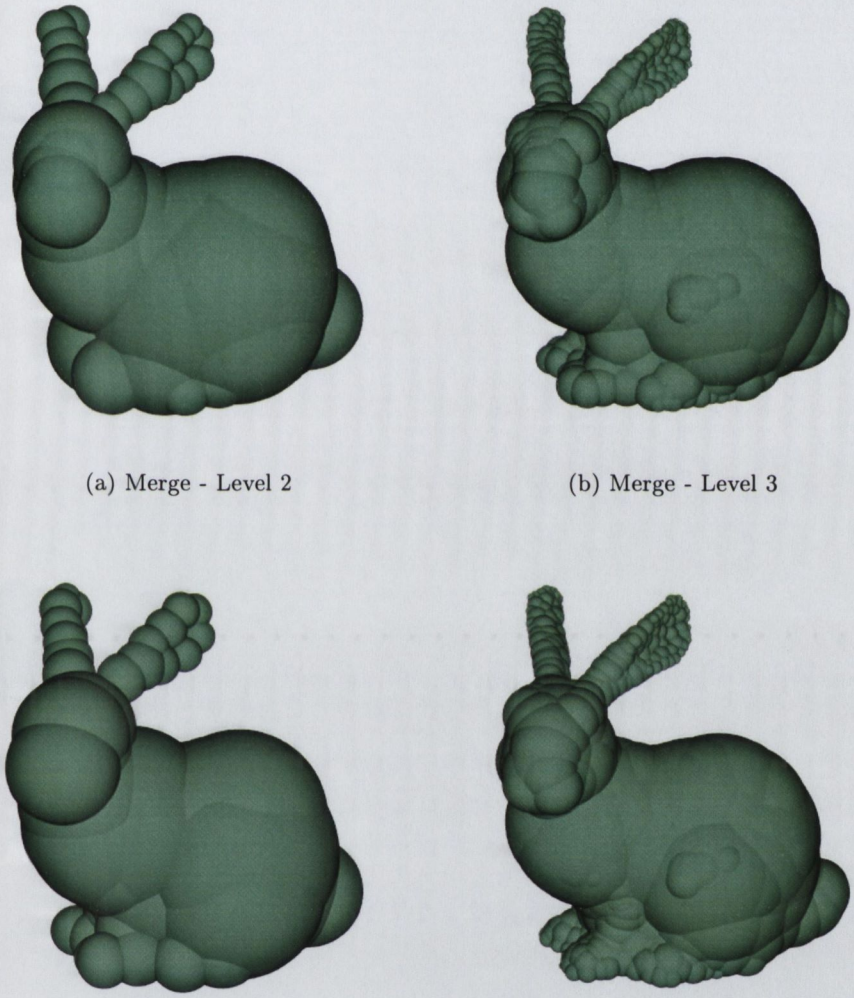


(b) 50 spheres

(c) 100 spheres

(d) 500 spheres

Figure B.6: Examples of the Bunny approximated with the Expand & Select algorithm.



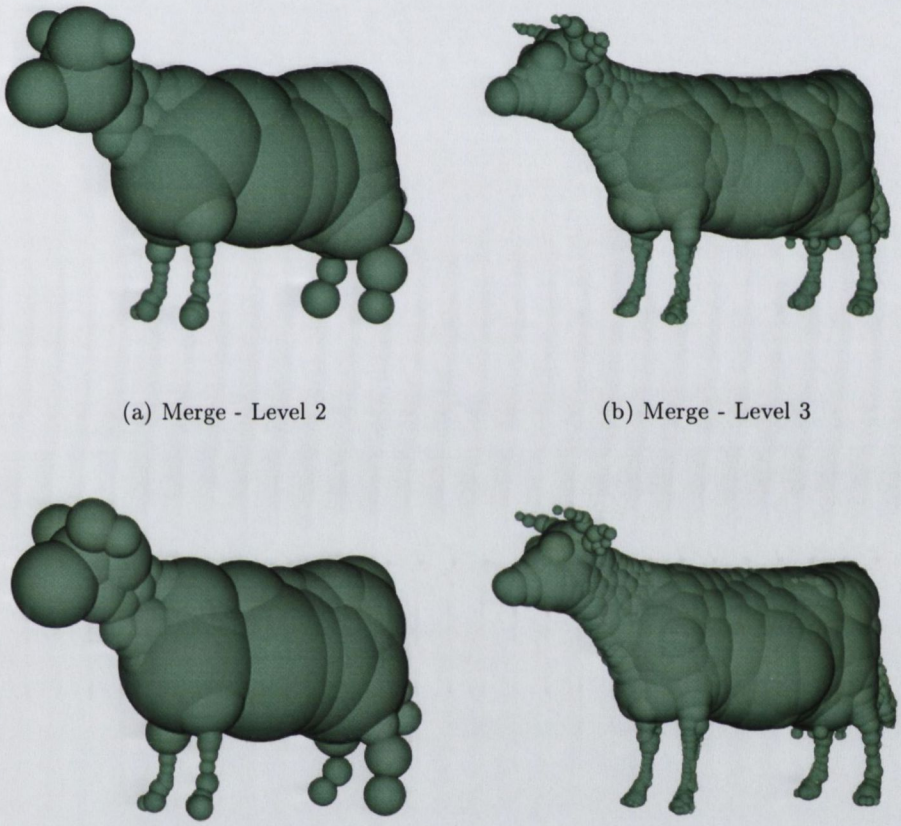
(a) Merge - Level 2

(b) Merge - Level 3

(c) Combined - Level 2

(d) Combined - Level 3

Figure B.7: Sphere-trees constructed for the Bunny.



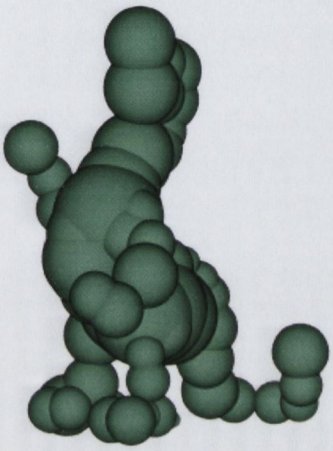
(a) Merge - Level 2

(b) Merge - Level 3

(c) Combined - Level 2

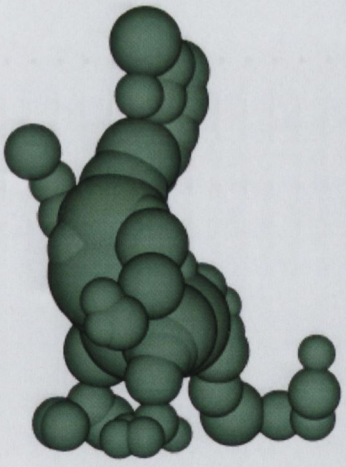
(d) Combined - Level 3

Figure B.8: Sphere-trees constructed for the Cow.



(a) Merge - Level 2

(b) Merge - Level 3



(c) Combined - Level 2

(d) Combined - Level 3

Figure B.9: Sphere-trees constructed for the Dragon.

Bibliography

[1] Frequently asked questions for comp.graphics.algorithms (usenet). http://www.faqs.org/faqs/graphics/algorithms-faq/.

[2] K. Abdel-Malek, H.J Yeh, and N. Maropis. Determining interference between pairs of solids defined constructively in computer animations. *Computers in Engineering*, 14(1):48–58, 1998.

[3] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*, pages 3–10. Elsevier Science Publishers, Amsterdam, North-Holland, 1987.

[4] G. Barequet, B. Chazelle, L.J. Guibas, J.S.B. Mitchel, and A. Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum*, 15(3):387–396, September 1996.

[5] G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Algorithms*, 38:91–109, 2001.

[6] R. Barzel, J.F. Hughes, and D.N. Wood. Plausible motion simulation for computer graphics animation. In *Computer Animation and Simulation'96*, pages 183–197. Springer-Wien, 1996.

[7] H. Blum and R.N. Nagel. Shape description using weighted symmetric axis features. *Pattern Recognition*, 10:167–180, 1978.

[8] U. Borgolte, H. Hoyer, and F. Wrosch. Online collision avoidance for two robots in 3d-space. In *Proc. of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems, Yokohama, Japan*, pages 1919–1926, 1993.

[9] A. Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.

[10] Gareth Bradshaw and Carol O'Sullivan. Extracting cross-sectional profiles from unorganized point clouds. In M.H. Hamza, editor, *Proceedings of Computer Graphics and Imaging (CGIM 2000)*, pages 175–180, November 2000.

[11] Z.L. Cai and J. Dill S. Payandeh. Haptic rendering: Practical modeling and collision detection. In *Proceedings of the ASME Virtual Environment and Teleoperator System Symposium*, pages 81–86, November 1999.

[12] S.A. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Transactions on Robotics and Automation*, 6(3):291–302, 1990.

[13] S.A. Cameron. Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proceedings of the Int. Conf. On Robotics and Automation*, pages 3112–3117, 1997.

[14] Stephen Cameron. Efficient intersection tests for objects defined constructively. *International Journal of Robotics Research*, 8(1):3–25, 1989.

[15] K. Chung and W. Wang. Quick elimination of non-interference polytopes in virtual environments. In *Proceedings of* 3^{rd} *European Workshop on Virtual Environments*, February 1996. Also appeared in the book Virtual Environments'96, Springer-Verlag Wien New York, 1996.

[16] K. Chung and W. Wang. Quick collision detection of polytopes in virtual environments. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology 1996*, July 96.

[17] J.D. Cohen, M.C. Lin, D. Manocha, and M.K. Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scaled environments. In *Proceedings of ACM Int.3D Graphics Conference*, pages 189–196, 1995.

[18] John Dingliana and Carol O'Sullivan. Graceful degradation of collision handling in physically based animation. *Computer Graphics Forum, (Proceedings, Eurographics 2000)*, 19(3):239–247, 2000.

[19] David Eberly. Dynamic collision detection using oriented bounding boxes. Magic Software: www.magic-software.com.

[20] David Eberly. Intersection of objects with linear and angular velocities using oriented bounding boxes. Magic Software: www.magic-software.com.

[21] S. Ehmann and M. Lin. Accurate and fast proximity queries between polyhedra using surface decomposition. *Computer Graphics Forum, (Proceedings, Eurographics 2001)*, 20(3):500–511, 2001.

[22] Carl Erikson and Dinesh Manocha. GAPS: General and automatic polygonal simplification. In *Proceedings of 1999 Symposium on Interactive 3D Graphics*, pages 79–88, 1999.

[23] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, pages 153–174, 1987.

[24] S. J. Fortune. Voronoi diagrams and Delaunay triangulations. *Euclidean Geometry and Computers*, pages 193–233, 1992. World Scientific Publishing Co., D.A. Du, F.K. Hwang, eds.

[25] S.J. Fortune. Voronoi Diagrams and Delaunay Triangulations. *CRC Handbook of Discrete and Computational Geometry*, pages 377–388, 1997.

[26] F. Ganovelli, J. Dingliana, and C. O'Sullivan. BucketTree: Improving collision detection between deformable objects. In *SCCG2000 Spring Conference on Computer Graphics*, pages 156–163, April 2000.

[27] M.A. García. A hierarchical world model represenation supporting heterogeneous multisensory integration. In *International Conference on Advanced Robotics*, pages 461–471, September 1995.

[28] A. García-Alonso, N. Serrano, and J. Flaquer. Solving the collision detection problem. *IEEE Computer Graphics and Applications*, 14:36–43, 1994.

[29] Bernd Gärtner. Fast and robust smallest enclosing balls. In *Proceedings of 7^{th} Annual European Symposium on Algorithms (ESA), Lecture Notes in Computer Science 1643*, pages 325–338. Springer-Verlag, 1999. Available from http://www.inf.ethz.ch/personal/gaertner/miniball.html.

[30] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation*, 4(2):193–203, 1988.

[31] Mark Gill and Albert Zomaya. On the collision detection problems for robot manipulators. *Cybernetics and Systems, An International Journal*, 26:165–188, 1995.

[32] S. Gottschalk. *Collision Queries using Oriented Bounding Boxes.* PhD thesis, Dept. of Computer Science, University of North Carolina, 2000.

[33] S. Gottschalk, M.C. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In *Proceedings of ACM SIGGRAPH '96*, pages 171–180, 1996.

[34] D. Green and D. Hatch. Fast polygon-cube intersection testing. In Alan W. Paeth, editor, *Graphics Gems V*, pages 375–379. Morgan Kaufmann, 1994.

[35] A. Gregory, S. Ehmann, and M.C. Lin. inTouch: Interactive multiresolution modeling and 3D painting with a haptic interrface. In *Proceedings of IEEE Virtual Reality 2000*, pages 45–52, 2000.

[36] A. Gregory, M. Lin, S. Gottschalk, and R. Taylor. H-COLLIDE: A framework for fast and accurate collision detection for haptic interaction. In *IEEE Virtual Reality Conference*, pages 38–45, 1999.

[37] A. Gregory, M.C. Lin, S. Gottschalk, and R. Taylor. Fast and accurate collision detection for haptic interaction using a three degree-of-freedom force-feedback device. *CGTA: Computational Geometry: Theory and Applications*, 15:69–89, 2000.

[38] M. Hariyama, T. Hanyu, and M. Kameyama. A collision detection multiprocessor for intelligent vehicles using a high-density CAM. In *Proc. of the Intelligent Vehicles '94 Symp*, pages 143–148, 1994.

[39] T. He. Fast collision detection using QuOSPO trees. In *Proceedings of the 1999 Symposium on Interactive 3D graphics*, pages 55–62, April 1999.

[40] T. He and A. Kaufman. Collision detection for volumetric objects. In *Proceedings of the 8^{th} IEEE Visualization '97 Conference*, volume 1, pages 27–35, 1997.

[41] Paul S. Heckbert and Micheal Garland. Optimal triangulation and quadric-based surface simplification. *Journal of Computational Geometry: Theory and Applications*, pages 49–65, November 1999.

[42] M. Held, J.T. Klosowski, and J.S.B Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Proceedings of 7th Canadian Conference on Computational Geometry*, pages 205–210, 1995.

[43] L.J. Hettinger and G.E. Riccio. Visually induced motion sickness in virtual environments. *Presence*, 1(3):306–310, 1992.

[44] H. Hoppe. Progressive meshes. In *Proceedings of ACM SIGGRAPH '96*, pages 99–108, 1996.

[45] H. Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of ACM SIGGRAPH '97*, pages 189–198, 1997.

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

[47] P.M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, 1995.

[48] P.M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[49] P.M. Hubbard. *Collision Detection for Interactive Graphics Applications.* PhD thesis, Dept. of Computer Science, Brown University, April 1995.

[50] T. Hudson, M.C. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-COLLIDE: Accelerated collision detection for VMRL. In *Proceedings of VRML 1997*, pages 117–124, 1997.

[51] M. Hughes, C. DiMattia, M.Lin, and D. Manocha. Efficient and accurate interference detection for polynomial deformation and soft object animation. In *Proceesings of Computer Animation '96*, pages 155–166, 1996.

[52] Wolf-D Ihlenfeldt. Virtual reality in chemistry. *Journal of Molecular Modeling*, 3:386–402, September 1997.

[53] H. Inagaki, K. Sugihara, and N.Sugie. Numerically robust incremental algorithm for constructing 3D Voronoi diagrams. In *Proceedings of the 4th Canadian Conference on Computational Geometry*, pages 334–339, 1992.

[54] D.R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[55] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: A survey. *Computers and Graphics*, 25(2):269–285, 2000.

[56] Y. Kitamura, H. Takemura, N. Ahuja, and F. Kishino. Efficient collision detection among objects in arbitrary motion using multiple shape representations. In *Proceedings 12th IAPR Int. Conf. On Pattern Recognition*, volume 1, pages 390–396, 1994.

[57] J.T. Klosowski. *Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments.* PhD thesis, State University of New York at Stony Brook, May 1998.

[58] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[59] Y. Kono, M. Seto, K. Nishimatsu, H. Fukumori, and Y. Muraoka. Parallel mesh generation for FEM - parallel construction of Voronoi diagram. *IPSJ SIGNotes - High Performance Computing*, 60, 1995.

[60] S. Krishnan, M. Gopi, M. Lin, D. Manocha, and A. Pattekar. Rapid and accurate contact determination between spline models using ShellTrees. In *Proceedings of Eurographics '98*, volume 17(3), pages 315–326, 1998.

[61] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shells: A higher order bounding volume for fast proximity queries. In *Proceedings of the 1998 Workshop on the Algorithmic Foundations of Robotics*, pages 122–136, March 1998.

[62] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Dept. of Computer Science, University of North Carolina, 1999.

[63] E. Larsen, S. Gottschalk, M.C. Lin, and D. Manocha. Fast distance queries using rectangular swept sphere volumes. In *Proceedings of IEEE International Conference on Robotics and Automation 2000*, 2000.

[64] E. Levey, C. Peters, and C. O'Sullivan. New metrics for evaluation of collision detection techniques. In *Proceedings of The* 8^{th} *International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media'2000.*, 2000.

[65] M. Lin. *Efficient Collision Detection for Animation and Robotics.* PhD thesis, University of California, Berkeley, 1993.

[66] M. Lin and D. Manocha. Efficient contact determination between geometric models. *Internation Journal of Computational Geometry and Applications*, 7(1):123–151, 1997.

[67] M.C. Lin and J.F. Canny. Efficient algorithms for incremental distance computation. In *Proc. IEEE Conference on Robotics and Automation*, pages 1008–1014, 1991.

[68] M.C. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proceedings of IMA Conference on Mathematics of Surfaces*, pages 33–52, 1998.

[69] M.C. Lin, A. Gregory, S. Ehmann, S. Gottschalk, and R. Taylor. Contact determination for real-time haptic interaction in 3D modeling, editing and painting. In *Proceedings of 1999 Workshop for PHANTOM User Group*, pages 58–61, 1999.

[70] M.C. Lin, D. Manocha, J. Cohen, and S. Gottschalk. Collision Detection: Algorithms and Applications. In Jean-Paul Laumond, M. Overmars, and A.K. Peters (eds.), editors, *Algorithms for Robotics Motion and Manipulation*, pages 129–142. 1997.

[71] Jean-Christophe Lombardo, Marie-Paule Cani, and Fabrice Neyret. Real-time collision detection for virtual surgery. In *Computer Animation'99*, May 1999.

[72] V. Milenkovic. Robust construction of the Voronoi diagram of a polyhedron. In A. Lubiw and J.Urrutia, editors, *Proceedings of the* 5^{th} *Canadian Conference on Computational Geometry*, pages 473–478, August 1993.

[73] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, 1998.

[74] Brian Mirtich. Timewarp rigid body simulation. In *Proceedings of ACM SIGGRAPH 2000*, July 2000.

[75] Tomas Möller and Eric Haines. Ray/triangle intersection. In *Real-time Rendering*, pages 303–305. A.K. Peters, 1999.

[76] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, 1988.

[77] Allison M. Okamura. Literature survey of haptic rendering, collision detection and object modelling. Technical report, Johns Hopkins University, Department of Engineering, Baltimore, August 1998.

[78] C. O'Sullivan. *Perceptually-Adaptive Collision Detection for Real-time Animation.* PhD thesis, Trinity College Dublin, Ireland, June 1999.

[79] C. O'Sullivan and J. Dingliana. Realtime collision detection and response using sphere-trees. In *Proceedings of the Spring Conference on Computer Graphics*, pages 83–92, 1999.

[80] C. O'Sullivan and J. Dingliana. Collisions and perception. *ACM Transactions on Graphics*, 20(3):151–168, 2001.

[81] C. O'Sullivan, R. Radach, and S. Collins. A model of collision perception for real-time animation. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation and Simulation'99*, pages 67–76. Springer-Wien, 1999.

[82] I.J. Palmer and R.L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14(2):105–116, 1995.

[83] M.K. Ponamgi, D. Manocha, and M.C. Lin. Incremental algorithms for collision detection between polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):51–64, 1997.

[84] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing, Downhill Simplex Method in Multidimensions*, chapter 10.4, pages 408–412. Cambridge University Press, second edition, 1992.

[85] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing, Recursive Stratified Sampling*, chapter 7.8, pages 323–327. Cambridge University Press, second edition, 1992.

[86] S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[87] R. Rabbitz. Fast collision detection of moving convex polyhedra. In P.S. Heckbert, editor, *Graphics Gems IV*, pages 83–109. Academic Press, Cambridge, MA, 1994.

[88] Martin Reddy. A survey of level of detail support in current virtual reality solutions. *Virtual Reality Research, Development and Application*, 1(2):85–88, 1995.

[89] J. Ritter. An efficient bounding sphere. In Andrew S. Glassner, editor, *Graphics Gems*, pages 301–303. Academic Press, 1990.

[90] David F. Rogers. *Procedural Elements for Computer Graphics*, pages 196–207. McGraw-Hill Book Company, New York, 1985.

[91] J. Rolhf and J. Helman. IRIS Performer: A high performance multiprocessing toolkit for real-time 3D graphics. In *Proceedings of ACM SIGGRAPH '94*, pages 381–393, 1994.

[92] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In Kurt Akeley, editor, *Proceedings of ACM SIGGRAPH 2000*, pages 343–352. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[93] S. Rusinkiewicz and M. Levoy. Streaming QSplat: A viewer for networked visualization of large, dense models. *2001 Symposium on Interactive 3D Graphics*, 2001.

[94] H. Sammet and R. Webber. Hierarchical data structures and algorithms for computer graphics. *IEEE Comp. Graphics and Applications*, 8(3):48–68, 1988.

[95] C.A. Shaffer and G.M. Herb. A real-time robot arm collision avoidance system. *IEEE Transactions on Robotics and Automation*, 8(2):149–160, 1992.

[96] I.P.W. Sillitoe, A. Batersby, and J. Edwards. A parallel architecture for efficient clash detection. *Progress in Transputer and Occam Research*, pages 32–39, 1994.

[97] A. Smith, Y. Kitamura, and F. Kishino. Efficient algorithms for octree motion. In *IAPR Workshop on Machine Vision Applications*, pages 172–177, 1994.

[98] A. Smith, Y. Kitamura, H. Takemura, and F. Kishino. A simple and efficient method for accurate collision among deformable polyhedral objects in arbitrary motion. In *Proceedings of the Virtual Reality Annual International Symposium*, pages 136–145. IEEE, March 1995.

[99] J.M. Snyder. Interval analysis for computer graphics. In *Proceedings of ACM SIGGRAPH'93*, pages 121–129, 1992.

[100] J.M. Snyder, A.R. Woodbury, K. Fleischer, B. Currin, and A. Barr. Interval methods for multi-point collisions between time-dependent curved surfaces. In *Proceedings of SIGGRAPH'93*, pages 321–334, 1993.

[101] C. Tseng and C. Wu. Collision detection for multiple robot manipulators by using orthogonal neural networks. *Journal of Robotic Systems*, 12:479–490, 1995.

[102] G. Turk. Interactive collision detection for molecular graphics. Master's thesis, Dept. of Computer Science, The University of North Carolina, 1989.

[103] G. Turk. Generating random points in triangles. In Andrew S. Glassner, editor, *Graphics Gems*, pages 24–28. Academic Press, 1990.

[104] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–13, 1997.

[105] G. van den Bergen. A fast and robust GJK implementation for collision detection between convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999.

[106] P. Volino and M. Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. In *Eurographics'94, Computer Graphics Forum*, volume 13, pages 155–166, 1994.

[107] B. Von-Herzen, A.H. Barr, and H.R. Zatz. Geometric collisions for time-dependent parametric surfaces. In *Proceedings of SIGGRAPH'99*, pages 39–48, 1990.

[108] Emo Weltz. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, pages 359–370. 1991.

[109] Dave White. *Smallest Enclosing Ball of Points/Balls.* http://vision.ucsd.edu/ dwhite/ball.html.

[110] Andrew J. Willmott. *Hierarchical Radiosity with Multiresolution Meshes.* PhD thesis, School of Computer Science, Carnegie Mellon University, 2000.

[111] A. Wilson, E. Larsen, D.Manocha, and M.C. Lin. IMMPACT: A system for interactive proximity queries on massive models. Technical Report TR98-031, Dept. of Computer Science, University of North Carolina, 1998.

[112] Julie C. Xia, Jihad El-Sana, and Amitabh Varshney. Adaptive real-time level-of-detail based rendering of polygon meshes. *IEEE Transactions on Visualisation and Computer Graphics*, 3(2):171–183, June 1997.

[113] J. Xiao and L. Zhang. Towards obtaining all possible contacts - growing a polyhedron by its location uncertainty. In *Proceedings 1994 IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, pages 1788–1795, September 1994.

[114] J.H. Youn and K. Wohn. Realtime collision detection for virtual reality applications. In *Proceedings IEEE Virtual Reality Annual Int. Sym.*, pages 18–22, 1993.

[115] M. Zeiller. Collision detection for objects modeled by CSG. *Visualization and Intelligent Design in Engineering and Architecture*, pages 165–180, 1993.

[116] M. Zeiller. *Collision Detection for Complex Objects in Computer Animation.* PhD thesis, Vienna University of Technology, 1994.

[117] M. Zeiller, W. Purgathofer, and M. Gervautz. Efficient collision detection for general CSG objects. In *Proceedings of EUROGRAPHICS* 6^{th} *Workshop on Computer Animation and Simulation*, 1995.

[118] D. Zhang and M.M.F Yuen. Collision detection for clothed human animation. In *Proceedings of the* 8^{th} *Pacific Conference on Computer Graphics and Applications*, pages 228–237, 2000.