Mälardalen University Press Dissertations
No.71

Adaptive Bounding Volume Hierarchies for Efficient Collision Queries

Thomas Larsson

January 2009

School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden

ISSN 1651-4238
ISBN 978-91-86135-18-8
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

Abstract

The need for efficient interference detection frequently arises in computer graphics, robotics, virtual prototyping, surgery simulation, computer games, and visualization. To prevent bodies passing directly through each other, the simulation system must be able to track touching or intersecting geometric primitives. In interactive simulations, in which millions of geometric primitives may be involved, highly efficient collision detection algorithms are necessary. For these reasons, new adaptive collision detection algorithms for rigid and different types of deformable polygon meshes are proposed in this thesis. The solutions are based on adaptive bounding volume hierarchies.

For deformable body simulation, different refit and reconstruction schemes to efficiently update the hierarchies as the models deform are presented. These methods permit the models to change their entire shape at every time step of the simulation. The types of deformable models considered are (i) polygon meshes that are deformed by arbitrary vertex repositioning, but with the mesh topology preserved, (ii) models deformed by linear morphing of a fixed number of reference meshes, and (iii) models undergoing completely unstructured relative motion among the geometric primitives. For rigid body simulation, a novel type of bounding volume, the slab cut ball, is introduced, which improves the culling efficiency of the data structure significantly at a low storage cost. Furthermore, a solution for even tighter fitting heterogeneous hierarchies is outlined, including novel intersection tests between spheres and boxes as well as ellipsoids and boxes. The results from the practical experiments indicate that significant speedups can be achieved by using these new methods for collision queries as well as for ray shooting in complex deforming scenes.

To my family with love!

Preface

This is a collection-of-paper thesis, which means that the main results have already been presented in published papers. Therefore, the thesis is divided into two parts. It starts with a so-called "coat" in Part I (Chapters 1–5), which gives a more thorough background and motivation to the work than what was possible in the individual papers. It also shows how the papers are connected and related to each other, and what the main contributions of this work are as a whole. Then Part II (Chapters 6–12) follows with the published papers reprinted. Although it would suffice to refer to these papers, they are reprinted in the second part of the thesis as a convenience for the reader. A list of the included papers is given in Table 4.1 on page 38.

Now at the conclusion of this work, I would like to thank my advisor, computer graphics expert, and paper co-author Professor Tomas Akenine-Möller for all his support and guidance. I would also like to thank my principal advisor Professor Björn Lisper for all the support he has given me. Furthermore, I really appreciate the fruitful cooperation I have had with Rikard Lindell when it comes to sharing the program responsibility for our bachelor programs in computer science and game development with me, which in particular helped me to find the necessary time to finish the last part of this thesis. And of course, my thanks go to my other colleagues here at the department. All have helped by contributing to the positive and creative research environment which we share daily. Thank you all!

More than anything else, I am also indebted to my wonderful wife Paulina and our two beloved sons, André and William, for always encouraging me, and for the inspiration you provide, and for sharing with me the more important things in life. Without your love and support I would not have finished this work. Finally, I would like to thank my

parents for always being there, and for their support during my undergraduate studies.

Thomas Larsson
Västerås, January 18, 2009

Contents

I

Thesis

Chapter 1

Introduction

This thesis is mainly concerned with how geometric collision queries can be realized efficiently in real-time computer graphics and visualization applications. The main goal of this work is to present novel practical data structures and algorithms applicable under varying conditions in interactive simulations. More precisely, adaptive bounding volume hierarchies are presented, together with geometrical algorithms which accelerate important operations such as collision detection (CD) for complex and dynamic scenes. The research has led to seven papers, which are presented in Chapter 4, and they are also included in their whole in Chapters 6–12.

To start with, however, a short introduction is given to computer graphics in general and to interactive visual simulation. Next, spatial data structures are introduced in brevity. Then the collision detection problem is presented, which is the main problem addressed in this work. Finally, an outline of the rest of the thesis concludes this chapter.

1.1 Computer Graphics

In 1960, designer William Fetter of Boeing Aircraft Company devised the term "computer graphics" to describe the design methods they developed to produce ergonomic descriptions for aircraft design. Much has happened since this early start of computer generated images. Nowadays, computer graphics is an indispensable tool in a broad range of

application areas such as printing, design and manufacturing, interactive simulations, scientific visualization, education, and entertainment. The perhaps most widely known application areas for computer graphics are in TV, moving picture production, and computer games, in which images generated by computer graphics play a critical role.

Computer graphics have also grown to become an important academic discipline. The Computing Curricula 2001 [1] gives the following definition of the computer graphics field:

> Computer graphics is the art and science of communicating information using images that are generated and presented through computation. This requires (a) the design and construction of models that represent information in ways that support the creation and viewing of images, (b) the design of devices and techniques through which the person may interact with the model or the view, (c) the creation of techniques for rendering the model, and (d) the design of ways the images may be preserved. The goal of computer graphics is to engage the person's visual centers alongside other cognitive centers in understanding.

As can be seen from this definition, communication through computer graphics imagery heavily relies on model and image representation, generation, and interaction. Accordingly, the main research areas in computer graphics are called modelling, rendering, and animation. Briefly, modelling deals with the problem of how to represent objects and build these representations, rendering is about generating synthetic images from model and scene descriptions, often with the goal of producing photo-realistic images, and animation is about specifying and controlling how objects move, change their shape, and interact with each other. Today's interactive computer graphics applications rely heavily on the complementary research from all these areas.

1.2 Interactive Visual Simulation

When an animation is driven and produced in real time by simulation, user interaction, and rendering, we can experience a virtual reality, or an interactive visual simulation. Many important applications can be created based on this type of simulation systems. For example, in flight

simulation, virtual worlds are created to mimic the real world, so that novice pilots can be trained for future flight operations under safe conditions. Virtual surgery makes it possible for surgeons to practise advanced operations under realistic, but safe, circumstances. Architectural walk-through applications help architects to design buildings, and make it possible for potential customers to experience buildings before they have been built. In interactive storytelling, fantasy worlds can be explored and experienced through computer graphics imagery.

How the simulation is driven forward is, in general, application-specific. For example, it can be done by applying physical laws of motion, or by applying some kind of procedural simulation rules. In interactive graphics systems, the user is allowed to control and dynamically change the state of the simulated scene, for example by using different kinds of input devices, such as mice, data gloves, and force feedback devices.

To make such applications possible, real-time rendering and simulation systems are required [2]. In this context, the term real-time often means that images of the scene can be generated or rendered at 30–90 frames per second. Nowadays, when scenes may be composed of millions of geometric primitives and that high definition image resolutions are mandatory, it is easy to see why these systems must be powerful. Most often the simulated scenarios tend to become too complex if realistic models are to be used. As an example, imagine a visual traffic simulation application using a scene that includes detailed geometric models of all the buildings, vehicles, and pedestrians in a big city. Clearly, sophisticated techniques would be needed to simplify the simulation sufficiently to make it computationally possible, while ensuring that by running the simulations we get valuable feedback and results. To make complex simulation scenarios possible, efficient data structures, algorithms and other speed-up techniques are required.

1.3 Spatial Data Structures

Since algorithmic improvements can lead to asymptotically faster execution times, they are essential when dealing with large complex scenes. A fundamental technique to accelerate applications in computer graphics, and in other fields as well, such as computational geometry, geometrical information systems (GIS), and robotics, is to use spatial data structures [3]. This type of data structure is used to represent scenes and

geometric data in an n-dimensional space. The data structures serve as a database that supports efficient search algorithms to answer different types of queries. For example, in rendering and animation, visibility and collision queries are common to answer questions such as: Which models, or parts of the models, are inside the observer's field of view? Given a directed ray or a line segment, which geometric primitive is hit first? Is there any intersection between two given geometric models, and if so, which parts of the models are in contact? Given n moving models, at which moment in time will the first collision occur?

Many spatial data structures are based on subdividing the space efficiently into hierarchical levels of non-overlapping convex regions or subvolumes. Examples of such space subdividing data structures are BSP trees [4, 5, 6, 7], kd-trees [8, 9], quadtrees [10, 11, 12, 13], octrees [14], and multi-level grids [15, 16]. The BSP tree and the kd-tree data structures are quite similar. At each level, both of them divide the space into two half-spaces using a single split plane. One of the main differences between them is that in a kd-tree the splitting planes are always perpendicular to one of the principal axes of the coordinate system, whereas in a BSP tree, the splitting planes can be arbitrarily positioned. Therefore, the kd-tree can be seen as a special case of the BSP tree.

When it comes to the quadtree and octree, on the other hand, each region is divided recursively into equally sized sub-regions using two and three split planes, respectively. This means that a split gives rise to four new sub-regions in the quadtree case, and eight new sub-regions in the octree case. In contrast to BSP trees and kd-trees, which are often used for subdividing spaces with any number of dimensions, the quadtree is often used to subdivide in 2D and the octree is often used to subdivide in 3D.

Multi-level grids, or nested grids, have been presented as a more efficient alternative compared to quadtrees and octrees for some applications, mainly ray tracing. Usually, the top-level grid is a box with $O(n)$ equally sized cells representing the relative locations of n geometric primitives. Each cell contains a reference to the m geometric primitives located wholly or partly inside it. Furthermore, a cell can potentially hold a reference to a sub-grid for refined representation when m is too large. The nestling of grids is then repeated recursively as needed. Normally, however, the number of levels used is quite small [16]. In some cases, using only a single level may be the best choice, and the data structure is then often called a uniform grid [17, 18, 19, 20].

Another type of spatial data structure, the bounding volume hierarchy (BVH), focuses on representing the space surrounding geometrical objects efficiently, rather than on tiling the space itself efficiently. This data structure encloses the geometrical object it covers at several increasingly more detailed levels. For more information on the bounding volume hierarchy, see Chapter 2.

Most of these mentioned data structures are hierarchical, which provides the means for logarithmic query times in many cases. Since the construction of the selected data structure is usually quite expensive it is preferably done as a precomputation or during application initialization. Necessary changes during run-time are then often made incrementally to amortize the update cost, and thereby avoiding severe performance bottlenecks that otherwise may occur.

1.4 Problem Description

As discussed above, specialized data structures and algorithms are needed to be able to handle efficiently the complexity of interactive visual simulations. An always recurring problem in such dynamic simulations is collision detection, which is essential to avoid objects from passing straight through each other in the virtual environment. Since collision detection is computationally very challenging and often reported to be a major bottleneck in physical simulations, this is the problem in focus in this dissertation.

Given a scene with n moving objects or bodies, the number of unique body pairs that can be selected is

$$N_b = \binom{n}{2} = \frac{n(n-1)}{2}. \tag{1.1}$$

Thus, a naive collision detector can check the current collision status in a scene by considering all these body pairs. Such a method suffers from the all-pair weakness, and it is far too slow for most interactive visual simulations. Even if initially only a fast constant time operation, such as a sphere-sphere overlap test, is executed per body pair just to find out that there is not a single collision, this would still take $O(n^2)$ time. Therefore, the goal of any collision detection algorithm is to first reduce the number of object pairs that must be considered using an efficient heuristic. Such an initial phase is often referred to as the *broad phase* of the collision detection process [21, 22].

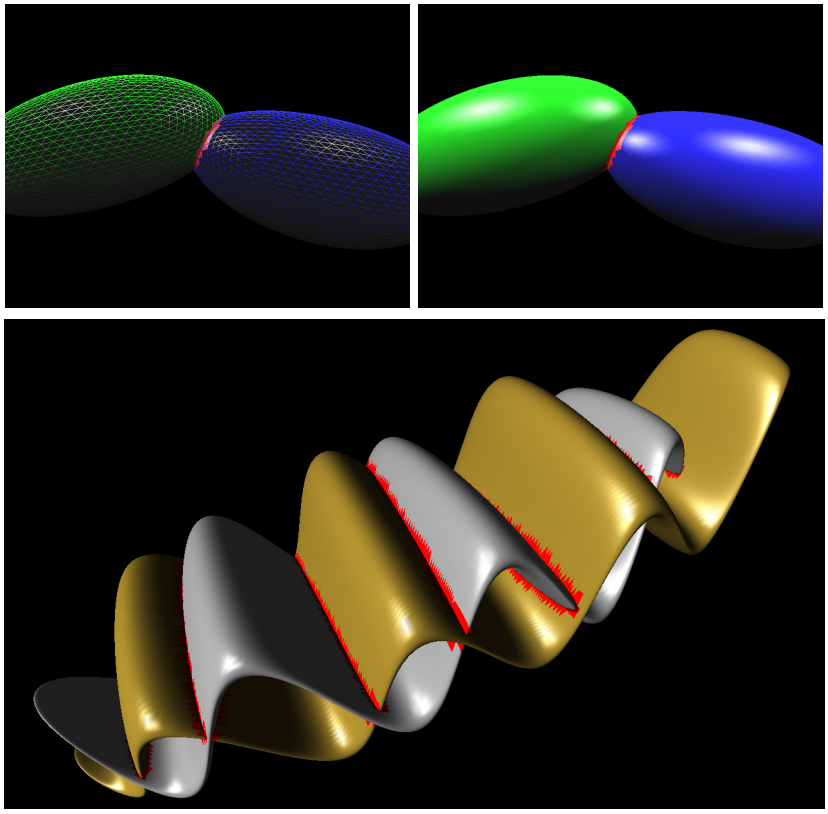

Figure 1.1: Two meshes with 5,120 triangles per mesh shown wireframe (top left) and Gouraud shaded (top right). Penetrating triangle pairs are shown in red. There are 168 intersecting triangle pairs. Naive testing results in 26,214,400 triangle-triangle overlap tests. At the bottom image, two models with 81,920 triangles per mesh are shown. In this case, there are 13,768 intersecting primitive pairs. Naive testing results in 6,710,886,400 triangle-triangle intersection tests. In this case, executing all these overlap tests sequentially took approximately 17 minutes on a laptop computer with an Intel CPU T2600 2.16 GHz.

Similarly, a slightly different variant of the all-pair weakness must be avoided when more detailed collision testing is necessary between the remaining body pairs, after the initial pruning in the broad phase. Suppose that two bodies in such a body pair are composed of m_i and m_j geometric primitives, respectively. Checking each geometric primitive of the first body against every primitive of the other body results in $m_i \times m_j$ intersection tests, which again is far too slow for all but the simplest bodies, as illustrated in Figure 1.1. Therefore, once again the goal must be to reduce the number of primitive pairs tested by using appropriate data structures and algorithms. This part of the collision detection process, where detailed tests are made between all pairs of objects that were not pruned by the broad phase, is often called the *narrow phase* of the collision detection process [21]. In total for n models, using a naive CD method in both the broad and narrow phase, would lead to a time complexity of $O(n^2m^2)$, assuming that each model has m geometric primitives.

Besides the number of rigid bodies in motion and the geometric primitive count in the scene, there are other aspects influencing the complexity of the problem. For example, certain types of complex contact scenarios arising in virtual assembly applications can trigger a worst-case behaviour of otherwise efficient hierarchical CD approaches with severe performance implications. In such cases, a tight-fitting hierarchical data structure, adaptive to the curvature of models, is needed to avoid severe and unnecessary performance bottlenecks [23]. Also, a potential cause of inaccuracies, often referred to as the tunnelling problem, is the employed time-stepping mechanism. Clearly, in a pure discrete time-stepping simulation, there is a chance that the moving bodies pass straight through each other between any two time steps t_i and t_{i+1}. In some applications, a strategy to avoid inaccurate motion of the bodies is needed, such as back-tracking in simulation time, event-based time-stepping, or four dimensional swept-volume intersection testing [24, 25, 26].

An area closely related to collision detection is proximity detection, which is a more general problem which for example is relevant in applications focusing primarily on collision avoidance [27, 28, 29]. Given a minimum allowed distance δ, a proximity query is performed to detect any two objects located too closely to each other. Collision detection can be considered a special case of proximity detection with $\delta = 0$. By using appropriately thickened versions of the involved objects in the BVHs, however, a CD query can be turned into a proximity query [30].

The simulation of soft or elastic bodies constitutes another type of challenge. Imagine a scene inhabited by complex deforming meshes, modelled by hundreds of thousands of geometric primitives. Obviously, acceleration data structures are needed to handle the geometric complexity, but since the bodies undergo deformations, the data structures must be rebuilt or updated in proper ways to remain useful. Therefore, adaptive hierarchical data structures and algorithms are needed also in this more dynamic case.

There are many types of interactive simulation systems that include dynamically deforming scenes, for example, in physical simulation, computational surgery, molecular modelling, animation, cloth simulation, and computer games. Deformable models whose contact behaviours need to be simulated include articulated characters with clothing, soft tissues and organs, biological structures, molecules, and other soft or elastic materials.

Finally, collision or intersection queries are not only important for the detection and resolution of the collisions of moving bodies in graphics simulations. In ray tracing, a huge number of ray-scene intersections must be determined as part of the rendering process. Therefore, ray tracing may be regarded as another type of collision detection problem, and similar types of data structures and algorithms are needed. In particular, interactive ray tracing of complex dynamic scenes is very challenging and requires highly efficient data structures and aggressive code optimizations.

1.5 Outline of Thesis

The rest of this thesis is organized as follows. Chapter 2 gives an introduction to bounding volume hierarchies and their usage, since this is the main data structure utilized in the proposed solutions. Then, in Chapter 3, collision queries are discussed, which is the main algorithmic problem studied in this thesis. In Chapter 4, the proposed algorithms for hierarchical collision detection of rigid and deforming meshes are described, which includes short summaries of the papers this thesis is based on and brief descriptions of the main contributions of each. For a more detailed treatment, please consult the original papers, referred to as papers **A**–**G** (see Table 4.1). These papers are also reprinted in Chapters 6–12 in this thesis as a convenience for the reader. Finally, Chapter 5

presents the conclusions as well as some interesting directions for future work.

Chapter 2

Bounding Volume Hierarchies

A bounding volume hierarchy, or a bounding volume tree, is an acceleration data structure for speeding up various types of geometric queries. The BVH provides a complete coverage (enclosure) of a set of geometric primitives at several levels-of-detail (LODs). By using a BVH, it is often possible to reduce the running time of a geometric query from, e.g., $O(n)$ to $O(\log n)$. BVHs have found extensive usage to speed up collision detection, motion planning, view-frustum culling, picking, ray tracing, and other spatial operations. As an example of a BVH, consider the visualization of some of the levels in a BVH of spheres on a teapot model given in Figure 2.1.

2.1 Definition

Bounding volume hierarchies have much in common with classical tree data structures such as binary search trees [31], interval trees [13], and in particular R-trees and their variants [32, 33, 34]. A bounding volume hierarchy is a tree data structure on a geometric model M that stores all the geometric primitives of M in the leaf nodes. Each node in the tree stores a volume that encloses all the primitives located below it, i.e., in its subtree. In this way, the root node stores a bounding volume (BV) enclosing all the primitives or the entire model. And the children

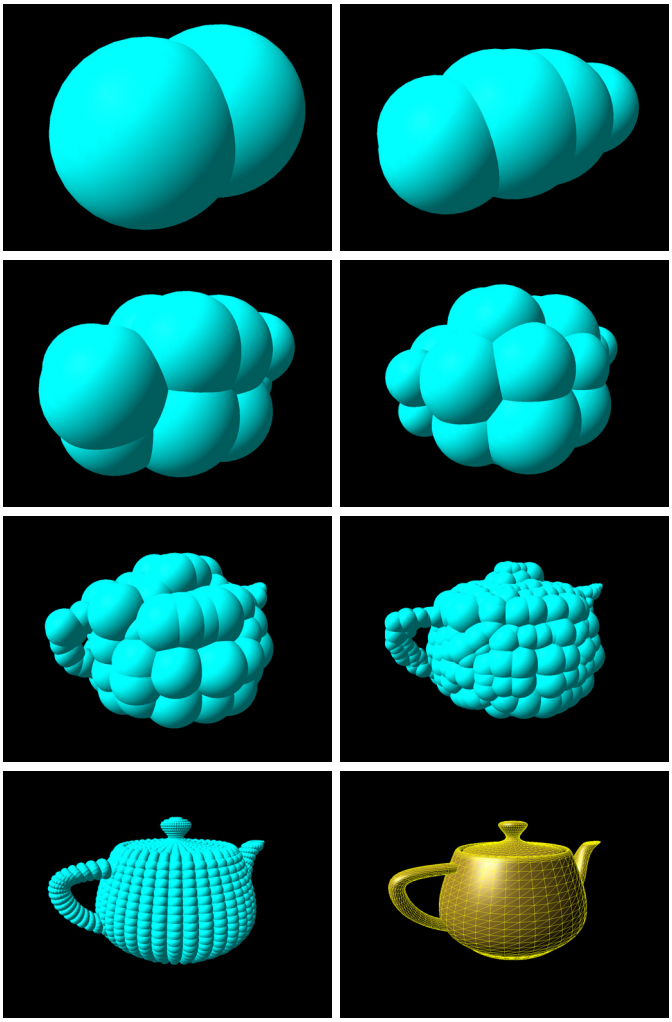

Figure 2.1: A visualization of a BVH of spheres on a teapot model. The top three rows show the levels 1, 3, 4, 5, 7, and 9. The bottom row shows the spheres in the leaf nodes (left) and the actual teapot mesh (right), which has 6,400 triangles.

nodes, store BVs enclosing various subsets of the primitives or parts of the model in a wrapped hierarchical fashion.

A BVH has degree k when each internal node, or non-leaf node, has exactly k children. Common values of k are 2, 3, 4, and 8, giving rise to binary, tertiary, quaternary, and octonary trees, respectively. However, if the number of children nodes located directly under a parent varies throughout the tree, it also makes sense to talk about the degree k_i of single nodes in the tree. In this case, the node with the maximum number of children in the tree determines the degree of the whole tree, i.e., $k = \max k_i$.

The levels of the trees are numbered starting with the root at level zero. Consequently, the height of a BVH on a model with n primitives is at least

$$h = \lfloor \log_k n \rfloor. \tag{2.1}$$

To see why, consider a complete binary tree data structure, i.e., a tree where all leaves are located at the same height, with n nodes (both internal and leaf nodes), m leaves, and height h. Then the number of nodes n can be written out as a sum of the nodes in all the tree levels, which gives

$$n = 2^0 + 2^1 + 2^2 + \ldots + 2^h = 2^{h+1} - 1 = 2m - 1. \tag{2.2}$$

From this formulation, it is clear that the number of levels below the root node, i.e., the height of the tree h, is exactly

$$h = \log_2 m = \lfloor \log_2 n \rfloor = \log_2(n+1) - 1. \tag{2.3}$$

In general, a complete tree with degree $k = k_i$ has

$$n = k^0 + k^1 + k^2 + \ldots + k^h = \frac{k^{h+1} - 1}{k - 1} = \frac{km - 1}{k - 1} \tag{2.4}$$

nodes. Thus, the height of such trees is

$$h = \log_k m = \lfloor \log_k n \rfloor. \tag{2.5}$$

The logarithmic height property of complete trees discussed above also holds true for all balanced trees, since in this case, the greatest allowed difference in depth of two leaf nodes is one. For arbitrary tree structures, however, which possibly contain one or a few long chains of

nodes, the height may degenerate to linear in the number of nodes n. To avoid this, most construction algorithms aim at building balanced, or reasonably balanced hierarchies (see Section 2.3).

The memory requirement of a BVH is linearly proportional to the number of leaf nodes. Given a complete tree data structure on a model with m geometric primitives, i.e. m leaves, and $k = k_i$, the total number of internal nodes, l, in the tree is given by

$$l = \frac{m-1}{k-1}. \tag{2.6}$$

A simple way to decrease the memory requirements of the tree data structure is to raise the degree of the tree. Suppose a complete binary tree is given with $2m-1$ nodes. By switching to a tree with a larger degree $k > 2$, and assuming that the tree remains complete, or almost complete, also after the switch, the reduction factor of the total number of nodes in the tree is well captured by the equation

$$\eta(k) = \lim_{m\to\infty} \frac{2m-1}{m + (m-1)/(k-1)} = 2 - \frac{2}{k}. \tag{2.7}$$

For example, going from degree 2 to degree 8 reduces the number of internal nodes in the resulting tree by a factor of seven, and the total number of nodes by approximately $\eta(8) = 1.75$.

Another prominent property of BVHs are their ability to approximate objects, rather than space, in an efficient way. This is in contrast to spatial space partitioning data structures, such as octrees, kd-trees, and grids, where the opposite is generally true. For example, this means that the child volumes in a BVH are allowed to intersect, and thereby partly covering the same space, see Figure 2.1. This makes it possible to insert each geometric primitive into a single leaf node in the tree, which effectively avoids the reporting of duplicate hits in search queries. This also leads to another advantage of the BVH data structure: the memory requirement of a BVH is always linear in the number of geometric primitives, as opposed to spatial space partition data structures, such as kd-trees and octrees, which sometimes requires super-linear storage space.

Shape	Fit	Test speed	Memory cost	Rot.Inv.
AABB	poor	good	6	no
26-DOP	fair	fair	26	no
Sphere	poor	good	4	yes
OBB	good	poor	15	yes
Ellipsoid	good	poor	15	yes
SCB	fair	fair	9	yes

Table 2.1: A rough comparison of the properties of different types of BVs. Note that no BV is best in all cases. The first two properties, tightness of fit and speed of the BV-BV overlap test, are given using a relative 3-degree scale (poor, fair, good). The memory requirements are given as the number of scalar values commonly used to represent the shape. The last column shows the rotational invariance of the shapes.

2.2 Choice of Bounding Shape

To realize a BVH we have to choose what shape (or shapes) to employ as bounding volumes in the nodes of the tree. Which shape to choose turns out to be a very important design choice. Two key factors to consider are the tightness of fit of the volume and how fast the required geometric tests are [35]. Figure 2.2 gives an example of two different types of bounding volumes, the sphere and the slab cut ball (SCB), which is a sphere cut by two parallel planes. In practice, a handful of simple convex volumes appear to be the most popular, for example the sphere [36, 37, 38, 39, 40, 41], axis-aligned bounding box (AABB) [42, 43, 44, 45, 46], oriented bounding box (OBB) [47, 32, 48], and discrete-orientation polytope (k-DOP) [49, 50, 51].

For all these bounding volumes types, there are simple and efficient algorithms to compute a minimal, or almost minimal, BV which encloses a given point or polygon set [52, 53, 54]. In particular, how to compute minimum bounding spheres, also called smallest enclosing balls, is a well-studied classical problem in computational geometry. Perhaps somewhat surprisingly, theoretical results show that the optimal bounding sphere of a point set can be computed in worst case $O(n)$ time [55]. Several other more practical methods for computing the optimal ball have also been presented, such as the recursive algorithm by Welzl, which has an expected linear running time by relying on randomization of the input points and a move-to-front heuristic [56, 57]. Some other inter-

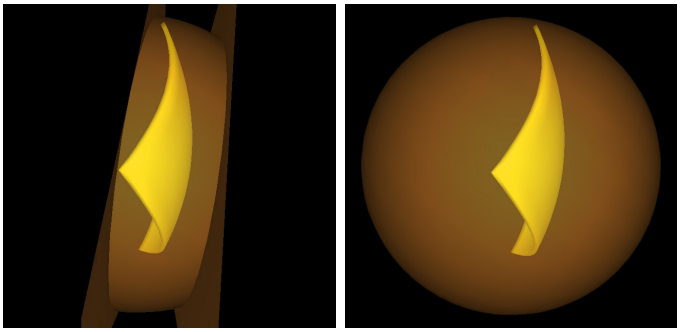

Figure 2.2: Examples of two different types of bounding volumes enclosing a polygon mesh. As can be seen, the slab cut ball (left) provides a much tighter approximation of the underlying mesh than the sphere (right), which speaks in favour for the slab cut ball. Another important factor to take into account is how fast geometric operations on the volumes can be performed and in this respect, the sphere is expected to be more advantageous.

esting choices of bounding volumes include the ellipsoid [58, 59], cylinder [60], sphere swept volumes [61, 62], quantized orientation slabs with primary orientation (QuOSPO) [63], intersection volume of a sphere and AABB [64], and spherical shell [65].

Clearly, the best choice of BV type appears to be highly dependent on both model and scenario. This conclusion is supported by Table 2.1, which presents a simple comparison of some important properties for some interesting BV types. For example, spheres are invariant under rotation, but the tightness of fit is in general quite poor. OBBs are known to have a good tightness of fit, but they also have a high storage cost, as compared to AABBs or spheres. Geometric tests on AABBs are generally very fast, but since AABBs are not invariant under rotation, recomputation of the shape is required even for simple rigid body motion. Alternatively, if the AABBs are simply rigidly transformed together with the moving bodies, they become OBBs.

The list of proposed bounding volume types in the research literature is steadily growing. Some of the more recent proposals include the zonotope [66], slab cut ball (SCB) [67], and velocity-aligned discrete oriented polytope (VADOP) [26]. Since it has been shown that the choice of

BV type in a BVH sometimes influence the execution time of geometric queries dramatically, a wise selection of the BV type with respect to the current application may be crucial (see e.g. [23, 67]).

2.3 Hierarchy Construction

Besides choosing an appropriate type of bounding volume, a tree-building algorithm must also be selected. Normally, the input consists of a set of geometric primitives, and the output is a partitioning (decomposition) of these primitives into a regular tree data structure, where each tree node stores a BV enclosing all the primitives in its subtree. However, since the number of structurally different BVHs that can be produced grows exponentially with the number of input primitives, finding a globally optimal tree structure is considered intractable. Instead, many different heuristics have been developed which can be categorized into three main types of hierarchy construction algorithms, which are often referred to as top-down, bottom-up, and incremental insertion construction methods.

In practice, top-down construction seems to be the most commonly used method. The example BVH in Figure 2.1 was constructed using a simple top-down building approach. In a top-down building algorithm, the root node is created first, and a BV is computed which encloses all the primitives. Then follows a recursive step, where the remaining primitives are divided into k subsets, and k child nodes are created with BVs enclosing these subsets. This recursive step is then applied for each created node, unless the remaining primitives are below a given threshold, in which case the recursion is terminated and a leaf node is created enclosing the remaining primitives. How to divide the primitives into appropriate subsets is managed by a so called split rule, which is often based on a sorting or bucketing strategy to determine the subsets. Klosowski et al. gives several examples of split rules [49]. Top-down BVH building belongs to the divide-and-conquer family of algorithms. Given that both the BV computation algorithm and the split method runs in $O(n)$ time, and that the produced tree structure has a height of the order $O(\log n)$, the whole tree building procedure runs in $O(n \log n)$ time. If an unbalanced tree is produced, however, the performance deteriorates to $O(n^2)$. This case is an analogue to the worst case behaviour of quicksort [68], and by using a robust and carefully designed split method, the worst case behaviour can be avoided in practice (see e.g. [67]).

In contrast to top-down approaches, constructions of hierarchies from the bottom-up are based on merging nodes rather than splitting them [32, 59, 69]. The construction starts by creating the leaves in the tree data structure, where each leaf node has a BV enclosing either a single geometric primitive or a few primitives located close to each other. Then follows a process where nodes with nearby BVs are grouped together to find appropriate parents. This grouping continues until there is only one node left, the root. Note that when k nodes are to be grouped, their BVs need to be merged to find a proper BV of the parent. The running time of the bottom-up construction depends on the time complexity of both the merge algorithm and the algorithm used to select suitable nodes to be merged. Since at least $O(n)$ merging operations and selection operations have to be done, the overall construction is $O(n)$ in the best case. Usually, however, more sophisticated BV merging and/or node grouping strategies are employed to ensure a better quality of the hierarchies, resulting in a time complexity of $O(n^2)$ or worse [69]. Clustering algorithms are essential for bottom-up construction of BVHs to guide the grouping of nodes. Since clustering is an important concept in many different research fields, much research has been conducted on clustering algorithms [70]. For example, practical methods for solving the facility location problem efficiently [71] can be applied in a BVH construction scheme. A quite different bottom-up construction method based on estimates of the mass distribution of a volumetric model has also been proposed to produce binary bounding volume hierarchies [72, 73]. In general, bottom-up construction algorithms are more complicated to implement and usually run slower than top-down methods [74].

BVHs can also be built using incremental insertion methods [75, 76]. The idea behind these methods is to add or insert one geometric primitive at a time to an initially empty hierarchy. Usually, the insertion proceeds from the root node to a leaf, where the path taken depends on a cost function that is used to minimize the insertion cost locally. Then, the insertion point of the primitive is chosen to be the node along this path that minimizes the total volume of the tree. Given that the evaluation of the cost function and the volume enlargement operation for each encountered node during insertion is executed in constant time, the whole hierarchy tree construction is expected to take $O(n \log n)$. Thus, this way of building BVHs is in general as practical and fast as top-down approaches. For example, incremental insertion guided by surface area heuristics has been used to build BVHs to accelerate ray tracing [75, 77].

The quality of the produced BVHs, however, depends on the insertion order of the primitives. Thus, randomizing the insertion order may be worthwhile. Finally, after all primitives have been inserted, an additional restructuring phase may be used to improve the hierarchy structure. Haber et al. propose two such global optimization heuristics referred to as successive re-insertion and elimination of ill-formed groups [76].

Omohundro presents five ball tree construction methods, one bottom-up, two top-down and two incremental insertion algorithms. Interestingly, the experimental results revealed that the bottom-up method produced the trees of highest quality. However, its high construction cost limits its usefulness. Consequently, for large data sets, a simple top-down or an incremental insertion method is preferred [74].

Regardless of which one of these three main classes of construction methods is chosen, several other important design issues must be dealt with. Clearly, balanced hierarchies may seem attractive from a theoretical perspective. For some inputs and construction methods, however, balanced BVHs may lead to a significant overlap between nearby BVs leading to truly inefficient search queries. Since the goals of minimizing the tree depth as well as the BV overlap between nearby nodes are usually in conflict with each other, the construction algorithm needs to deal with a trade-off between the maximum allowed depth of nodes, and the amount of overlap between adjacent volumes that can be tolerated. Construction speed is another factor that is involved in this trade-off. To exemplify, consider the well known issue of selecting pivot element in quicksort. Strictly enforcing a completely balanced partitioning of the elements to be sorted, by always selecting the median as the pivot element, would of course avoid the worst-case $O(n^2)$ behaviour of quicksort, but it would also make quicksort much slower for almost all inputs. Instead, choosing the median-of-three randomly selected elements as the pivot element has been widely adopted, because it is very fast, and it also effectively avoids the worst-case in practice.

Another reason why it sometimes makes sense not to require the hierarchies to be balanced is update cost. The tree structure may become too expensive to maintain due to sudden dynamic changes during runtime. However, classic tree data structures such as almost balanced AVL-trees [78, 79] and red-black trees [80, 31] may of course be interesting to adopt in BVH creation and maintenance. The height of a red-black tree is guaranteed to be within a constant factor of two compared to the height of a balanced binary tree. Thus, for a balanced binary tree

with n nodes, the height of a corresponding red-black tree cannot exceed $2\log_2(n+1)$. Another interesting data structure is the splay tree, which is a type of binary search tree that automatically moves frequently accessed elements nearer to the root [81]. Similar techniques may be fruitful also for BVHs to optimize query times for application-specific scenarios.

Another concern in the design of BVHs is the choice of the branching factor. In practice, the most common choice seems to be binary BVHs. Some analytical arguments for choosing node degree $k = 2$ are given by Klosowski et al. [49]. However, no definite answer to what is best has been given. Clearly, a higher k gives shorter search paths from the root to the leaves, but at the same time the work at each encountered node increases, since there are more branches to consider. The opposite holds true for a lower k, which makes the search paths longer, but the operations and branch selections at each node faster. In the end, the best choice appears to be application and machine specific. It is not unusual that practical experiments indicate advantages for using a $k > 2$ [44, 51]. In particular, parallelization of the work at the hierarchy nodes speaks in favour of choosing multi-way trees. Sometimes binary tree nodes are chosen simply because this appears to simplify the design of the construction algorithm. In any case, however, a binary BVH structure can be converted to a quaternary BVH simply by removing every other level and to an octonary BVH by removing two levels at a time.

Uneven distribution of the sizes of the geometric primitives in the models is another source of inefficiency in many constructed BVHs. Consider, for example, a giant polygon stored in a leaf node at the maximum depth of a BVH. This polygon will cause the computation of huge bounding volumes in all nodes from this leaf all along the path up to the root node. In the worst case, this polygon is so large that all these bounding volumes must have the same size, although they are minimal. In fact, it will then overlap entirely with all other BVs in the hierarchy. If there are many giant polygons in a model, this problem degrades the performance severely, because of the resulting overlaps among lots of BVs internally. How can the construction algorithm deal with giant polygons to avoid unnecessary performance breakdowns? A possible solution could be to store problematic large geometric primitives in internal nodes. In this way, these primitives can be stored at a much higher level in the hierarchy to avoid a troublesome expansion of all the BVs in a complete path down to a leaf node. A similar technique would be to add an extra leaf node that stores the primitive directly below the internal node in ques-

tion, thereby raising the degree of the internal node, rather than storing primitives in internal nodes. Another way of attacking the problem with problematic variation in the primitive sizes would be to clip primitives and/or BVs that are considered too large into several pieces [82]. In this case, however, the attractive $O(n)$ storage cost of a BVH can no longer be guaranteed.

The final design choice discussed here is related to how the bounding volumes are computed. In a layered hierarchy, each bounding volume of a parent node completely covers all bounding volumes located in its subtree. However, this property of a layered hierarchy is not always needed. In many cases, it is preferable to build tighter fitting hierarchies by letting each parent BV completely cover all the geometric primitives located in its subtree, rather than also enclosing all the bounding volumes in the subtree. A hierarchy with this property is sometimes referred to as a wrapped hierarchy. Note that a wrapped hierarchy is always as tight or tighter than the corresponding layered hierarchy [83, 30]. For some applications, however, a layered layout of the BVs is chosen, either because of the way search queries are executed or because BV update operations can be made faster [84, 85].

In addition to the points discussed above, construction of memory [86, 87] and cache friendly [34, 88] hierarchies is also attractive. How to construct BVHs of high quality is a complex subject, and it remains an open research problem which can be attacked from many angles.

2.4 Fundamental Operations

Since the BVH is a spatial data structure, it is mainly used to perform different types of geometric queries concerning the relative location of objects. The queries are realized by designing different types of search algorithms traversing one or more BVHs. For example, BVHs can be used to efficiently find ray-primitive intersections, or to determine potentially visible primitives from a certain viewpoint, in a scene. BVHs can also be utilized to perform fast distance queries, such as finding the nearest neighbour to a given query object, or to report all the intersecting primitives between two models. Queries for collision detection or interference detection and ray tracing are discussed further in Chapter 3.

The computational complexity of these operations is dependent on several factors such as the tree height, the ability of the BVs to approx-

imate the underlying geometry tightly, the amount of overlap between BVs inside the BVH, and the size of the output, i.e., the number of elements in the search result. Although the theoretical worst-case time complexity of a BVH-based search algorithm may sometimes look daunting, BVHs are well-known for their good performance in many computer graphics applications.

As an example, consider the case of performing a query to find all the primitives hit by a ray. This operation is expected to take $O(\log n)$ time, and in the best case, when the ray misses the BV in the root, the query even finishes in $O(1)$ time. In the worst case, however, all BVs in the hierarchy are hit by the ray, which means the ray query is $O(n)$, which admittedly does not look promising for ray shooting. Still, BVHs are used to accelerate ray tracing with good results [46]. Interestingly, since the performance of a ray-BVH traversal is dependent on the actual number of BVs hit by the ray, the worst case of a traversal is given by the stabbing number s, which is the maximum number of BVs that can be hit in a traversal seen over all possible rays [32]. This means that the worst case performance of ray shooting using a BVH is $O(s)$. Therefore, a natural design goal in BVH construction for ray tracing, besides keeping a logarithmic height of the tree, would be to keep the stabbing number $s \in O(\log n)$, which in particular involves avoiding too much overlap between sibling volumes.

How to design efficient BVHs for different applications with theoretically proven asymptotic worst-case bounds remains an open research question. Only a few research efforts in this direction have been published, see e.g. [89, 83].

2.5 Scene Graphs

A scene graph is a common data structure that is related to the BVH. Usually, a scene graph represents both logical and spatial relations in a scene or virtual environment. In a scene graph, different types of nodes, representing e.g. groups of objects, transformations, geometric primitives, light sources, and cameras are arranged into a tree or directed acyclic graph (DAG) [90, 91].

Interestingly, several scene graph packages also let the scene graph play a role as a BVH, since it is usually straightforward to extend a scene graph to also become a BVH by storing bounding volume data

at appropriate locations in the structure. In this way, an acceleration data structure is directly available together with the scene description at a small additional memory cost as compared to using another separate data structure, such as a kd-tree or octree. Operations such as view-frustum culling, picking, collision detection, and range queries can then be implemented conveniently as different kinds of scene graph traversals.

2.6 Adaptive Hierarchies

Whenever possible, the BVHs are created in a preprocess or during application initialization, since most construction algorithms are super-linear. Once built, the data structure can then be used to perform various types of queries without performing any time-consuming changes or updates to the hierarchy during run-time. Many scenes, however, involve different types of dynamic features. For example, geometric objects may be stationary or moving. Objects in motion may be rigid, deformable, and even breakable. New geometric objects may be inserted on-the-fly in the scene, due to unpredictable events. Keeping BVHs up to date due to dynamic changes like these constitutes a significant challenge in real-time graphics simulations.

If a BVH has the attractive feature that it remains useful in a real-time graphics simulation even when the circumstances change by adapting to the new situation, we refer to the BVH as an adaptive hierarchy. For deformable models, this means that when the shape of a model is changed, its BVH can adapt to the new situation in an efficient way by, for example, BV refitting schemes [44, 84, 92], incremental reconstruction [45, 93], and/or amortized updating [46] and afterwards still remain an efficient acceleration data structure. Clearly, careful design of insertion and deletion operations is essential for intelligent dynamic updates of tree data structures (cf. [78, 80, 81]). For example, AVL-trees have been leveraged in a BVH-based approach for faster collision detection between fracturing objects [94].

Even for rigid models, it makes sense to refer to a BVH as adaptive if it has the ability to remain efficient over a wide range of queries and scenarios. In particular, if a number of models in a rigid body simulation all of a sudden enter a highly complex configuration with respect to the geometric queries, without significant performance breakdowns, the BVH can be said to be adaptive to this new complex scenario, even

though the actual BVH data structures are static [47, 67]. The performance of a BVH can also be improved by learning from actual use. For instance, the hierarchy could be restructured to provide faster access to frequently queried elements, and construction of the data structure can also be deferred until queries are issued. In this way the trees are constructed piece by piece guided by actual queries. Such techniques have been proposed to create adaptive BSP-trees with good results [95, 96].

The loose octree can also be seen as an adaptive space partitioning data structure. Similarly to the BVH, the loose octree allows overlapping cells or blocks [97, 3]. By expanding the block size in each direction by, for example, a factor of two, too small objects straddling the previously unexpanded block borders can be inserted at more appropriate levels in the octree, which may lead to more efficient spatial queries. Also, since objects can be inserted or deleted in $O(1)$ time, the loose octree seems to be suitable for dynamic environments with a large number of moving objects.

As can be seen in Chapter 4, adaptive BVHs for collision queries in dynamic simulation environments is what this thesis is mainly about. Of course, the ultimate goal of adaptive BVHs is that one type of BVH could be used for every type of model, query, and scene. It seems clear that research in this area need to focus more on developing BVHs with a broader applicability.

Chapter 3

Collision Queries

As discussed in Chapter 1, being able to answer collision queries in complex scenes is a fundamental requirement in virtual environments. Many efforts have been described in the research literature to solve this problem. Therefore, this chapter gives an overview of important and related research. Two types of collision queries are considered more closely. The first one is needed in almost all kinds of rigid and deformable body simulations. Given a set of geometric models, are there any contact points between them? Various attempts to solve this kind of collision detection or interference determination problem are reviewed briefly in the next section.

The second query type considered is ray shooting, which is a fundamental operation in many rendering algorithms. Given a set of rays, are there any contact points between the rays and the scene objects? In Section 3.2, a background to ray tracing is given and previous attempts to solve the ray shooting problem are discussed.

3.1 Collision Detection

Hundreds of papers have been written on collision detection in various situations, primarily in the fields of computer graphics, robotics, and computational geometry. Whereas most early efforts were focused on solving the collision detection problem in rigid body simulation [98, 99, 100, 28, 52], nowadays deformable bodies also receive significant attention [101]. There is currently no single best collision detection method.

The algorithm to be chosen depends on many factors that play different roles in different applications [47].

In some applications it is sufficient to use approximate methods whereas other applications might require accurate collision calculations. The best performance is often achieved by using specialized or simplified methods that utilize specific knowledge about the application. For example, in a virtual bowling application, simple cylinder approximations were used to represent the pins in the collision detection calculations with plausible results [102]. Another example where application specific knowledge has been utilized to speed up the CD significantly can be found in water wave simulation where precomputed wave-land interaction points are stored in wave-train boxes with fixed locations [103]. Sometimes, an application-specific solution may even include algorithms that make the collision detection obsolete, as is the case in a proposed method for finding the range of motion in the human hip joint [104].

In many other cases, a sufficient accuracy of the collision calculations must be guaranteed. For example, in robotics, inaccuracies in the virtual simulation process might lead to severe damage, since the simulations are often used to verify the correctness of the corresponding real world scenarios. Furthermore, in rigid body simulation, when the force computations are based on the intersection data reported from the collision detection algorithm, small errors might cause fundamentally different body trajectories, which is unacceptable in certain applications. In general, what actions to perform given the results reported by the CD process is determined by the collision response algorithm [105, 106, 41]. Whereas CD is fundamentally a math or geometry problem, collision response is usually a physics or dynamics problem.

The combined need for accuracy and speed in real-time simulations makes the collision detection problem very challenging. The time available to resolve the collisions may, for example, be somewhere in the range 0.1–5 milliseconds, depending on the application, so highly efficient solutions are needed. Some fast search methods are available, when the involved bodies are convex [107, 108, 22, 109]. Concave objects can also benefit from these methods if they are decomposed into convex parts [110].

For more general and complex rigid bodies, bounding volume hierarchies have often been found to be the best choice [101]. Examples of bounding volumes that have been used for efficient CD between rigid bodies are spheres [38, 37], axis-aligned bounding boxes (AABBs) [111,

43], arbitrarily oriented bounding boxes (OBBs) [47, 32], discrete orientation polytopes (k-DOPs) [49, 50], spherical shells [65], slab cut balls (SCBs) [67], tetra-cones [112], and convex pieces [110].

To further speed up hierarchical collision detection methods, temporal coherence can also be utilized. By using different types of caching techniques, results from the previous simulation time step can be reused for faster determination of new results [113, 114, 110].

In the case of soft or deformable bodies, much work remains to be done [99]. For CD between deformable bodies using BVHs, spheres, AABBs, and k-DOPs are very attractive, since refitting these volumes is both simple and fast [44, 84, 51, 92, 85, 39, 115]. For highly dynamic triangle soups, octrees [116, 117], uniform grids [118, 18], hierarchical spatial hashing [19, 119], and BVHs [45] have been used with good results. Also, the recent development of programmable graphics hardware has made GPU-based CD methods an interesting alternative [120, 121, 122, 123]. Some other interesting efforts aimed at different geometries and types of applications that have been described include methods for higher order surfaces [124, 125, 126] and cloth simulation [127, 128].

Some initial work has also been performed in the field of virtual surgery. One proposed method relies on graphics hardware to test the interpenetration of a deformable organ and a user-controlled rigid tool [129]. In a work on laparoscopic surgery, a special bucket data structure was used to store closely located polygons [130]. This data structure was then used to search for contacts between a simple tool and an organ represented by a polygonal mesh.

Another important topic is continuous collision detection (CCD), which can be used to improve accuracy in, for example, motion planning application [101]. In this area, the methods aim to avoid unwanted tunnelling effects, which may arise due to discrete time stepping. Usually, time-swept versions of the geometric primitives between two consecutive discrete time instants are employed in the overlap tests. Naturally, these volumes depend on the motion trajectories of the models used in the simulation. The real motion of the bodies can be approximated by, for example, linear interpolation between the start and end positions of the geometric primitives, thereby trading accuracy for speed. As in discrete CD, BVHs are also frequently utilized for CCD [48, 131]. However, since CCD is computationally much more costly than discrete CD, improved testing schemes are needed. For instance, feature-based hi-

erarchies have been proposed, which reduces the number of elementary tests between feature pairs during the CCD queries between deformable triangle meshes. The reduction is accomplished by using representative triangles, i.e., triangles that are associated with a limited set of their features (edges and vertices) [132]. This concept, however, only works for meshes with topological connectivity information, and not for more general polygon soups.

```
DualBvhTraversal(A, B, r)
    input:   A and B are hierarchy nodes
    output:  r is a container storing the intersection result
1.      if Intersection(V1 ∈ A, V2 ∈ B) then
2.          if Internal(A) and Internal(B) then
3.              if Volume(V1) > Volume(V2) then
4.                  for each child c ∈ A
5.                      DualBvhTraversal(c, B, r)
6.              else
7.                  for each child c ∈ B
8.                      DualBvhTraversal(A, c, r)
9.          else if Internal(A) then
10.             for each child c ∈ A
11.                 DualBvhTraversal(c, B, r)
12.         else if Internal(B) then
13.             for each child c ∈ B
14.                 DualBvhTraversal(A, c, r)
15.         else
16.             for each primitive pair (t1 ∈ A, t2 ∈ B)
17.                 if Intersection(t1, t2) then
18.                     Insert(r, t1, t2)
```

Figure 3.1: Pseudocode for a recursive BVH-based, 2-body, CD traversal.

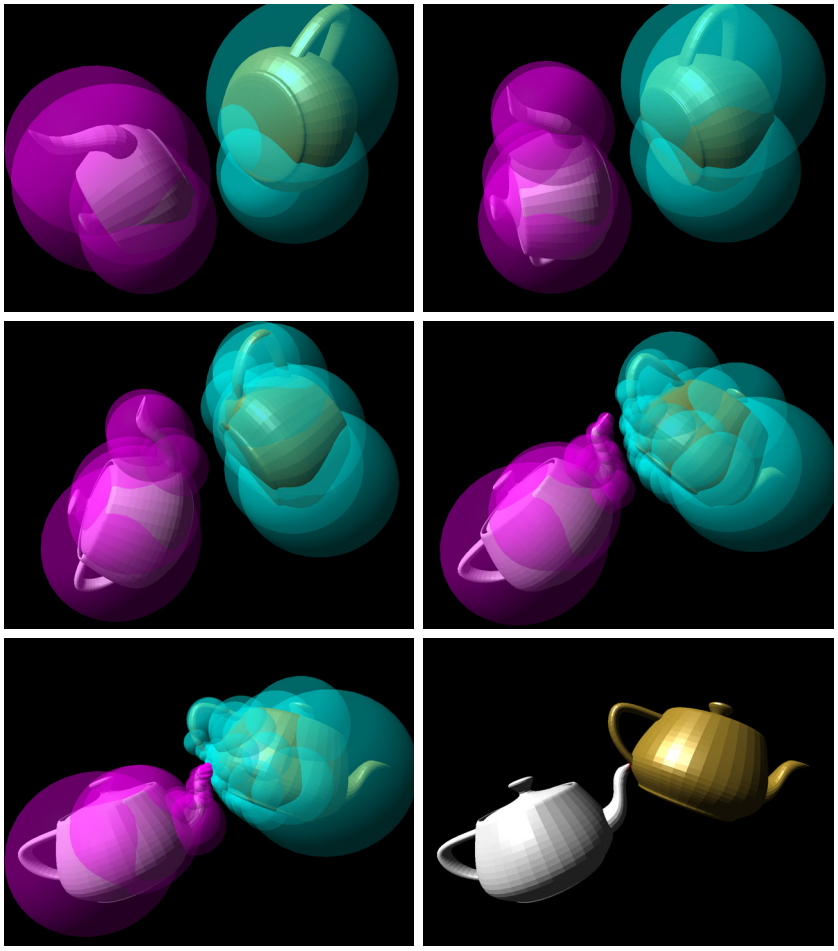

Figure 3.2: Visualizations of the dual hierarchy traversal between the BVHs of two teapot models for five different time steps in a simulation. In each case, only the deepest spheres encountered in the tree branches during the dual traversal are shown. Thus, the number of spheres rendered to visualize a dual traversal here is a measure of the amount of work required to determine the collision status. As can be seen, the closer the teapots get to each other, the more work the algorithm has to perform. The bottom row shows the traversal at the last time step, where the models are finally colliding with each other.

3.1.1 Collision Detection using BVHs

To check the collision status of two models, their bounding volume hierarchies are traversed in tandem while searching for intersecting primitive pairs. The pseudocode for such a dual hierarchy traversal is given in Figure 3.1. First of all, the overlap status between the BVs of the root nodes is tested. If these volumes are disjoint, testing is done in $O(1)$ time (Line 1). Otherwise, recursive refined testing is performed by descending in the subtree of the node with the largest current volume (Lines 2-8). The conditional test used here to decide which subtree to descend next (Line 3), can be replaced by another, perhaps more appropriate, so called descent rule (for some examples, see [52]). When one of the current nodes A and B is a leaf, we descend in the subtree of the remaining internal node (Lines 9-14). Finally, when both nodes are leaves, the geometric primitives associated with these leaves are intersection tested, and intersecting pairs are inserted in a CD result list (Lines 15-18). After the completion of the algorithm, the list of intersecting primitive pairs is usually passed on to a collision response method for further processing.

In situations where a simple true or false answer is sufficient, the dual BVH traversal can be aborted when the first intersecting primitive pair is found, which leads to significantly faster query times in many cases [44]. For this, the recursive dual traversal presented in Figure 3.1 can easily be transformed into a corresponding iterative traversal by using a stack, and an immediate exit on the first found hit can then be realized by a single return statement.

To get as good performance as possible, the algorithm in Figure 3.1 is also dependent of highly optimized low-level intersection test methods (Lines 1 and 17). How to optimize such low-level routines remains an important topic in computer graphics, since usually they are part of the inner loop of more complex geometrical algorithms (see e.g. [133, 134, 52, 135, 53]).

A visualization of a dual traversal between the BVHs of two teapot models is given for five different collision queries in Figure 3.2. As can be seen, the dual traversal effectively zooms in on close surface areas between the two teapots. The performance of the traversal is dependent on the number of geometric primitives in the models, as well as the number of overlapping BV pairs encountered during the traversal. For rigid bodies, a traversal is expected to be sub-linear in many cases, even when intersecting primitive pairs are found, since the height of a

hierarchy storing n primitives is expected to be proportional to $\log n$.

In the worst case, however, this BVH traversal algorithm is $O(n^2)$. For example, Chazelle's polyhedra illustrate that the number of overlapping geometric primitive pairs can be quadratically many [136, 30]. In such cases, a BVH-based CD approach does not offer much improvement over the naive method mentioned in Chapter 1. The bounding volume test tree, which is a structure that captures the behaviour of the above presented dual hierarchy traversal algorithm, has a maximum size of $O(n^2)$ nodes, and hence it also illustrates the worst case behaviour of the algorithm [61, 62, 23].

Despite this theoretical quadratic worst case, BVH-based CD is repeatedly reported to be highly successful in practice. Some attempts to theoretically explain the good performance of BVHs in practice under certain assumptions have been published [137, 138, 139]. Interestingly, some of these assumptions may be incorporated as design goals in BVH construction for collision detection. Also, Haverkort et al. present some theoretical results for range queries [89].

The choice of which bounding volume type to use is not simple, as discussed in Section 2.2. Therefore, to evaluate the performance of bounding volume hierarchies, it has been suggested that a cost function can be used [35, 47, 49]. This function states that the cost, T, of a certain collision detection query is given by

$$T = N_v C_v + N_p C_p + N_u C_u, \tag{3.1}$$

where N_v is the number of performed BV/BV intersection tests and C_v is the cost of one such test. Similarly, N_p is the number of geometric primitive pairs that are intersection-tested and C_p is the cost of one such intersection test. Finally, N_u is the number of BVs that are updated or recalculated because of model changes and C_u is the cost of updating one BV. By using tighter fitting bounding volumes in the hierarchies, N_v, N_p, and N_u can be lowered, but on the other hand, tighter volumes often mean larger values of C_v and C_u. To minimize the cost function, one has to deal with such conflicting goals.

Despite all the previous efforts, new and faster collision detection methods are needed to increase speed and realism in both rigid and deformable body simulations. The algorithms proposed in this thesis are fast and accurate down to the finest resolution of the models. The methods are based on adaptive bounding volume hierarchies. Summaries of these methods are given in Chapter 4.

3.2 Ray Tracing

Ray tracing is a classic image synthesis technique. It was introduced as early as 1968 by Appel as a shadow determination technique [140]. In 1980, Whitted published an article describing the basic recursive ray tracing algorithm, which extended the original algorithm to handle specular reflection and refraction [141]. This version of the algorithm is essentially the basic ray tracing method described in many computer graphics textbooks of today [142, 143, 144]. Hundreds of articles on different aspects of the subject have been published and there are books entirely devoted to ray tracing [145, 146, 147, 148].

In ray tracing, images are generated by tracing rays of light backwards, from the eye through the pixels in the image plane. These rays are then recursively traced according to the rules that have been set up for their interaction with the three-dimensional scene while the colour contributions are gathered.

While ray tracing is well known for its ability to create stunning pictures, it seems to be equally well known for its extremely high computational cost. Naive ray shooting is $O(n)$, and naive ray tracing of an entire image with k pixels is then $O(kn)$. We may regard the image resolution as constant and drop the factor k, but still the number of pixels is substantial using, e.g., a full HD resolution of 1920×1080 pixels. Using a super-sampling method to generate higher quality anti-aliased images increases k even further since several rays per pixel are spawned. For example, to generate a single image in full HD, super-sampled with 9 primary rays per pixel, almost 20 million primary rays need to be intersected with the geometry in the scene and potentially more than 100 million secondary rays, i.e. shadow, reflections and refraction rays. Thus, given complex scenes, with millions of geometric primitives, the number of intersection calculations required is huge. This disadvantage of ray tracing caused it for many years to be considered as an offline rendering method only.

The tremendous recent improvements in computer technology, however, have begun to change the view of ray tracing as a rendering method which is too slow for interactive graphics applications. In real-time ray tracing, something like 30–90 generated images per second are needed. To be able to handle the computational burden, clever algorithms are needed to reduce the number of intersection computations and utilize the inherent parallelism of the ray tracing process.

Today, it has been shown that interactive ray tracing is possible, even on affordable PCs [149, 150, 151, 152, 153, 46]. Still, however, achieving real-time ray tracing requires highly optimized implementations which besides a carefully constructed acceleration data structure, also utilize, for example, vectorized instruction sets, ray packets, cache-coherent traversal schemes, and multi-threading.

Interestingly, fast ray tracing can also enable faster global illumination computations, since most global illumination algorithms rely heavily on ray tracing [154]. Thus, ray tracing may play an important role in achieving a long term goal in computer graphics — physically correct simulation of light transport at interactive frame rates in complex and dynamic environments.

Most proposed ray tracing approaches, however, are dependent on acceleration data structures that are pre-built before simulation. Many of these methods rely on certain limiting assumptions, such as that it is only the camera which is animated, the rest of the scene remaining static. Others have permitted the presence of certain dynamic models in the scene, as a special case, but these have been assumed to be rigid bodies [149]. Grids are a noteworthy exception, since they can be efficiently reconstructed when used in animated scenes [155, 153, 16]. Fast construction of other acceleration data structures has also become possible using today's powerful computers [156, 157]. Programmable shaders have also been used to implement interactive ray tracing on commodity graphics hardware [158, 159].

3.2.1 Ray Tracing using BVHs

To make ray tracing a competitive choice and an interesting alternative for interactive graphics applications in general, dynamically changing scenes must be supported. This also includes scenes with complex deformable or flexible models, including breakable and exploding objects. Although kd-trees have been used with much success for real-time ray tracing of static scenes [151, 152], they are less suitable for dynamic scenes due to their relatively high construction costs. In this respect, BVHs are more attractive, and several recent papers propose using them for ray tracing of both rigid and deformable scenes [160, 86, 161, 87, 46].

Furthermore, BVHs are more advantageous than kd-trees by requiring asymptotically less memory space. This is another major reason why BVH-accelerated ray tracing has gained a lot of interest lately with

the goal of constructing BVHs that can keep up with the best kd-trees also for ray tracing static models. In particular, the surface area heuristic (SAH) introduced by Goldsmith and Salmon has become known as an effective technique for evaluating the quality of building operations during hierarchy construction [75]. Several new improved construction methods based on SAH have been developed recently [77, 69]. Unfortunately, large geometric primitives giving rise to BVs with significant overlap are not handled efficiently using SAH and traditional BVHs. To ensure a high quality of the BVHs for scenes with such primitives too, a technique called early split clipping has been proposed [82]. Other recently proposed BVH-like data structures, which are also influenced by the kd-tree, includes the bounding interval hierarchy (BIH) [162], DE-tree [163], and B-KD Tree [164]

Real-time ray tracing is an emerging research field with many problems left to solve. Clearly, highly efficient and robust techniques for rendering complex and dynamic scenes at steady real-time rates are needed. Although the proposed BVHs and acceleration methods in this thesis are focused mainly on collision detection, several of the techniques can be applied to improve ray tracing as well. In particular, the algorithms for efficient reconstruction and updating of BVHs due to model deformation during simulation are equally important for both collision detection and interactive ray tracing. The applicability and benefits of these novel methods are presented next in Chapter 4.

Chapter 4

Contributions

This chapter includes summaries of the conducted research and the main contributions are emphasized. The results have been published in seven papers, which are listed in Table 4.1. In the following discussion, these papers are referred to as papers **A**–**G**. For a more detailed treatment, consult the original papers. For convenience, the full text of the papers is also reprinted in part **II** of this thesis.

The main contribution of paper **A** is a new algorithm for hierarchical collision detection of meshes undergoing arbitrary vertex repositioning. In particular, a hierarchy update method based on a new efficient BV refitting scheme is given, yielding a significant speedup compared with previous approaches. Paper **B** introduces a novel algorithm for collision detection of morphing models, whose performance is of the same order as algorithms previously used for hierarchical collision detection of rigid bodies. In paper **C**, the main contribution is a new algorithm for ray tracing of deforming meshes. The paper shows that the hierarchy update method, which was proposed in paper **A**, gives a speedup of an order of magnitude in the reconstruction phase, which is needed in ray tracing of dynamically changing scenes. This makes feasible the interactive ray tracing of scenes with hundreds of thousands of deforming geometric primitives. Paper **D** introduces a generalized collision detection algorithm based on dynamic BVHs. The method allows efficient CD even for unstructured breakable or exploding objects. In paper **E**, a BVH based on a novel type of BV, the slab cut ball, is introduced, which enables highly efficient CD queries between rigid bodies. In papers **F**

A Thomas Larsson and Tomas Akenine-Möller. Collision Detection for Continuously Deforming Bodies. In Eurographics Conference 2001, Short presentations, pages 325–333, Manchester 2001.

B Thomas Larsson and Tomas Akenine-Möller. Efficient Collision Detection for Models Deformed by Morphing. *The Visual Computer*, 19(2–3):164–174, 2003.

C Thomas Larsson and Tomas Akenine-Möller. Strategies for Bounding Volume Hierarchy Updates for Ray Tracing of Deformable Models. MRTC Report, Mälardalen University, February 2003.

D Thomas Larsson and Tomas Akenine-Möller. A Dynamic Bounding Volume Hierarchy for Generalized Collision Detection. *Computers & Graphics*, 30(2):451–460, 2006.

E Thomas Larsson and Tomas Akenine-Möller. Bounding Volume Hierarchies of Slab Cut Balls. MRTC Report, Mälardalen University, June 2008. (Conditionally accepted for publication in *Computer Graphics Forum*.)

F Thomas Larsson, Tomas Akenine-Möller, and Eric Lengyel. On Faster Sphere-Box Overlap Testing. *journal of graphics tools*, 12(1):3–8, 2007.

G Thomas Larsson. An Efficient Ellipsoid-OBB Intersection Test. *journal of graphics tools*, 13(1):31–43, 2008.

Table 4.1: The papers included in this thesis.

and **G**, efficient overlap tests are proposed between boxes and spheres, and boxes and ellipsoids. In CD, intersection tests between BVs must be highly optimized, since they are usually part of the inner loop in the hierarchy traversals.

4.1 Research Methodology

In all these above-mentioned works, the main methodology employed has been based on literature studies, simulation experiments and running benchmarks. The results of the experiments have been compared with those of previously suggested methods, both in terms of correctness and running times. This part of our work clearly shows that our proposed solutions are efficient and useful in practice.

Since benchmarking collision queries and intersection testing is a difficult endeavour [165], the experimental set-ups have been chosen with care. For example, it is important to make sure that a rich set of geometric configurations are included in the experiments to reveal both the weak and strong aspects of the algorithms, and that known problematic special cases, or any other known limitation of the methods, are stated clearly. Other issues that must be considered are the potential bias of the results, for example, because of limited efforts to optimize any one of the used algorithms, the choice of compiler optimization settings, hardware setup, etc.

Since an awareness of these issues has guided the presented work, it is expected that the reported experiments are reliable and have a high practical value. Of course, theoretical arguments explaining and supporting the experimental results are also very important. In particular, when it can be shown theoretically that a method has the advantage of an asymptotically lower time complexity, and this also can be demonstrated in practice, a strong case can be built.

4.2 Collision Queries for Deforming Models

In deformable-body collision detection, the primary challenge is to deal with the changing shapes of the geometric models efficiently. Clearly, the implications this leads to for the used BVHs depend on the type of deformation the bodies are undergoing. Three main types of deformation are considered here: (i) fixed-connectivity meshes undergoing arbi-

trary vertex repositioning, (ii) meshes deformed by a specific bounded deformation model, such as morphing, and (iii) general deformation, i.e. meshes undergoing arbitrary changes of the mesh connectivity (mesh topology) as well as arbitrary vertex repositioning.

4.2.1 Hierarchy Refitting for Vertex Deformation

In paper **A**, a new collision detection algorithm is proposed for multiple deforming bodies represented by polygon meshes. The bodies are allowed to undergo a complete change of shape from one time step to another by arbitrary vertex movements, but the topology or mesh structure must be kept the same. First, potential collisions are sorted out in a broad phase, which is done by a sweep and prune method [166, 22]. The result is a list of close body pairs that need to be examined more carefully. This is done in a narrow phase, where a hierarchical search is performed for each body pair in the list by using bounding volume hierarchies of axis-aligned boxes. When the bodies deform, the boxes can be refitted very efficiently, and also, the needed BV/BV intersection test between two AABBs is extremely fast.

The bounding volume trees are constructed in a preprocess before simulation time. A simple top-down tree-building strategy is used in which the mesh is recursively split into new tree nodes until only one face remains. During simulation, the structure of the tree is kept fixed. When a body deforms, the axis-aligned bounding boxes in the nodes are refitted according to a scheme referred to as the hybrid bottom-up/top-down update method, which aims at finding an efficient balance between the number of updated tree nodes and the number of tree nodes encountered during collision traversals.

The performance of the proposed collision detection algorithm compares very favourably with other suggested approaches. In the experiments performed, it was found to be four to five times faster than a previously leading method. Deforming meshes of up to tens of thousands of geometric primitives were used in these experiments.

The main contributions of paper **A** are summarized below:

- A new collision detection algorithm is proposed for deforming meshes. All the vertices of the meshes can be arbitrarily deformed at every simulation time step. The algorithm is based on bounding volume hierarchies that can be pre-built and then efficiently updated during simulation.

– A novel hierarchy update method is presented. The upper levels of the hierarchies are updated bottom-up in an incremental way, whereas the lower levels of the hierarchies are updated lazily, as needed, during the collision traversals. This update method is significantly faster than previously suggested methods.

– By examining and comparing the performance of bounding volumes trees with k-ary tree nodes, where $k = 2$, 4, and 8, it was found that $k = 8$ gave slightly better performance in all conducted experiments. A higher value of k gives lower heights of the trees, but increases the work to be performed per node in the collision traversals.

– Two different ways of partitioning the faces of the meshes into the hierarchy nodes during tree construction are proposed. The first operates on the initial shape of the deforming body and the other on the interconnectivity of the faces of the mesh, which remains the same during simulation. Which is the better method depends on the model and how it is deformed.

In paper **C**, the hierarchy refitting scheme presented in paper **A** is further analyzed and evaluated for ray tracing of deforming meshes. In complex and dynamically changing scenes, the reconstruction phase, responsible for updating the data structures as the simulation proceeds, is likely to become a major bottleneck. Furthermore, the ray tracing phase can be parallelized very efficiently, as compared with the reconstruction phase, for which it seems harder to create successful parallel solutions. With the new approach, it is shown that by only updating the upper levels of the pre-built bounding volume hierarchies, the reconstruction phase can be made an order of magnitude faster. The remaining parts of the hierarchies are updated lazily as needed in the ray tracing phase. This approach saves computation and leads to significant speedups in many scenes.

The high performance of the approach has been verified in complex scenes. It has been demonstrated that scenes with hundreds of thousands of deforming and reflective triangles can be ray traced at interactive frame rates. Completely dynamic and interactive scenes are supported, since no a priori information of forthcoming deformations are used.

The main contributions of paper **C** are summarized below:

– The proposed solution extends the set of scenes that can be ray traced at interactive frame rates. It is shown that interactive ray tracing is possible for scenes with complex deforming meshes consisting of hundreds of thousands of geometric primitives.

– By using the hybrid bottom-up/top-down update method from paper **A**, the reconstruction phase runs an order of magnitude faster as compared with using bottom-up refitted hierarchies. This is important since the reconstruction phase risks becoming the bottleneck in complex dynamic scenes.

– The suggested update scheme takes advantage of the fact that the hierarchies do not need to be updated for the occluded parts of the scene, although they may be deforming. This yields significant speedups in the total rendering time of many complex scenes.

4.2.2 Hierarchy Refitting for Specific Deformation

In paper **B**, a new algorithm for the rapid detection of collisions or intersections between morphing models is proposed, which further extends the usage of bounding volume hierarchies for collision detection. The morphing model is a polygonal mesh that is gradually transformed or blended between a set of reference meshes. All the possible deformations are bounded locally by the reference meshes, which make it possible to create a morphing-aware bounding volume hierarchy for each morphing model that can be updated by transforming or blending sets of reference bounding volumes, which correspond to associated parts of the reference meshes.

The hierarchies are built in a preprocess before simulation and then their structures are never changed. Collision queries are performed by dual hierarchy traversals which sort out possible intersections in an efficient way. Whenever an outdated tree node is reached during a traversal, it is updated by an extremely fast $O(1)$ bounding-volume blending operation, given that a fixed number of reference meshes are used. Strictly speaking, the blending operation is $O(m)$ for m reference meshes, but since m is a very small integer, usually $m \leq 4$, it makes sense to regard m as a constant.

In practice, the performance of a full collision query is expected to be sub-linear, as in hierarchical collision detection methods for rigid bodies. Note that only for face pairs that are found to be located closely during

the collision traversals, must the actual blended vertices of those faces be calculated. This means that for rendering, the vertices of the meshes can be blended by the graphics processing unit (GPU), which saves CPU time, and may improve the overall performance.

The expected high performance of the proposed algorithm has been verified by different types of experiments with morphing meshes defined by tens of thousands of triangular polygons. This algorithm for morphing models was found to be significantly faster than the more general collision detection method presented in paper **A**.

The main contributions of paper **B** are summarized below:

- A novel collision detection algorithm for morphing models is suggested. It is based on morphing-aware bounding volume hierarchies. The performance of this algorithm is of the same order as that for hierarchical rigid body collision detection with expected logarithmic query times in most practical cases, and a best case of $O(1)$ when the root BVs are disjoint.

- A simple hierarchy construction method is presented that creates a single hierarchy per morphing model with k-ary tree nodes, which store one bounding volume per reference model per tree node.

- In the experiments, a tree degree of $k = 8$ was found to be a slightly more efficient choice when compared with $k = 2$ and $k = 4$. Note that it is very common in other works on hierarchical collision detection for rigid bodies to suggest binary tree structures.

- A top-down update method is proposed, which is completely integrated with the collision traversals. Only the nodes that are encountered during the dual hierarchy traversals are updated, and a node is updated simply by blending the stored reference bounding volumes.

- It is shown that the proposed method works for bounding volume hierarchies of axis-aligned bounding boxes, k-DOPs, and spheres.

- The proposed collision detection strategy applies generally to other types of bounded deformations for which a bounding volume refit function is known that is independent of the geometric primitives stored in the volume (see later published approaches [85, 39, 167, 168, 169]).

4.2.3 Hierarchy Restructuring for Breakable Models

In paper **D**, an adaptive BVH-based algorithm for highly dynamic collision detection is presented, which generalizes the usage of bounding volume hierarchies to a much broader category of models undergoing non-rigid motion. Clearly, when the topology of the models is changed during simulation due to, for example, melting, explosions, fracture, cutting, or tearing, the culling efficiency of an associated BVH is often decreased drastically.

To cure arising culling inefficiencies in a BVH, two main techniques are employed: (i) BVH restructuring which aims at reconstructing a specific part of a BVH by re-organizing the set of geometric primitives causing the inefficiency, and (ii) BV refitting which recomputes the bounds or extents of a selected set of BVs without any re-partitioning of the geometric primitives.

Clearly, the first technique has a much better potential when it comes to restoring a high culling efficiency of the BVH, However, a high quality re-partitioning of a node with m primitives usually requires an $O(m \log m)$ strategy. Therefore, the restoration may give rise to severe performance bottlenecks. BV refitting schemes, on the other hand, run in worst case $O(m)$ time, but may not always be enough to restore a reasonable culling efficiency. Consequently, there is a fine balance between these two techniques.

The proposed method is able to efficiently balance low-level update work by amortizing structural changes of the dynamic BVHs over time. Lazy evaluation and temporal coherence are exploited to determine both restructuring and refitting. These techniques are urgently needed to maintain query speed during highly dynamic scenarios, where the number of executed low-level operations arising from node splitting operations and refitting parent nodes by BV merging must be kept in balance with the cost of the number of executed low-level intersection tests needed in the dual hierarchy CD traversals. To reflect this, the cost function in Equation 3.1 is extended to also include relevant terms for the primitive operations arising from hierarchy maintenance. In addition, by using these techniques, only active parts of the BVHs are used, updated, and constructed, which means that the storage requirements of the BVH becomes adaptive to the current complexity of the collision queries.

So far, we have only considered CD between two or more different

bodies. Using the BVH for self-intersection detection, also called intra-object CD, is a more challenging problem, since for connected primitives the queries are misled to find lots of pairs with neighbouring geometric primitives, and the detection of these false intersections consumes much of the execution time. Although the proposed method can still be used for this case, this type of query is clearly slower than the inter-object queries. To evaluate the relative performance of our BVH also for intra-object CD, a sorting-based CD method was also designed called SWIPER. This method is an improvement of the well-known sweep-and-prune CD method designed for n-body simulations [166, 22].

The efficiency of the proposed approaches has been demonstrated in scenes with breakable models consisting of hundreds of thousands of independently moving primitives. In the benchmarks, significant speed-ups between 4.4–12.7 were observed for inter-object CD. For intra-object CD, the observed speedups in our benchmark were between 0.3–54 using the proposed dynamic BVH, and between 1.4–13.6 using SWIPER as compared to the classical sweep-and-prune algorithm.

The main contributions of paper **D** are listed below:

- A novel object-space CD algorithm is presented which is applicable for a broad category of deformable bodies including highly dynamic breakable objects. Furthermore, the algorithm can easily be combined with other BVH-based CD approaches to efficiently support queries among a mix of rigid, modestly deformed, and highly deformable models.

- A simple method is proposed to detect BVH degradation and to perform localized reconstruction of currently active parts of the BVHs. Furthermore, the remaining parts of the BVH are updated using an adaptive refitting scheme running in linear time. By exploiting temporal coherence, the updated BVs correspond to the currently active parts of the BVHs.

- To utilize temporal coherence also in the updating of BVHs for deforming meshes with fixed connectivity, the refitting scheme introduced here can be used in place of the hybrid bottom-up/top-down scheme presented in paper **A**.

- Clearly, the mechanisms for detecting BVH degradation and localized reconstruction are general techniques, and not limited to

BVH-based CD. Hence, the method introduced can with very small adjustments be applicable for accelerating other types of queries needed in, for example, view-frustum culling, occlusion culling, picking, and ray tracing. The benefits of the general approach has been confirmed by several other papers, which also utilize incremental update techniques for BVHs for faster CD between fracturing objects [94] and hierarchy restructuring for ray tracing highly dynamic scenes [93, 77].

– An improved sweep-and-prune algorithm called SWIPER is proposed for detecting self-intersections.

4.3 Collision Queries for Rigid Bodies

In rigid-body collision detection, precomputations can be utilized to a much higher degree compared to scenes with shape-changing objects. Thus, the BVH construction phase can involve more sophisticated heuristics to produce higher quality hierarchy structures with tighter fitting bounding volumes, which will lead to improved culling efficiency in search queries.

It has been shown that the tightness of fit of the volumes in the BVHs is essential to avoid explosive growth of the parameter N_v in the cost function in Equation 3.1 in parallel close proximity situations. Certain BV types, such as the sphere and AABB, fail to deliver the required tightness of fit in such cases. Here we will consider two approaches to achieve attractive tight fitting hierarchies: (i) construction of homogeneous BVHs by carefully choosing an appropriate BV type, and (ii) designing heterogeneous BVHs for excellent tightness of fit.

4.3.1 Tight Fitting Hierarchies using Slab Cut Balls

In paper **E**, a BVH based on a novel type of enclosing volume is proposed called the slab cut ball (SCB). In short, an SCB is a sphere that has been cut by two parallel planes (a slab). An example is given in Figure 2.2. The benefits of using this type of BV in the hierarchies are threefold. It provides tight fitting volumes, efficient overlap tests, and low storage requirements. As shown in the paper, this leads to highly efficient collision queries for rigid bodies over a wide range of geometric situations, from bodies well separated to bodies barely touching each

other, and bodies at deep interpenetration. The algorithm is very general, since every input model can be a polygonal soup associated with a rigid body transform, which means it is applicable in a broad range of interactive graphics simulations.

The successful realization of the SCB tree relies on two fundamental algorithms. First, given a point set with n vertices, efficient computation of a tight fitting enclosing SCB is necessary. For this, a deterministic $O(n)$ algorithm is presented which finds both the required tight fitting sphere and a belonging narrow slab. Second, a highly optimized BV-BV intersection test is required. Therefore, a robust conservative overlap test using less than 100 simple arithmetical operations (add, sub, mul), and zero divisions, is also presented. The purpose of using a conservative test rather than an exact version is to speed up the entire hierarchical collision query [170]. Besides these fundamental algorithms, an appropriate and robust tree building heuristic must be employed. For this, a traditional $O(n \log n)$ top-down construction method is presented, which uses a very efficient partitioning method inspired by quicksort [68].

The collision query between two polygonal soups is carried out by traversing the corresponding SCB trees simultaneously according to the pseudocode given in Figure 3.1. The proposed algorithm is very fast, robust and numerically stable. No divisions, square roots, or other potentially problematic mathematical functions are used in the computations, and unlike the OBBTree method [47], the traversal does not use cascading multiplication of transformation matrices during the recursion.

Interestingly, the OBB tree has been shown to have quadratic convergence to underlying smoothly curved geometry, whereas sphere and AABB trees only have linear convergence [23]. This gives a significant performance advantage for OBBs in certain close proximity situations such as those arising when fitting machine parts together in a virtual assembly simulation. Since the arguments for the quadratic OBB convergence, which concerns the relative change of diameter and width of the BVs as a BVH is descended, also applies to the SCB volume, a similar performance advantage is also expected for SCB trees. Indeed, the experimental results obtained confirm this. In parallel close proximity situations, the SCB tree is asymptotically faster than the sphere tree.

The SCB tree solution can also be regarded as an improvement over the seminal OBB tree data structure, since it requires less storage and it involves a faster BV-BV overlap test. Interestingly, from a theoretical point of view, optimal SCBs can also be computed asymptotically faster

than minimum OBBs. Therefore, building an OBB tree with minimum OBBs is a more complex operation than building a SCB tree with optimal SCBs.[1] As shown by O'Rourke, the computation of a minimum OBB can be made in $O(n^3)$ time [171], and it appears to be the asymptotically fastest method to date [134]. Therefore, OBB tree construction would in this case be $O(n^3 \log n)$. Using Gottschalk's practical approach for OBB fitting, however, results in an $O(n \log^2 n)$ tree-building method, given that the convex hull is computed in the OBB creation process, and otherwise $O(n \log n)$ [23].

On the other hand, an enclosing SCB volume can be computed in $O(n^{3/2+\varepsilon})$ time using a minimum width slab algorithm [172]. In this case, the computation of the slab in the SCB computation would dominate the running time, since minimum spheres can be computed in deterministic linear time. Accordingly, the construction of an SCB tree would take $O(n^{\frac{3}{2}+\varepsilon} \log n)$ time. Computing the SCB tree using our practical method, however, takes $O(n \log n)$ time using our worst case $O(n)$ time SCB fitting algorithm. In the asymptotical running times given here, it has been assumed that the used tree-building method manages to partition the primitives in each hierarchy level in linear time, and that the produced trees have a logarithmic depth.

To summarize, the main contributions of paper **E** are as follows:

- The slab cut ball is introduced as a novel type of BV providing an attractive balance between the computational cost of the overlap test, tightness of fit, and storage cost.
- An efficient worst case $O(n)$ algorithm for computing an enclosing SCB of a point set is given.
- A practical and efficient top-down BVH construction method is proposed with expected $O(n \log n)$ running time.
- A fast overlap test between two SCBs for use in collision detection traversals using less than 100 simple arithmetical operations is given.
- The experimental evaluation of the performance of the SCB tree for collision queries between pairs of polygonal models shows a

[1] By optimal, we here mean an SCB created as the intersection of two independently determined volumes, i.e. the smallest bounding sphere and minimum width slab.

significant performance advantage for SCB trees over sphere trees. In addition, the SCB trees are also faster than the OBB trees in the benchmarks.

4.3.2 Heterogeneous Bounding Volume Hierarchies

Since the tightness of fit of the volumes in a bounding volume hierarchy can be dramatically improved by switching to heterogeneous or hybrid bounding volume hierarchies, the development of high-quality automatic construction methods of such tight hierarchies looks attractive. Unfortunately, heterogeneous bounding volume hierarchies is an understudied subject [173]. An early initial effort in ray tracing utilized a mix of spheres, cylinders, and boxes as BV types [35]. Su et al. stored a list of candidate bounding volumes in each non-leaf tree node.[2] Then during collision detection between two tree nodes, they adaptively chose the type of intersection test to perform by selecting the BV-BV pair which minimized a cost function [174]. In another approach, three different types of sphere swept volumes (point swept sphere, line swept sphere and rectangle swept sphere) are used as candidate volumes in the BVH construction [61].

A powerful set of BV types to be used in hybrid BVHs could for example include the AABB, OBB, sphere, and ellipsoid. For all these BV types, practical $O(n)$ methods exist to compute almost optimal enclosing volumes. Obviously, by computing all these types of volumes for a given subset of points during tree construction, the best shape can be chosen and stored, while the other shapes are discarded. To determine the best shape, a quality metric is needed, which is based on the size of the volume (or area) in relation to the expected operation cost of the required geometric operations.

The remaining problem, and by far the most difficult one, is the construction of a high quality BVH structure with respect to all of the possible bounding volume types. The design goal would be to find some reasonably fast and useful heuristic producing significantly tighter fitting hierarchies than corresponding homogenous BVHs.

Efficient ray intersection tests are available for all the above mentioned BV types [145, 53]. For collision detection, however, the number of overlap tests needed grows quadratically with the number of BV

[2] They mentioned AABB, OBB, cylinder, sphere, cone, and wedge as example candidate BV types.

types. More specifically, if there are n BV types, the number of overlap test routines, N_o, that have to be supported is

$$N_o = n + \frac{n(n-1)}{2} = \frac{n(n+1)}{2}. \tag{4.1}$$

Thus, by for example limiting the number of BV types to the four previously mentioned shapes, it would be sufficient to implement 10 different overlap tests. In this case, the following tests are needed: AABB-AABB, OBB-OBB, sphere-sphere, ellipsoid-ellipsoid, AABB-OBB, AABB-sphere, AABB-ellipsoid, OBB-sphere, OBB-ellipsoid, and sphere-ellipsoid.

Implementing the overlap tests AABB-AABB and sphere-sphere efficiently is straightforward. Efficient methods for the overlap tests OBB-OBB and ellipsoid-ellipsoid are also well known [47, 133]. For the AABB-OBB overlap test, a simpler version of the OBB-OBB overlap test can be utilized. For the sphere-ellipsoid overlap test, the ellipsoid-ellipsoid test is applicable, since a sphere is a special case of an ellipsoid. Alternatively, an iterative search method can be used to find the minimal distance from the sphere centre to the ellipsoid, which is then compared to the sphere radius [134]. Efficient algorithms for the remaining cases, which have been developed in papers **F** and **G**, are presented next.

4.3.3 Sphere-Box Overlap Testing

In paper **F**, faster algorithms for the AABB-sphere and OBB-sphere overlap tests are presented, which makes it more attractive to use a mix of spheres, AABBs, and OBBs as bounding containers to speed up BVH-based CD methods. As pointed out by Arvo, the distance from a point $\mathbf{p}$ on the box to the centre $\mathbf{c}$ of the sphere is given by

$$d(\mathbf{p}) = \sqrt{(p_x - c_x)^2 + (p_y - c_y)^2 + (p_z - c_z)^2}. \tag{4.2}$$

By finding the point that minimizes the distance d, we can compute d^2 and compare it to the squared radius of the sphere to get the overlap status. It is, however, sufficient to minimize each term under the root expression independently, since each term is non-negative. This observation leads to Arvo's elegant AABB-sphere intersection test algorithm [175].

In paper **F**, it is shown that Arvo's method can be made faster by adding early outs at appropriate places in the algorithm. The idea behind the early outs can be described geometrically as follows: If the

bounds of the box are enlarged by the radius of the sphere along each one of its principal axes, no intersection can possibly occur, if the centre of the sphere is outside this enlarged box. Note that this type of quick rejection can be inserted in the algorithm as the original distance calculations proceed, simply by adding a single conditional test per box axis, without increasing the number of required arithmetic operations at all. Alternatively, all the early outs can be executed first of all in the beginning of the algorithm.

Finally, it is also shown that an implementation of Arvo's method based on a vectorized instruction set, such as Intel's SIMD SSE, gives opportunities for both branch and loop elimination, which results in even faster execution time compared to the tested sequential methods.

The main contributions of paper **F** are summarized below:

- Arvo's original sphere-AABB intersection test is improved by adding early outs based on a simple geometric condition at appropriate places in the algorithm.
- An efficient generalization of the algorithm for the sphere-OBB case is also presented.
- Even faster vectorized versions of the sphere-AABB and sphere-OBB algorithms are presented, which eliminates branches as well as the entire loop over the axes of the box.
- It is shown that simple conservative overlap tests lead to even faster execution times without introducing too many false positives. This type of test may improve performance further when used in conjunction with hierarchical CD queries, which refine the search results as needed by descending to more fine-grained levels.
- The algorithms work for spheres and boxes in n-dimensional space, although the presentation only focuses on the three-dimensional case.

4.3.4 Ellipsoid-Box Overlap Testing

In paper **G**, a novel algorithm for the ellipsoid–OBB overlap test is proposed.[3] This intersection test makes it more attractive to use a mix of

[3]It is straightforward to use the same test method for the ellipsoid-AABB test too.

ellipsoids and OBBs as bounding boxes in a heterogeneous BVH. The main idea is to transform the three-dimensional space in such a way that the ellipsoid deforms into the unit sphere, which at the same time deforms the OBB to a parallelepiped. This is possible since, as shown in the paper, this type of affine transformation is guaranteed to leave the intersection status of the objects unaltered. After the transformation, the calculation proceeds in the obtained simplified test space, which means the overlap status can be determined rapidly by only using a constant number of simple arithmetical operations and comparisons. Also, the overlap test takes advantage of the regular shape of the parallelepiped to eliminate unnecessary test cases. In addition, a simple early rejection test is proposed. All these techniques are expected to speed up the overlap test significantly, which is also confirmed by the practical experiments.

The main contributions of paper **G** are summarized below:

- A fast intersection test between arbitrarily oriented ellipsoids and boxes is proposed.
- In particular, the algorithm can be used as a building block in a future realization of a CD system using heterogeneous BVHs.
- The general idea of transforming the geometric objects to the canonical sphere space of the ellipsoid is clearly applicable for designing other rapid intersection tests involving ellipsoids, such as ellipsoid-polygon and ellipsoid-polyhedron overlap tests.

Chapter 5

Conclusions

In this dissertation, various types of adaptive bounding volume hierarchies have been shown to be powerful tools to accelerate different types of collision queries. The involved models have been polygonal meshes or polygonal soups, and three main types of deformation have been considered for these models: (i) arbitrary vertex re-positioning for meshes with static topology, (ii) specific bounded deformation which makes possible sub-linear refitting of a node's BV independently of where in the tree structure it is located, and (iii) independently deforming polygon soups undergoing unstructured relative motion. Indeed, these deformation models cover a very broad set of geometric scenes that could be expected to appear in interactive graphics simulation. Thus, the proposed methods for efficient collision queries are quite generally applicable.

Since the running times of the proposed algorithms have been in focus, it is also worth noting that the improvements of the performance of CPUs, from year to year, will not be great enough to make advanced collision detection algorithms obsolete. In computer graphics, there is a never-ending demand for larger and more complex simulations. In many respects, the solutions and algorithms proposed here scale well with scene complexity, and they may therefore be useful for simulation applications for many years to come.

Although the computer graphics field has been in focus in this work, it is also expected that the proposed approaches can be used in applications, or inspire improved methods, in other related fields such as robotics, computational geometry, computational biology, and medical

simulation.

Generalization of the proposed algorithms to work also for tetrahedral meshes is straightforward. Curved surfaces can also be supported easily by tessellation to polygons. However, this gives only an approximation to the real surfaces. If higher accuracy is needed, some minor additions such as supporting a fast overlap test between pairs of surface patches and computation of bounding volumes covering these patches, should suffice to generalize the hierarchical approaches to work also for models with surface patches as geometric primitives.

The collision detection schemes proposed in papers **A**, **B**, **D**, and **E** all have in common that the focus has been on performing efficient collision queries between two models. If a hundred, or more, models are included in a simulation, an efficient broad phase must be added to these collision detection systems, whose purpose is to sort out all body pairs out of n bodies that are potentially capable of colliding [176]. The sweep-and-prune method for moving body simulation seems suitable for this [126, 22]. For all three types of deformable models considered here, and for the rigid body case as well, all that needs to be done to apply this method is to first update or compute an AABB for the root node in all the hierarchies. For example, in the case of morphing, the root AABB is simply computed by blending the reference bounding volumes they store, an extremely efficient operation. The computation of an AABB covering an SCB can also be implemented as a simple constant time operation. Besides the sweep-and-prune approach, uniform grid and spatial hashing approaches are also good alternative data structures for accelerating broad phase collision detection [177, 178, 19, 119].

The ultimate goal would of course be to design a single adaptive bounding volume hierarchy capable of supporting all the above mentioned deformation models (and others) at the same time in an efficient way. Interestingly, it can be noted that all four collision detection approaches, given in papers **A**, **B**, **D**, and **E**, are based on quite similar bounding volume hierarchies and hierarchy traversals. Therefore, they can easily be integrated into a very general collision query framework to support simulations that include all three types of deformation models.

To realize such a framework, a specialized dual tree collision detection traversal must be implemented for each possible type of hierarchy pair. The only two additional intersection tests that must be supported between tree nodes to integrate our methods are the AABB-SCB and OBB-SCB tests. Then collisions between all combinations of the differ-

ent types of the deformable models mentioned above, as well as rigid models, can be efficiently tracked.

Other types of bounding volume trees can also be supported, given that efficient intersection tests between tree nodes can be provided. Therefore, a natural extension of the framework would be to integrate OBB trees and sphere trees, primarily as an alternative way of representing rigid models. By permitting the use of different types of homogenous hierarchies, it becomes possible to choose the most suitable BVH type for each model, which can improve the tightness of fit and lead to better performance.

The flexibility within a single BVH can be increased further by switching to heterogeneous bounding volume hierarchies. The overlap tests given in papers **F** and **G** add to the collection of required overlap tests to make collision queries using heterogeneous bounding volume trees possible. It is here conjectured that this type of BVH will improve rigid body collision detection significantly, as compared to corresponding homogenous BVHs. However, this requires high-quality heuristics for building superior tight-fitting hierarchies.

Given two heterogeneous bounding volume trees representing two rigid models, an efficient dual hierarchy collision detection traversal, similarly to the algorithm given in Figure 3.1, can for example be realized by using the double dispatch technique [179] in a programming language such as C++.[1] In this way, the correct type of overlap test can be called for each encountered node pair without adding branches in the code to handle the different BV types during the traversal.

Paper **C** illustrates the usage of the proposed data structures for a different type of collision query, i.e. between rays of light and the geometric primitives of models. Although no experimental results have been obtained for the other types of hierarchies (proposed in papers **B**, **D**, and **E**) in the context of ray tracing, it is expected that these types of hierarchies may be utilized for efficient ray tracing as well. Experiments are of course needed to evaluate the data structures and for designing improved versions that are found to be more suitable for ray tracing. This is of great importance for extending the set of scenes which can be used in interactive ray tracing.

[1] C++ supports single dispatch natively, i.e. dynamic binding of a message based on the type of the receiver object.

5.1 Future Work

Much further work remains to be done in the fields of spatial data structures and rigid and deformable body simulation. Certain directions in which research relating to collision detection and interactive ray tracing is desirable are described briefly below:

- The extension of the proposed collision detection methods to support efficient handling of self-intersections would be beneficial. This is very important in certain applications. In cloth simulation, for example, some interesting approaches have already been published [180, 181, 182].

- The proposed collision detection methods are not designed to handle situations where a single model is torn, cut, or broken, into several widely separated parts. In such cases, the BVH of the model probably needs to be broken into several independent BVHs as well, which corresponds to the arising independent sub-models. Therefore, further generalization of the algorithms to support breakable bodies more efficiently is a subject worth more attention.

- In the field of bioinformatics and computational biology new specialized collision detection algorithms are needed. The simulation of protein folding, for example, is becoming increasingly more important, but also computationally very challenging. Deforming molecules are systems with many degrees of freedom and efficient proximity detection is crucial. Therefore, the development of new data structures and algorithms that are well suited for the simulation of protein folding is needed. An example of an interesting recent proposal is a graph data structure called deformable spanner [183].

- The design of a set of standardized benchmark scenes for collision detection of deformable bodies would make fair comparisons between algorithms much easier. Such scenes must be designed with care so that they include a broad spectrum of different and interesting contact scenarios. So far, only a few initial efforts to propose suitable standard benchmarks have been made [165].

- It has been shown that bounding volume hierarchies can be used for interactive ray tracing [153]. However, new improved methods for

reconstruction of bounding volume hierarchies are still needed to make interactive rendering of highly complex and dynamic scenes using BVHs a reachable goal [45, 93]. For example, it would be interesting to study acceleration data structures for scenes with millions of independently deforming primitives, including parallelization of the construction and/or reconstruction phase. Several recent papers present promising results in this direction [157, 20].

– For rigid bodies, a technique designated generalized front tracking has been suggested to utilize the temporal coherence that is present in many types of simulations [114, 110]. It would perhaps be interesting to examine this technique further in the context of deformable body simulation. When using a node degree larger than two, however, which is often the case when BVHs are used to represent deformable objects, the hierarchies become flatter, which decreases the potential benefits of using front tracking [51, 45].

– Bounding volume hierarchies of SCBs have been successfully used for efficient CD [67]. It would also be worthwhile to study the effects of using the same, or similar, hierarchies to accelerate other types of geometric queries such as view-frustum culling and ray tracing. By also considering methods to update the hierarchies, e.g. by deleting and inserting geometric primitives, as well as supporting schemes for efficient volume refitting and lazy hierarchy restructuring, efficient queries for deformable objects may be developed as well [45, 94].

– The SCB tree seems to be particularly suitable also for time-critical CD [25]. In this case, approximate collision response can be computed based on the encountered uncut sphere in each node as in existing sphere tree approaches [106], or more interestingly, the quite simple shape of the SCB itself can be used to design a more accurate approximate collision response mechanism. Furthermore, since significantly fewer BVs might be needed to reach a certain tightness of fit compared to ball trees and AABB trees, this opens promising opportunities for both time and memory critical BVH solutions.

– It would be interesting to see how fast an exact SCB-SCB overlap test can be made. Replacing the proposed conservative overlap

test with an exact intersection test in collision queries using SCB trees would improve the culling efficiency, but how would it affect the overall performance?

– It is well known that the smallest enclosing ball can be constructed in $O(n)$ time [56], and that the minimum width slab can be computed in $O(n^{\frac{3}{2}+\varepsilon})$ time [172]. However, when considering the SCB volume, which is the intersection volume of a ball and a slab, the question arises how we can compute a minimum volume SCB. It is clear that the intersection volume between the minimum width slab and the minimum volume ball does not give the minimum SCB in the general case. How to best compute the smallest volume SCB enclosing a set of n points is therefore an interesting open problem. A related problem is how to find the minimum SCB enclosing a set of n balls. Furthermore, an algorithm to compute the smallest SCB given n SCBs as input would be interesting in e.g. bottom-up construction of SCB trees. Clearly, practical algorithms for computing the minimum volume SCB in various contexts would increase the attractiveness of using the SCB as a bounding container.

– It is expected that the development of an improved construction method for SCB hierarchies would lead to significantly tighter fitting levels of approximations. For example, by finding "islands" of rather flat areas of a model in the construction process, the polygons in such an area can be grouped under a node and enclosed by a tight-fitting SCB.

– How to create high-quality heterogeneous bounding volume hierarchies is an understudied subject. In general, this type of BVH would significantly improve the tightness of fit of the approximating levels in the tree, and thus, potentially lead to a major breakthrough for BVH-accelerated spatial queries.

– The majority of the collision detection algorithms focus on polygon meshes or soups. Generalizing the presented CD methods to also handle other representations of the models efficiently, for example, point clouds [184], tetrahedral meshes, and higher order surfaces, would be possible. Besides making the methods more generally applicable, this may also lead to new general insights on BVH-based collision detection.

- With the advent of multi-core CPUs and modern programmable GPUs, the development of new parallel CD methods, based on BVHs, grids [118], or any other suitable data structure seems very attractive.

- Currently, the architectures used in graphics hardware are evolving rapidly. The graphics processing unit (GPU) is more and more being turned into a highly parallel general computation unit. This means that the traditional clear distinction between CPUs and GPUs is getting blurred [185]. Consequently, using the GPU to speed up tasks such as global illumination, collision detection, physics, and artificial intelligence is getting more attractive, and this is a research area that definitely needs more attention [186].

Hopefully, the contributions put forward in this thesis have provided some interesting answers to the question: How can efficient collision queries using adaptive bounding volume hierarchies be realized? Indeed, collision detection is a broad and exciting area, with many opportunities for stimulating research and further progress. Now, when the introduction of new massively parallel architectures for desktop computers lies right before us, the future for improved geometric queries in collision detection and rendering looks very promising!

Bibliography

[1] Computing curricula 2001. *Journal on Educational Resources in Computing (JERIC)*, 1(3), 2001.

[2] Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering, 2nd Edition*. A K Peters, 2002.

[3] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[4] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH '80: Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques*, pages 124–133, 1980.

[5] Bruce Naylor, John Amanatides, and William Thibault. Merging BSP trees yields polyhedral set operations. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, pages 115–124. ACM Press, 1990.

[6] Jr. George Vaněček. Brep-index: a multidimensional space partitioning tree. In *SMA '91: Proceedings of the First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 35–44, New York, NY, USA, 1991.

[7] W. Bouma and Jr. G. Vanecek. Collision detection and analysis in a physical based simulation. In *Eurographics Workshop on Animation and Simulation*, pages 191–203, 1991.

[8] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

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

[10] Raphael Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4(1), 1974.

[11] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[12] Hanan Samet and Robert E. Webber. Hierarchical data structures and algorithms for computer graphics. Part I: Fundamentals. *IEEE Computer Graphics and Applications*, 8(3):48–68, 1988.

[13] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications, 2nd Edition*. Springer-Verlag, 1997.

[14] Hanan Samet and Robert E. Webber. Hierarchical data structures and algorithms for computer graphics. Part II: Applications. *IEEE Computer Graphics and Applications*, 8(4):59–75, 1988.

[15] K. Klimaszewski and T. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, 17:42–51, 1997.

[16] Thiago Ize, Peter Shirley, and Steven G. Parker. Grid creation strategies for efficient ray tracing. In *IEEE Symposium on Interactive Ray Tracing*, pages 27–32, 2007.

[17] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: Accelerated raytracing system. *IEEE Computer Graphics and Applications*, 6:16–26, 1986.

[18] V. Akman, W. R. Franklin, M. Kankanhalli, and C. Narayanaswmi. Geometric computing and uniform grid technique. *Computer Aided Design*, 21(7):410–420, 1989.

[19] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of Vision, Modeling and Visualization*, pages 47–54, November 2003.

[20] Ares Lagae and Philip Dutré. Compact, fast and robust grids for ray tracing. In *Eurographics Symposium on Rendering*, 2008.

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

[22] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav Ponamgi. I-COLLIDE: an interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 189–196. ACM Press, 1995.

[23] Stefan Gottschalk. *Collision Queries using Oriented Bounding Boxes*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 2000.

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

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

[26] Daniel S. Coming and Oliver G. Staadt. Velocity-aligned discrete oriented polytopes for dynamic collision detection. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):1–12, 2008.

[27] Jen-Duo Liu, Ming-Tat Ko, and Reui-Chuan Chang. Collision avoidance in cloth animation. *The Visual Computer*, 12:234–243, 1996.

[28] Ming C. Lin and Dinesh Manocha. Collision detection. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 787–807. Chapman & Hall, 2004.

[29] Liangjun Zhang, Xin Huang, Young J. Kim, and Dinesh Manocha. D-plan: Efficient collision-free path computation for part removal and disassembly. *Computer Aided Design and Applications*, 5(6):774–786, 2008.

[30] An Nguyen. *Implicit Bounding Volumes and Bounding Volume Hierarchies*. PhD thesis, Stanford University, 2006.

[31] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 2nd Edition*. MIT Press, 2000.

[32] Gill Barequet, Bernard Chazelle, Leonidas J. Guibas, Joseph S. B. Mitchell, and Ayellet Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum*, 15(3):387–396, August 1996.

[33] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Yannis Theodoridis. R-trees have grown everywhere. Technical report, 2003.

[34] Lars Arge, Mark de Berg, and Herman Haverkort. Cache-oblivious R-trees. In *SCG '05: Proceedings of the Twenty-first Annual Symposium on Computational Geometry*, pages 170–179, New York, NY, USA, 2005. ACM.

[35] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Transanctions on Graphics*, 3(1):52–69, 1984.

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

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

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

[39] Ladislav Kavan and Jiri Zara. Fast collision detection for skeletally deformable models. *Computer Graphics Forum*, 24(3):363–372, 2005.

[40] Cesar Mendoza and Carol O'Sullivan. Interruptible collision detection for deformable objects. *Computers & Graphics*, 30(2):432–438, 2006.

[41] Thanh Giang and Carol O'Sullivan. Approximate collision response using closest feature maps. *Computers & Graphics*, 30(2):423–431, 2006.

[42] Robert Webb and Mike Gigante. Using dynamic bounding volume hierarchies to improve efficiency of rigid body simulations. In *Proceedings of the 10th International Conference of the Computer Graphics Society on Visual computing: Integrating Computer Graphics with Computer Vision*, pages 825–842. Springer-Verlag New York, Inc., 1992.

[43] Gino van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *journal of graphics tools*, 2(4):1–14, 1997.

[44] Thomas Larsson and Tomas Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics Conference 2001, Short Presentations*, pages 325–333, September 2001.

[45] Thomas Larsson and Tomas Akenine-Möller. A dynamic bounding volume hierarchy for generalized collision detection. *Computers & Graphics*, 30(2):451–460, 2006.

[46] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics*, 26(1), 2007.

[47] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 171–180, 1996.

[48] S. Redon, A. Kheddary, and S. Coquillart. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, 21(3):279–288, September 2002.

[49] 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.

[50] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 90–97, March 1998.

[51] J. Mezger, S. Kimmerle, and O. Etzmuss. Hierarchical techniques in collision detection for cloth animation. *Journal of WSCG*, 11:322–329, 2003.

[52] Christer Ericson. *Real-Time Collision Detection*. Morgan Kaufmann, 2005.

[53] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering, 3rd Edition*. A K Peters, 2008.

[54] Thomas Larsson. Fast and tight fitting bounding spheres. In *Proceedings of The Annual SIGRAD Conference*, pages 27–30. Linköping University Electronic Press, November 2008.

[55] Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12:759–776, 1983.

[56] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and Trends in Computer Science, Lecture Notes in Computer Science 555*, pages 359–370. Springer, 1991.

[57] Bernd Gärtner. Fast and robust smallest enclosing balls. In *ESA '99: Proceedings of the 7th Annual European Symposium on Algorithms*, pages 325–338. Springer-Verlag, 1999.

[58] Yi-King Choi, Jung-Woo Chang, Wenping Wang, Myung-Soo Kim, and Gershon Elber. Real-time continuous collision detection for moving ellipsoids under affine deformation. Technical report, HKU CS Tech Report TR-2006-02, 2006.

[59] Lin Lu, Yi-King Choi, Wenping Wang, and Myung-Soo Kim. Variational 3D shape segmentation for bounding volume computation. *Computer Graphics Forum*, 26(3):329–338, 2007.

[60] Sergey Bereg. Cylindrical hierarchy for deforming necklaces. *International Journal of Computational Geometry & Applications*, 14(1-2):3–17, 2004.

[61] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast proximity queries with swept sphere volumes. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill, 1999.

[62] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast distance queries with rectangular swept sphere volumes. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3719–3726, 2000.

[63] Taosong He. Fast collision detection using QuOSPO trees. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 55–62, 1999.

[64] Norio Katayama and Shin'ichi Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 369–380, 1997.

[65] Shankar Krishnan, Amol Pattekar, Ming C. Lin, and Dinesh Manocha. Spherical shell: a higher order bounding volume for fast proximity queries. In *WAFR '98: Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics: the Algorithmic Perspective*, pages 177–190, 1998.

[66] Leonidas J. Guibas, An Nguyen, and Li Zhang. Zonotopes as bounding volumes. In *SODA '03: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 803–812, 2003.

[67] Thomas Larsson and Tomas Akenine-Möller. Bounding volume hierarchies of slab cut balls. Technical report, Mälardalen University, June 2008.

[68] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–15, 1962.

[69] Bruce Walter, Kavita Bala, Milind Kulkarni, and Keshav Pingali. Fast agglomerative clustering for rendering. In *IEEE Symposium on Interactive Ray Tracing*, 2008.

[70] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):101–108, 2005.

[71] Jiri Skala and Ivana Kolingerova. Clustering geometric data streams. In *Proceedings of The Annual SIGRAD Conference*, pages 17–23. Linköping University Electronic Press, November 2007.

[72] Kerawit Somchaipeng, Kenny Erleben, and Jon Sporring. A multi-scale singularity bounding volume hierarchy. In *Proceedings of International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 179–186, 2005.

[73] Kerawit Somchaipeng. *Multi-Scale Singularity Trees*. PhD thesis, The Department of Computer Science, University of Copenhagen, Denmark, 2006.

[74] Stephen M. Omohundro. Five balltree construction algorithms. Technical Report 89-063, International Computer Science Institute, Berkeley, California, November 1989.

[75] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[76] Jörg Haber, Marc Stamminger, and Hans-Peter Seidel. Enhanced automatic creation of multi-purpose object hierarchies. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, pages 52–61, 2000.

[77] Martin Eisemann, Thorsten Grosch, Marcus Magnor, and Stefan Müller. Automatic creation of object hierarchies for ray tracing of dynamic scenes. In *WSCG Short Papers Proceedings*, 2007.

[78] G. Adelson-Velskii and E. M. Landis. An algorithm for the organization of information. In *Proceedings of the USSR Academy of Sciences*, pages 263–266, 1962.

[79] Michael T. Goodrich and Roberto Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples*. John Wiley & Sons, Inc., 2002.

[80] Leo J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science*, pages 8–21, October 1978.

[81] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the Association for Computing Machinery*, 32(3):652–686, 1985.

[82] Manfred Ernst and Gunther Greiner. Early split clipping for bounding volume hierarchies. In *IEEE Symposium on Interactive Ray Tracing*, pages 73–78, 2007.

[83] Pankaj Agarwal, Leonidas Guibas, An Nguyen, Daniel Russel, and Li Zhang. Collision detection for deforming necklaces. *Computational Geometry Theory and Applications*, 28(2-3):137–163, 2004.

[84] J. Brown, S. Sorkin, C. Bruyns, and J. Latombe. Real-time simulation of deformable objects: Tools and application. In *Proceedings of Computer Animation*, November 2001.

[85] Doug L. James and Dinesh K. Pai. BD-tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics*, 23(3):393–398, 2004.

[86] J. Mahovsky and B. Wyvill. Memory-conserving bounding volume hierarchies with coherent ray tracing. *Computer Graphics Forum*, 25(2):173–182, 2006.

[87] David Cline, Kevin Steele, and Parris Egbert. Lightweight bounding volumes for ray tracing. *journal of graphics tools*, 11(4):61–71, 2006.

[88] Sung-Eui Yoon and Dinesh Manocha. Cache-efficient layouts of bounding volume hierarchies. *Computer Graphics Forum*, 25(3):507–516, 2006.

[89] Herman J. Haverkort, Mark de Berg, and Joachim Gudmundsson. Box-trees for collision checking in industrial installations. In *SCG '02: Proceedings of the Eighteenth Annual Symposium on Computational geometry*, pages 53–62, 2002.

[90] Paul S. Strauss and Rikk Carey. An object-oriented 3D graphics toolkit. In *SIGGRAPH '92: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, pages 341–349. ACM Press, 1992.

[91] John Rohlf and James Helman. IRIS performer: a high performance multiprocessing toolkit for real-time 3D graphics. In *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 381–394. ACM Press, 1994.

[92] Thomas Larsson and Tomas Akenine-Möller. Efficient collision detection for models deformed by morphing. *The Visual Computer*, 19:164–174, 2003.

[93] Sung-Eui Yoon, Sean Curtis, and Dinesh Manocha. Ray tracing dynamic scenes using selective restructuring. In *Eurographics Symposium on Rendering*, 2007.

[94] Otaduy, O. Chassot, D. Steinemann, and M. Gross. Balanced hierarchies for collision detection between fracturing objects. In *IEEE Virtual Reality Conference*, pages 83–90, 2007.

[95] Sigal Ar, Bernard Chazelle, and Ayellet Tal. Self-customized BSP trees for collision detection. *Computational Geomtery: Theory and Applications*, 15(1-3):91–102, 2000.

[96] Sigal Ar, Gil Montag, and Ayellet Tal. Deferred, self-organizing BSP trees. *Computer Graphics Forum*, 21(3):269–278, 2002.

[97] T. Ulrich. Loose octress. In Mark DeLoura, editor, *Game Programming Gems*, pages 444–453. Charles River Media, 2000.

[98] M.C. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proceedings of IMA, Conference of Mathematics of Surfaces*, pages 602–608, 1998.

[99] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: a survey. *Computers & Graphics*, 25:269–285, 2001.

[100] Carol O'Sullivan, John Dingliana, Fabio Ganovelli, and Gareth Bradshaw. T6: Collision handling for virtual environments. In *Eurographics 2001 Tutorial Proceedings*, 2001.

[101] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1):61–81, 2005.

[102] Zhigeng Pan, Weiwei Xu, Jin Huang, Mingmin Zhang, and Jiaoying Shi. Easybowling: a small bowling machine based on virtual simulation. *Computers & Graphics*, 27:231–238, 2003.

[103] Kristian Yrjölä and Thomas Larsson. Real-time generation of plausible surface waves. In *Proceedings of The Annual SIGRAD Conference*, pages 11–16. Linköping University Electronic Press, November 2007.

[104] E. Arbabi, R. Boulic, and D. Thalmann. A fast method for finding range of motion in the human joints. In *Engineering in Medicine and Biology Society*, pages 5079–5082, 2007.

[105] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, pages 289–298. ACM Press, 1988.

[106] John Dingliana and Carol O'Sullivan. Graceful degradation of collision handling in physically based animation. *Computer Graphics Forum*, 19(3):239–248, 2000.

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

[108] S. Cameron. Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proceedings of the International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[109] Brian Mirtich. V-clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics (TOG)*, 17(3):177–208, 1998.

[110] Stephan A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum*, 20(3):500–510, September 2001.

[111] G. Zachmann and W. Felger. The boxtree: Enabling real-time and exact collision detection of arbitrary polyhedra. In *First Workshop on Simulation and Interaction in Virtual Environments*, pages 104–113, 1995.

[112] Juan Jose Jimenez, Francisco R. Feito, Rafael J. Segura, and Carlos J. Ogayar. Particle oriented collision detection using simplicial coverings and tetra-trees. *Computer Graphics Forum*, 25(1):53–68, 2006.

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

[114] Tsai-Yen Li and Jin-Shin Chen. Incremental 3D collision detection with hierarchical data structures. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 139–144. ACM Press, November 1998.

[115] Gabriel Zachmann and René Weller. Kinetic bounding volume hierarchies for deformable objects. In *ACM International Conference on Virtual Reality Continuum and Its Applications (VRCIA)*, Hong Kong, China, June 2006.

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

[117] F. Ganovelli, J. Dingliana, and C. O'Sullivan. Buckettree: Improving collision detection between deformable objects. In *Spring Conference in Computer Graphics (SCCG2000)*, pages 156–163, 2000.

[118] W. R. Franklin, N. Chandrasekhar, M. Kankanhalli, M. Seshan, and V. Akman. Efficiency of uniform grids for intersection detection on serial and parallel machines. In *New Trends in Computer Graphics (Proc. Computer Graphics International)*, pages 27–32, 1988.

[119] Mathias Eitz and Gu Lixu. Hierarchical spatial hashing for real-time collision detection. In *SMI '07: Proceedings of the IEEE International Conference on Shape Modeling and Applications*, pages 61–70. IEEE Computer Society, 2007.

[120] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 25–32. Eurographics Association, 2003.

[121] Naga K. Govindaraju, Ming C. Lin, and Dinesh Manocha. Quick-cullide: Fast inter- and intra-object collision culling using graphics hardware. In *IEEE VR*, 2005.

[122] Alexander Gress, Michael Guthe, and Reinhard Klein. GPU-based collision detection for deformable parameterized surfaces. *Computer Graphics Forum*, 25(3):497–506, 2006.

[123] Naga K. Govindaraju, Ming C. Lin, and Dinesh Manocha. Fast and reliable collision culling using graphics hardware. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):143–154, 2006.

[124] Brian Von Herzen, Alan H. Barr, and Harold R. Zatz. Geometric collisions for time-dependent parametric surfaces. In *Proceedings of the 17th Annual Conference on Computer graphics and Interactive Techniques*, pages 39–48. ACM Press, 1990.

[125] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. In *Proceedings of the 17th Annual Conference on Computer graphics and Interactive Techniques*, pages 19–28. ACM Press, 1990.

[126] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, pages 303–308. ACM Press, 1992.

[127] Pascal Volino, Martin Courchesne, and Nadia Magnenat Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 137–144. ACM Press, 1995.

[128] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. *Computer Graphics Forum*, 20(3):261–267, 2001.

[129] J. Lombardo, M. Cani, and F. Neyret. Real-time collision detection for virtual surgery. In *Proceedings of Computer Animation*, pages 33–39, 1999.

[130] S. Cotin, H. Delingette, and N. Ayache. Real-time elastic deformation of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):62–73, 1999.

[131] René Weller and Gabriel Zachmann. Kinetic separation lists for continuous collision detection of deformable objects. In *Third Workshop in Virtual Reality Interactions and Physical Simulation*, 2006.

[132] Sean Curtis, Rasmus Tamstorf, and Dinesh Manocha. Fast collision detection for deformable models using representative-triangles. In *SI3D '08: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, pages 61–69, 2008.

[133] Wenping Wang, Jiaye Wang, and Myung-Soo Kim. An algebraic condition for the separation of two ellipsoids. *Computer Aided Geometric Design*, 18(6):531–539, 2001.

[134] Philip Schneider and David H. Eberly. *Geometric Tools for Computer Graphics*. Morgan Kaufmann, 2002.

[135] Thomas Larsson. An efficient ellipsoid-OBB intersection test. *journal of graphics tools*, 13(1):31–43, 2008.

[136] Jeff Erickson. Local polyhedra and geometric graphs. *Computational Geomtery: Theory and Applications*, 31:101–125, 2005.

[137] Subhash Suri, Philip M. Hubbard, and John F. Hughes. Analyzing bounding boxes for object intersection. *ACM Transactions on Graphics (TOG)*, 18(3):257–277, 1999.

[138] Jan Klein and Gabriel Zachmann. The expected running time of hierarchical collision detection. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Posters*, page 117, 2005.

[139] Rene Weller, Jan Klein, and Gabriel Zachmann. A model for the expected running time of collision detection using AABB trees. In *Proceedings of the 12th Eurographics Symposium on Virtual Environments*, pages 11–17, May 2006.

[140] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the Spring Joint Computer Conference*, pages 37–45, 1968.

[141] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, 1980.

[142] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice, 2nd Edition.* Addison-Wesley, 1996.

[143] Alan Watt. *3D Computer Graphics, 3rd Edition.* Addison-Wesley, 2000.

[144] Edward Angel. *Interactive Computer Graphics: A Top-Down Approach Using OpenGL, 5th Edition.* Addison-Wesley, 2008.

[145] Andrew Glassner. *An Introduction to Ray Tracing.* Academic Press, 1989.

[146] Peter Shirley. *Realistic Ray Tracing.* A K Peters, 2000.

[147] Peter Shirley. *Realistic Ray Tracing, 2nd Edition.* A K Peters, 2003.

[148] Kevin Suffern. *Ray Tracing from the Ground Up.* A K Peters, 2007.

[149] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter Shirley, Brian Smits, and Charles Hansen. Interactive ray tracing. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, pages 119–126. ACM Press, 1999.

[150] Ingo Wald and Philipp Slusallek. State of the art in interactive ray tracing. In *State of the Art Reports, Eurographics 2001*, September 2001.

[151] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, 20(3):153–164, 2001.

[152] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. *ACM Transanctions on Graphics*, 24(3):1176–1185, 2005.

[153] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G. Parker. Ray tracing animated scenes using coherent grid traversal. *ACM Transanctions on Graphics*, 25(3):485–493, 2006.

[154] Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. Interactive global illumination using fast ray tracing. In *Eurographics Workshop on Rendering*, 2002.

[155] E. Reinhard, B. Smits, and C. Hansen. Dynamic acceleration structures for interactive ray tracing. In *Proceedings of the 11th Eurographics Workshop on Rendering*, pages 299–306, 2000.

[156] Maxim Shevtsov, Alexei Soupikov, and Alexander Kapustin. Highly parallel fast KD-tree construction for interactive ray tracing of dynamic scenes. *Computer Graphics Forum*, 26(3):395–404, 2007.

[157] Ingo Wald, Thiago Ize, and Steven G. Parker. Fast, parallel, and asynchronous construction of BVHs for ray tracing animated scenes. *Computers & Graphics*, 32(1):3–13, 2008.

[158] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics (TOG)*, 21(3):703–712, 2002.

[159] David Roger, Ulf Assarsson, and Nicolas Holzschuch. Whitted ray-tracing for dynamic scenes using a ray-space hierarchy on the GPU. In *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)*, pages 99–110, 2007.

[160] Thomas Larsson and Tomas Akenine-Möller. Strategies for bounding volume hierarchy updates for ray tracing of deformable models. Technical Report MDH-MRTC-92/2003-1-SE, Department of Computer Science and Engineering, Mälardalen University, February 2003.

[161] C. Lauterbach, Sung-Eui Yoon, David Tuft, and Dinesh Manocha. RT-DEFORM: Interactive ray tracing of dynamic scenes using BVHs. In *IEEE Symposium on Interactive Ray Tracing*, pages 39–46, 2006.

[162] Carsten Wächter and Alexander Keller. Instant ray tracing: The bounding interval hierarchy. In *Eurographics Symposium on Rendering*, pages 139–149, 2006.

[163] Miguel R. Zuniga and Jeffrey K. Uhlmann. Ray queries with wide object isolation and the DE-Tree. *journal of graphics tools*, 11(3):27–45, 2006.

[164] Sven Woop, Gerd Marmitt, and Philipp Slusallek. B-KD trees for hardware accelerated ray tracing of dynamic scenes. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, pages 67–77, 2006.

[165] Sven Trenkel, René Weller, and Gabriel Zachmann. A benchmarking suite for static collision detection algorithms. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, Plzen, Czech Republic, 2007.

[166] David Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. PhD thesis, Cornell University, 1992.

[167] Ladislav Kavan, Carol O'Sullivan, and Jiří Žára. Efficient collision detection for spherical blend skinning. In *GRAPHITE '06: Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, pages 147–156, 2006.

[168] Jonas Spillmann, Markus Becker, and Matthias Teschner. Efficient updates of bounding sphere hierarchies for geometrically deformable models. *Journal of Visual Communication and Image Representation*, 18(2):101–108, 2007.

[169] Miguel A. Otaduy, Daniel Germann, Stephane Redon, and Markus Gross. Adaptive deformations with fast tight bounds. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 181–190, 2007.

[170] Thomas Larsson, Tomas Akenine-Möller, and Eric Lengyel. On faster sphere-box overlap testing. *journal of graphics tools*, 12(1):3–8, 2007.

[171] Joseph O'Rourke. Finding minimal enclosing boxes. *International Journal of Computer and Information Sciences*, 14(3):183–199, June 1985.

[172] Pankaj K. Agarwal and Micha Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete & Computational Geometry*, 16(4):317–337, 1996.

[173] Kenny Erleben. An introduction to approximating heterogeneous bounding volume hierarchies. Technical report, Department of Computer Science, University of Copenhagen, 2002.

[174] Chuan-Jun Su, Lin Fu-Hua, and Xiao ke Zhang. An efficient collision detection methodology for virtual assembly. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 360–365, Oct 1998.

[175] James Arvo. A simple method for box-sphere intersection testing. In Andrew Glassner, editor, *Graphics Gems*, pages 335–339. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[176] Philip M. Hubbard. Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, pages 24–31, 1993.

[177] M. H. Overmars. Point location in fat subdivisions. *Information Processing Letters*, 44:261–265, 1992.

[178] B. Mirtich. Efficient algorithms for two-phase collision detection. In *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 203–223, 1998.

[179] Daniel H. H. Ingalls. A simple technique for handling multiple polymorphism. *ACM SIGPLAN Notices*, 21(11):347–349, 1986.

[180] P. Volino and Nadia Magnenat Thalmann. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, 13(3):155–166, 1994.

[181] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics (TOG)*, 21(3):594–603, 2002.

[182] David Baraff, Andrew Witkin, and Michael Kass. Untangling cloth. *ACM Transactions on Graphics (TOG)*, 22(3):862–870, 2003.

[183] Jie Gao, Leonidas J. Guibas, and An Nguyen. Deformable spanners and applications. In *SCG '04: Proceedings of the Twentieth Annual Symposium on Computational Geometry*, pages 190–199, 2004.

[184] Jan Klein and Gabriel Zachmann. Point cloud collision detection. *Computer Graphics Forum*, 23(3):567–576, 2004.

[185] Kayvon Fatahalian and Mike Houston. GPUs: a closer look. *Queue*, 6(2):18–28, 2008.

[186] William Mark. Future graphics architectures. *Queue*, 6(2):54–64, 2008.

II

Included Papers