uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Peiran Liu

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Ph.D. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Progressive Transmission and Multi-resolution Collision Detection of Polygonal Meshes in Virtual Environments

TITRE DE LA THÈSE / TITLE OF THESIS

Nicolas Georganas

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

Gerhard Roth

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Jochen Lang

WonSook Lee

Ming Lin

Michiel Smid

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

Progressive Transmission and Multi-resolution Collision Detection of Polygonal Meshes in Virtual Environments

Peiran Liu

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the PhD degree in Electrical Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering
School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-15029-0
Our file Notre référence
ISBN: 978-0-494-15029-0

To my parents

Acknowledgments

I would like to thank Professor Georganas for his enthusiastic support and inspiring advices. The School of Information Technology and Engineering has been a great work environment.

I am grateful to Dr. Roth for accepting co-advisorship and for providing many good advices and supports.

Special thanks go to all current and former members of our extraordinary research lab "Distributed and Collaborative Virtual Environments Research Laboratory".

I am grateful to the LORNET NSERC Research Network for supporting this research.

Last but not least, my thanks go to my parents for their persistent encouragement and for always being there.

Peiran Liu,

Canada in April 2006.

Abstract

Virtual reality applications such as computer games form a large portion of the computing industry. The increasing expectations for immersive and interactive entertainment require fast graphics rendering with a lot of details and high-fidelity physics simulation. Interference and collision detection between general polygonal models, including all those tasks in the simulated motion of solids which cannot penetrate one another, has been widely explored. The most recent works have been focused on increasing the number of collision queries per second to an interactive rate, assuming that the computing resources such as memory, CPU capacity, and network bandwidth are sufficient all the time. However, the performance of a collision detection algorithm may vary in real environments where the available computing resources are not sufficient or are dynamic at runtime. Such an unstable performance directly affects the interactive rate of many applications, which is not acceptable. This drawback has been noticed extensively by practitioners. Little research work has been introduced to systematically solve this problem. One way out of this problem is to use a technique to dynamically adjust the cost of the collision detection (CD) task according to the measurements of available resources.

In this dissertation, a solution to the problem is proposed. The problem is analyzed under three different physical environments: single low end computer, clustered multiple-machine setting, and distributed environment. An extensive comparative study of several methods is done in the context of polygonal models. A new approach, "multi-resolution collision detection", is proposed. It performs time-critical and exact interference detection on continuous level of detail (LOD) representations of arbitrary triangle meshes undergoing

rigid body motion. The goal of this work is to extend the bounding volume hierarchy (BVH) based collision detection approach from static to dynamic in order to adapt the cost of collision detection to the available resource in either centralized or distributed environments. Several adaptive mesh refinement criteria based on viewing distance, application context, and movement velocity are proposed. Global and local mesh refinement algorithms and a BVH CD algorithm are introduced. They are computationally simple and achieve reliable real-time performance. The complexity of the proposed algorithms is analyzed and proved. Measurements are taken to compare with the analytical results and strong agreements are achieved. The major application areas considered are distributed virtual environments (DVEs) and interactive haptic applications.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AABB	Axis-aligned Bounding Box
BVH	Bounding Volume Hierarchy
CD	Collision Detection
CDPM	Collision Detection Progressive Mesh
CLOD	Contact LOD
CPM	Compressed Progressive Meshes
CR	Collision Response
ECOL	Edge Collapse
DVE	Distributed Virtual Environment
GPU	Graphics Processing Unit
HCI	Human-computer Interface
HRTC	Haptic Real Time Controller
LOD	Level of Detail
OBB	Oriented Bounding Box
PM	Progressive Mesh
QEM	Quadric Error Metrics
SPM	Space Partitioned Mesh
VE	Virtual Environment
VR	Virtual Reality
VSP	Vertex Split

Chapter 1 Introduction

1.1 Computer Graphics and Virtual Environments

Computer graphics is a field concerned with all aspects of producing pictures or images of three-dimensional objects using a computer. This field covers three areas, modeling, rendering, and animation [Shi02, Ang97, BHT97]. *Modeling* deals with the mathematical specification of shape and appearance properties in a way that can be stored in the computer. *Rendering* is a term inherited from the arts and deals with the creation of shaded images from 3D computer models. *Animation* is a technique to create an illusion of motion through sequences of images. Here, modeling and rendering are used, with the handling of time as an issue not usually dealt with in basic models. The combination of computers, networks, and the complex human visual system through computer graphics, has led to new ways of displaying information, seeing virtual worlds, and communicating with both other people and machines. Although the applications of computer graphics are many and varied, we can divide them into four major areas. Display of information (medical imaging, information visualization, film special effects); Design (CAD/CAM); Simulation and animation (video games, flight simulator, haptic rendering); User interfaces (graphical network browser).

From a historical point of view, virtual reality is a logical consequence of an on-going virtualization of our every-day life [EDd+96]. The definition of virtual reality involves three components:

1. Real-time interaction;
2. Real-time simulation;

3. Immersion and direct interaction by I/O devices.

From a technical point of view, a VR system must meet three criteria:

1. Interaction with the virtual environment must be immersive and intuitive;

2. Rendering must be done in real-time and without perceptible lag (30 Hz for graphics, 1k Hz for haptics);

3. Object behavior must be simulated in real-time.

As an important part of virtual reality technology, computer graphics deals with real-time display of images to fulfill the requirements of human visual perception. Virtual reality also involves some other human factors, such as human auditory perception, force field display / haptic display, and computer generated artificial intelligence. With the advent of virtual reality technology computer-based synthetic worlds, which are referred to as virtual environments (VE), are attempting to imitate the real world. A distributed virtual environment (DVE) is a networked multi-user VR system where participants navigate in synthetic 3D space and see, meet and interact with other users and applications.

1.2 Problem Statement

A major objective of virtual reality is to create realistic-looking motion. A major component of realistic motion is the physically-based reaction of rigid bodies to commonly encountered forces such as gravity and those resulting from collisions. Collision detection (CD) and collision response (CR) are two issues to be addressed in which collision detection is the most time-consuming task of many simulation issues. *Collision detection* is strictly a kinematic issue in that it has to query the positions and orientations of objects and how they change over time [Ric02]. It deals with finding out whether or not there is a geometric contact about to occur or has occurred. Sometimes, we also want to know when and where

this event happens. *Collision response* is a consideration in physics based simulation. Instead of the geometric extent of an object, the distribution of its mass is an important part of the calculations. In a geometric context, a *collision query* or *proximity query* reports information about the relative placement of two objects. Some of the common examples of such queries include checking whether two objects overlap in space, or whether their boundaries intersect, or computing the minimum Euclidean separation distance between their boundaries. These computations become quite involved when dealing with a large number of non-trivial complex object models (i.e., several tens or hundreds of objects, each consisting of some 10,000–100,000 polygons). Many research works on different aspects of these queries have been done in robot motion planning, dynamic simulation, haptic rendering [MPT99], interactive walkthroughs (Figure. 1.1), computer gaming, and molecular modeling. A virtual environment is used to present the user a sense of presence in a synthetic world and it should make the images of both the user and the surroundings feel solid. Objects in the virtual world should not pass through each other. They should move as expected when pushed, pulled, or

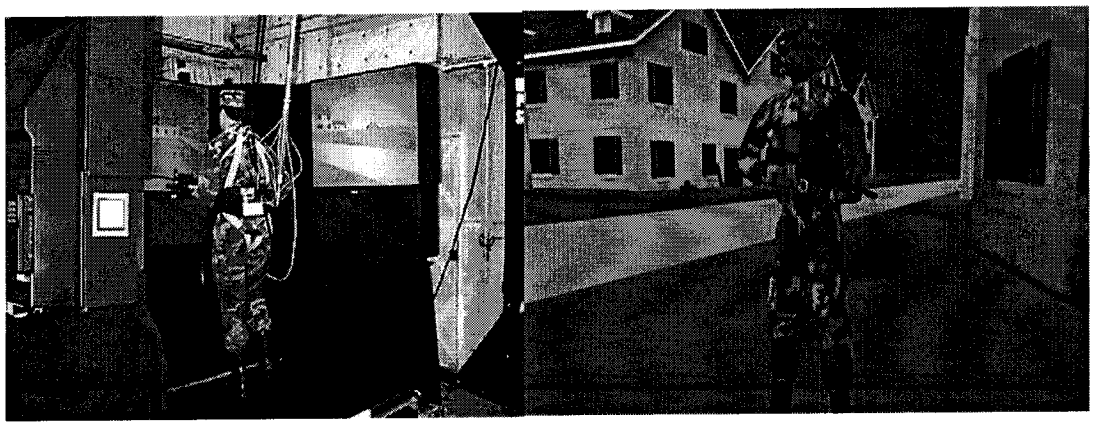


Figure 1.1 Illustrate a Walkthrough Application Built by NPSNET-IV for Articulated Humans Either Mounted or Dismounted [SZ99].

grasped. These interactions among the objects require fast and accurate collision detection among their geometric representations. A large number of collision detection algorithms have been proposed. However, most of them are static algorithms. First of all, they claim that real-time performance is achieved under the condition that the computing resources are sufficient. In other words, the resources are static. Free memory space is always available to load models and store required data structures, CPU is always free to do this specific computing task, and the available network bandwidth is always sufficient to transmit geometric models in a negligible time. However, this assumption may not be satisfied in some real computing environments in which the resources are very limited or are varied continuously. The drawback is that the real-time performance may not be satisfied or it may become very unstable, which directly affects the interactive rate of the applications. For many real-time applications such as haptic surgery and flight simulation, etc., this is unacceptable. Second, the geometric models as input data to these algorithms are static. The geometry and topology of the models are not changed at runtime. The data are input to the program only at initialization phase. The collision detection can not start until all models are loaded. In addition, some constrains are imposed on the input of the algorithms, such as the

Figure 1.2 Virtual Mall (Courtesy of www.activeworlds.com)

representation of the object (Constructive Solid Geometry, convex or non-convex polygonal model), the complexity of the representation (the number of features of the geometric models), the motion of the object (parameterized or arbitrary trajectory), and the number of pair wise collision queries. Due to these limitations, the algorithms may not satisfy the desired performance of many new applications such as distributed virtual malls (Figure 1.2), massive multiplayer online games (Figure1.3), and haptic rendering of large structures. The emerging of the new applications introduces new issues, which motivate a number of practical problems.

1.2.1 Massive Complex Models in Virtual Environments

Massive complex models in non-convex polygonal representation are required in virtual environments. Highly detailed geometric models are necessary to increase the accuracy of collision detection and collision response. These models are often represented as complex triangle meshes. As the complexity of the models increase, the costs of collision queries increase accordingly. As the number of objects in the virtual world rises, the number of pair-

Figure 1.3 A Screenshot of Rainbow Six, a Multiplayer Online Game (Courtesy of www.ubi.com)

wise collision queries increases Therefore, the overall cost of collision detection increases significantly. Algorithms with the current best run time for convex polytope collision queries take $O(n)$ time [CW96, GJK88], where n is the number of features (face, edge, and vertex). If the objects do not move swiftly the best runtime can be roughly constant [LC91]. Algorithms for collision queries between non-convex polygonal models are dominated by the hierarchical bounding volume (BVH) approach of which the cost is determined by the choice of bounding volume and the complexity and the relative placement of models [GLM96]. When applying these algorithms to applications in which the size of input models in terms of the number of features is small, a real-time performance may be achieved. However, when the models are represented by millions of polygonal faces, the performance of collision queries degrades. To accelerate collision queries, many algorithms utilize spatial and temporal coherences of geometry, which require a time consuming preprocessing to collect topology and geometry information of the models. Frequent changes to the models make the algorithms very inefficient because time consuming preprocessing is required whenever the changes happen. Practically, in order to speed up collision queries, some algorithms deliberately introduce errors by reducing the input size. Collision queries are performed on crudely approximating shapes of the objects. For example, an avatar rendered in thousands of polygons might be approximated as a sphere or a convex bounding polytope in a collision query. This simplification can lead to inconsistencies, whereby it visually appears to the users that the avatar can squeeze through an area, though the simulation makes it impossible to do so.

It is clear that the complexity of input models is a factor that significantly affects the complexity of the CD algorithms. In this work, we pursued an approach to control the

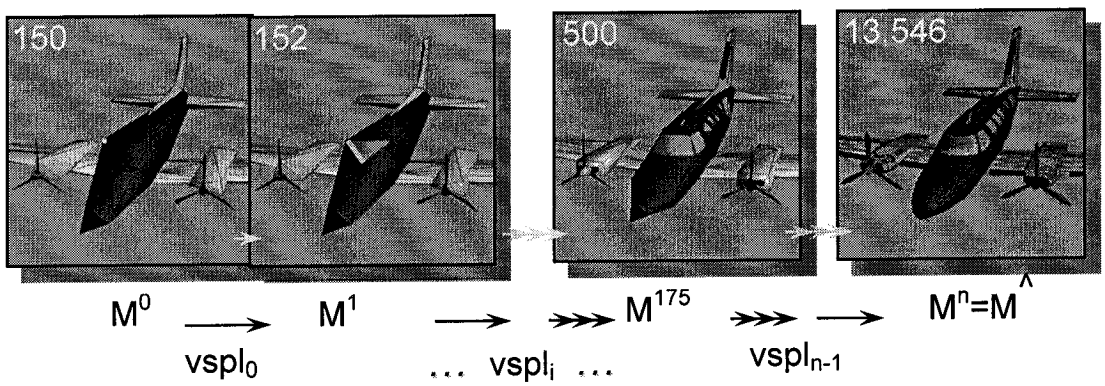


Figure 1.4 A Progressive Mesh (PM) at Multiple Resolutions [Hop96]

complexity of CD by dynamically adjusting the complexity of the models. For a model, the level of complexity is determined by the availability of computational resources at an instance and by how realistic and consistent animation a user expects on the specific object in the virtual world. The goal is to maximize the utilization of the computational resources and to provide an overall more consistent and realistic virtual environment.

1.2.2 Storage and Transmission

Complex triangle meshes are expensive to store and transmit. In a distributed virtual environment, e,g., an online virtual mall, maintaining a huge volume of geometric data of static and dynamic objects requires a large storage space. Updating and distributing these data requires a reliable and fast data communication network. These tasks cannot be handled in a single computer. From the perspective of physical deployment, a common solution is to store the data in a geographically dispersed grid-computing environment or in distributed clusters, which enables sharing data across networks and collaboration among end users. From the perspective of data processing, the problem of minimizing the storage space for meshes has been addressed in two ways. One is mesh simplification: a mesh is simplified to a nearly indistinguishable approximation with far fewer faces [Hop93]. The other is mesh

compression: minimizing the space taken to store a particular mesh [TR98]. The problem of minimizing the transmission bandwidth of meshes for graphics rendering has been investigated in the work of progressive mesh [Hop96] as shown in Figure 1.4. The idea comes from static level of detail (LOD) and the transmission of images in progressive JPEG. The static LOD defines several versions of a model at various levels of detail. A detailed mesh is used when an object is close to the viewer in view volume, and coarser approximations are substituted as the object recedes. A progressive mesh is represented in a continuous LOD. When a mesh is transmitted over the network, viewers would see progressively better approximation to the model as data is incrementally received [CLL+03].

However, when performing a CD operation the problem of minimizing the transmission bandwidth of meshes introduces some new problems. Existing CD algorithms have a limitation in common. In a distributed application, to start a collision query at the client, mesh data have to be completely loaded in local memory. This will cause the client to start slowly if the application runs on an unreliable or low bandwidth network. In a distributed application, collision detection on objects that rarely collide with other objects, such as walls and ceilings, requires the transmission of the coarsest meshes of the objects from a server. For those objects at high risk of collision, such as avatars and moving cars, the meshes in finest resolution need to be transmitted. Such a globally refined approach can significantly reduce the transmission bandwidth of the meshes and start the client instantly. Each mesh captures the surface of a model at a uniform (global) resolution. Sometimes it is desirable to do collision queries on selected regions where collisions are most likely to happen. In a flight simulation for instance, as a user flies over a terrain mesh, the regions near the viewer are at high risk of collision, therefore need to be refined to the finest resolution. The regions far

away from the viewer only need to be refined to the coarsest resolution. When relative position and orientation among objects change the topology of the meshes is changed accordingly. The input size of collision detection is affected by the predicted contact regions of the objects in the virtual world instead of the data volume of the meshes. Such a locally refined approach can further increase the accuracy of the collision detection.

An accurate and fast collision detection algorithm is a fundamental component of a distributed complex virtual environment. How to accelerate collision detection and maintain a high accuracy is a challenging issue. In this thesis, we explore an approach to selectively refine and progressively transmit meshes either globally or locally for collision detection. Figure 1.5 illustrates the globally refined collision detection algorithm. The goal is to minimize transmission bandwidth requirement in distributed environments, accelerate the initialization at the client, lower the complexity of collision detections, and selectively increase the accuracy to perform collision detection on objects.

1.2.3 Highly Interactive Haptic Applications

Compared to presentation of visual and auditory information in virtual environments, methods for haptic interaction are not as well developed. Haptic interaction, as an augmentation to visual and auditory displays, enriches the perception and understanding of both force fields and world models populated in synthetic environments. Multiple tasks, such as haptic sensing/actuation, visual updates must be accomplished in a synchronized manner in haptic applications. It becomes commonplace to separate tasks into computational threads or processes, to accommodate different update rates, distribute computation load, and optimize computation. Conventionally, multithreading and multiprocessing software architectures are applied to develop effective multimodal VEs to optimize the usage of CPU

capabilities [HBS99]. However, an important consequence of the conventional architectures is that it makes the operating system an inherent component of the applications. This means that the operating system scheduling algorithms limit the application's quality of service. The application may request a theoretical rate of force display but it is the scheduler that determines the actual rate. This scheduler is itself a complex algorithm, particularly when considered in terms of its interactions with the other services provided by the operating system [KS04]. Multi-machine solution for a haptic application was addressed in [SBN+03, SZE+04]. Unlike conventional multithreading or multiprocessing approaches for haptics, this multi-machine model solution applies a hard real-time operating system for haptic control, while applying a mainstream soft real-time operating system for the application and the graphics. It also provides the potential for tele-haptic applications to switch between multiple protocols, for example, one for large-scale distributed simulations and another adapted to collaboration when several users meet and need to perform a collaborative task. It also requires an architecture that supports those different protocols.

Two major tasks in haptic interaction paradigms are collision detection and collision response. The goal of collision detection is to detect collisions between the end point of a generic probe and the objects in a scene, while the goal of collision response is to respond to the detection of collision in terms of how the forces reflected to the user are computed. Studies of human tactile perception of contact information have shown that a desired force update rate is preferably 1 kHz. Unfortunately many proposed algorithms targeted graphics applications which require a relatively low collision query rate (desirably 30Hz).

In this dissertation we show that the update rate can be raised by applying the "locally refined" collision detection on multi-resolution meshes in haptic interaction applications. At runtime, the meshes are dynamically refined to higher resolution in areas that are most likely to collide with other objects. The algorithms are successfully demonstrated in a simulated interactive haptic environment. Compared to existing CD algorithms on single resolution models, noticeable performance improvement has been observed in terms of the precision of collision queries, frame rate, and memory usage.

1.3 Contributions

Despite promising research efforts since the beginning of virtual reality, there are a number

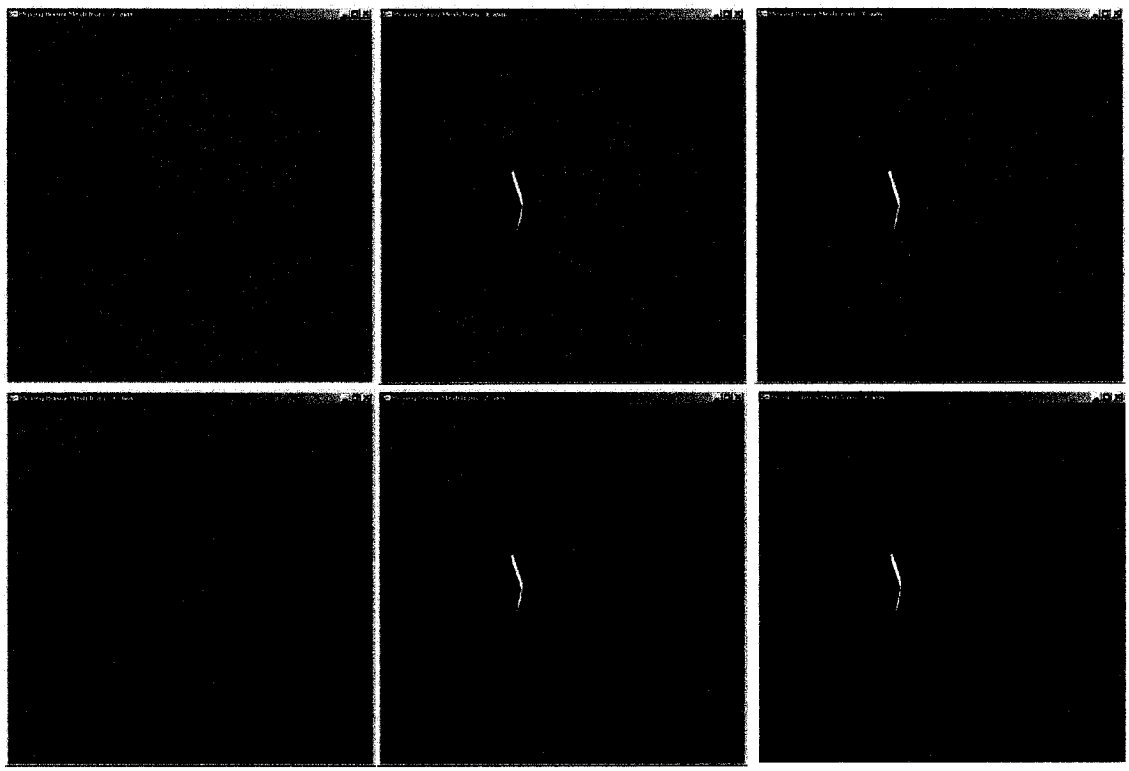


Figure 1.5 Illustration of a Multi-resolution Collision Detection Algorithm. From Left to Right the Sphere Model in Collision is Refined from Low LOD to High LOD.

of unsolved problems. Rendering speed is still too slow for complex scenes. Human-computer interface devices are very immature. They are cumbersome, clumsy, inaccurate, and limited. This includes tracking as well as visual and haptic / tactile rendering. Creating or authoring virtual environments for highly interactive scenarios with complex object behavior and complex interactive processes is a time-consuming task.

Real-time physically-based simulation of non-trivial object behavior has been recognized as one of the major missing ingredients. Behavior includes rigid body dynamics, inverse kinematics, flexible objects, etc. All of these problems have been solved in theory and in non-real-time systems. However, simulating that behavior in real-time for non-trivial object complexities and numbers (i.e., several tens or even hundreds of objects, each consisting of some 10,000–100,000 polygons) has not been resolved. To my experience, as the most time-consuming part of many simulation problems, collision detection which is superior in theory might lose in practice. To the best of my knowledge, little research work has been published to discuss and resolve the issue of unreliable performance of collision detection in environments where cacheable memory space is not large enough to load all required data, underlining operating system is not real-time, network connection is unreliable, and network bandwidth is unstable. The goal of this work is to establish a solution to provide real-time collision detection in practical virtual environment applications.

The solutions presented in this dissertation are applicable to not only centralized but also distributed virtual environment systems in general. In this thesis, we present a new approach to CD, called multi-resolution collision detection. It is based on a form of bounding volume hierarchy ("AB-Tree"). The main contributions are considered from the following aspects.

Time Cost

Typically, the input to a collision detection algorithm is a large number of geometric objects comprising an environment, together with a set of objects moving within the environment. In addition to detecting accurately the contacts that occur between pairs of objects, one needs also to do so at real-time rates. The complexity of a BVH CD algorithm is determined by the complexity of the input models and relative configurations of objects in the environment. VR applications run faster on computers with more powerful CPUs. However, if the underling operating system is not real-time, then the available CPU time might be changed at any time. This makes the performance of CD which is the most time consuming task of physically-based simulation, degrade in practice. In order to obtain a real-time performance, the time cost for CD must be adjustable according to available CPU time. The relative configurations of objects in the environment are context oriented. They are mostly controlled by users through human-computer interface (HCI) or by computer generated intelligent agents. Thus we focus on controlling the complexity of input models. The "multi-resolution mesh" technique is used to adjust the complexity of the models. Collision queries are performed on the BVHs. The BVH algorithms proposed by other researchers assume that the object models don't change after being loaded to local cache. Thus the BVHs constructed initially can not be reconstructed very fast when the models are changed at run time. This thesis presents a new data structure, an AB-Tree along with the associated algorithms which solve the problem of performing fast CD on models with dynamically changing complexity. Also, in this way applications running on low end computers can obtain the same collision detection frame rate as high end computers but lose some accuracy. The multi-resolution collision detection is useful for applications, such as a video game, walk through application, or where real-time interaction is more important than realism of physics simulation.

Space Cost

The increasing expectation for fast high-fidelity physics simulation requires more and more complex models. In a large scale VE application with hundreds of complex objects, the required storage space for the entire environment is not available on a single computer. BVH CD algorithms assume that the input objects models do not change after loading time. For example, if there are 100 objects in the environment, each of size 10 MB, then at least a 1GB main memory space is required just for CD initialization. Considering the space required for other tasks of the application, those CD algorithms can not be handled by regular PCs in practice. By integrating the LOD selection method proposed in this thesis with the multi-resolution CD approach, the requirement for storage space is significantly reduced without sacrificing the overall realism of the physics simulation. For large scale models with millions of polygon, a locally refined approach is presented to further reduce space cost by selectively refine the models to higher resolution in areas that collisions are very likely to occur. Other areas of the models remain in low resolution. Although the total size of the models is large, the size of the data in use is small.

Data Availability

In a distributed virtual environment, object models are stored in servers and are transmitted through network to clients upon request. When VE applications are run in distributed environments, network transmission and related issues have to be considered. First, the network connection may be unreliable. For example, when the network connection is down, only some of the objects belonging to the virtual scene or part of an object is received at the client. This may suspend the application from execution until the network connection failure is recovered and all the requested data are received successfully. For internet based

applications, such as massive multiplayer online games, suspending the application from time to time is unacceptable. The proposed multi-resolution collision detection solves this problem very well. Models in simplified shape are transmitted initially. Then the complexity of the models is increased progressively by streaming the models in a specific representation. If the network is down after the simplified models are received at the clients, CD can be performed on the simplified models. Thus the task for physics simulation does not suspend. Second, the network bandwidth is low. In such a situation, if the entire scene has to be received before starting the simulation, then the client has to wait a long time to start execution. Again, the multi-resolution CD approach allows the clients to start execution whenever the simplified models are received. In other words, the clients can start instantly if the sizes of the simplified models are relatively small.

The main contributions of this thesis include:

- A new BVH CD algorithm, "AB-Tree", is introduced which performs output-sensitive CD on dynamically refined meshes at a cost similar to that of the existing CD algorithms on static LOD meshes. The cost of mesh refinement is as low as a small constant when the objects do not move swiftly and do not move very often.

- A real-time collision detection framework for DVEs is proposed. A new approach, "multi-resolution collision detection" is introduced. It performs collision detection between multiple complex object models in a collision detection progressive mesh (CDPM) representation. A new LOD selection method is introduced in this work. It has been shown that the proposed multi-resolution CD approach significantly reduces the requirement of data transmission over high bandwidth networks, saves the time

for initialization at the client, and selectively increases the accuracy of collision detection on object models.

- In addition, a new multi-resolution mesh representation "Space Partitioned Progressive Mesh (SPM)", which is an extension to CDPM, is introduced. The meshes can be progressively transmitted on demand and locally refined at runtime on large scale mesh models. This research work has successfully exploited the potential of dynamic network resources to increase the accuracy of CD for interactive DVEs. The local mesh refinement method is derived from view-dependent LOD techniques. The major contributions are to integrate it with the proposed BVH CD algorithm to solve particular issues of CD in DVEs and to prove the correctness of local refinement algorithm.

- The collision detection framework for DVEs is further extended for haptic interactions between multiple probes and complex object models in a multi-machine architecture. The high update rate of force display and the limited memory capacity of haptic rendering hardware are two of the most challenging issues we tackled. In such a real-time multi-machine framework, SPMs are progressively transmitted on demand and locally refined at runtime. The AB-tree algorithm allows us to bound the input size of the problem, thus improving the desired collision query performance for force display.

Publications arising from this thesis:

[LSG+03] P. Liu, X. Sun, N. D. Georganas, E. Dubois. *Augmented Reality: A Novel Approach for Navigating In Panorama-Based Virtual Environments (PBVE).* In Proc. IEEE International Workshop on Haptic, AudioVisual Environments and their Applications (HAVE'2003), Ottawa, Sept. 2003.

[LSG+05] P. Liu, X. Shen, N.D. Georganas and G. Roth. *Multi-resolution Modeling and Locally Refined Collision Detection for Haptic Interaction,* In Proc. of 3-D Digital Imaging and Modeling conf., Ottawa, Jun. 2005.

[LGR06] P. Liu, N.D. Georganas, and G. Roth. *Locally Refined Collision Detection of Large Scale Complex Polygonal Meshes in Distributed Virtual Environments.* In Proc. IEEE International Conference on Multimedia & Expo (ICME06), Toronto, July, 2006.

[LGR] P. Liu, N.D. Georganas, and G. Roth. *Handling Rapid Interference Detection of Progressive Meshes Using Active Bounding Trees.* Journal of Graphics Tools, accepted for publication.

[LGR] P. Liu, N.D. Georganas, and G. Roth. *Multi-resolution Collision Detection in Distributed Virtual Environments.* Computer Graphics Forum, the International Journal of the Eurographics Association, submitted.

1.4 Thesis Organization

In this chapter, the main problems of collision detection in DVEs is introduced and a brief overview of the contributions of the thesis is provided. Chapter 2 presents a comprehensive survey of DVEs, haptic interaction applications, mesh coding techniques, and state-of-art CD methods. Chapter 3 presents a multi-resolution collision detection approach which tackles

the issue of providing real-time performance in non-uniform and dynamic computing environments. A BVH CD algorithm "AB-tree" and a multi-resolution mesh CDPM are introduced. Chapter 4 presents a framework for integration of multi-resolution collision detection into DVEs. Various issues relating to implementation and performance are discussed in it. Chapter 5 deals with the problem of CD of large scale models in DVEs. A locally refined CD approach is developed comprising the earlier presented AB-tree algorithm and a space partitioned mesh representation method SPM. The algorithm for local refinement is presented and the correctness is proved. In addition, implementation and performance issues are presented. A summary of the thesis and a discussion of future works are given in chapter 6. Several applications that can be built on top of the frameworks and algorithms presented earlier are described. They prove the usefulness, flexibility, and power of the algorithms and frameworks developed in this thesis, helping to make physics simulation in DVEs more realistic.

Chapter 2 Review of the State of the Art

In this chapter, we provide a comprehensive review of the following topics: distributed virtual environments, haptic interaction, multi-resolution meshes, view-dependent LOD, and collision detection methods. Then we justify why the problems we choose are important and how our work differs and improves on the work of others.

2.1 Distributed Virtual Environment

A distributed virtual environment (DVE) is a network-based multi-user VR system where participants navigate in synthetic space and see, meet and interact with other users and computers. The history of the development of DVEs is illustrated in Figure 2.1. The origin of DVEs comes from two communities: military simulation and networked games. A series of successful distributed simulation projects pushed the promotion of standards for distributed simulation, e.g., DIS and HLA [SZ99]. The fast growing networked games triggered the improvement of techniques for high-quality interactive graphics applications.

In DVEs, two main approaches have been proposed to distribute virtual objects from servers to clients. Many early systems such as SIMNET and DIVE [HLS97, CH93] employ the *complete replication* approach to distribute geometry data to the clients before the simulations are started. This approach assumes the use of high-speed networks to transmit the usually large volume of data. For example, DIVE environments as illustrated in Figure 2.2 are distributed on heterogeneous networks (making use of the Isis library [Bir85]). New participants of the virtual world can join at any time and on doing so they will receive a copy of the current database. All behavior is specified as a (usually very simple) finite state-

machine (FSM). Any FSM is part of some object's attributes. Database consistency is achieved by using distributed locks. *On-demand transmission* is another approach used in NPSNET and Bricknet [SSP+94] to distribute the data to the clients at runtime. This approach requires the transmission of only the visible region of the virtual environment to the clients, which reduces startup time and optimizes network usage. For example, the NPSNET Visual Simulation System is a real-time, interactive distributed simulation system that was developed by the Naval Postgraduate School (NPS). The system is written in C++ and uses SGI's Performer. NPSNET reads MultiGen Flight databases and is DIS-compliant. The system can also read in SIMNET models. In the system, components are dynamically loaded in a distributed component-based environment through a directory service Lightweight Directory Access Protocol (LDAP).

However, there is a problem of maintaining an interactive rate at the clients. The visible objects need to be retrieved from the server in advance so that they are available whenever the clients need them. This problem is solved by a scheduling method [WM00] and by pre-fetching and caching methods [CLL+03]. These solutions only focused on providing

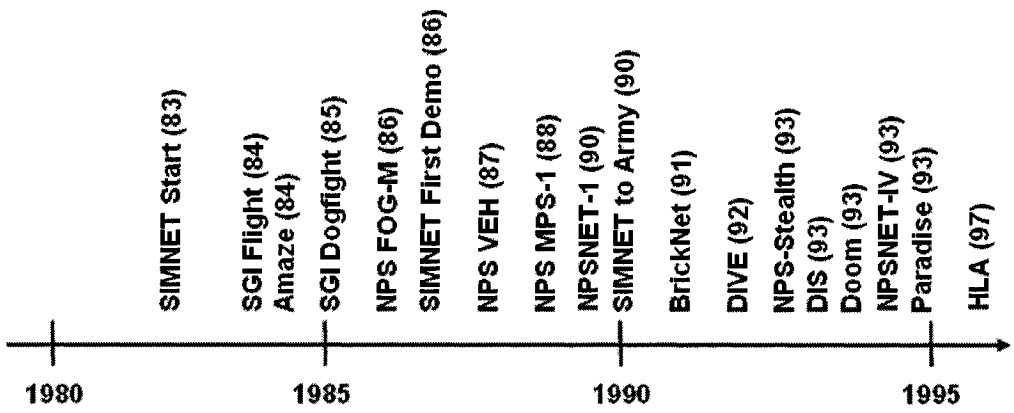


Figure 2.1 Illustration of the History of DVEs.

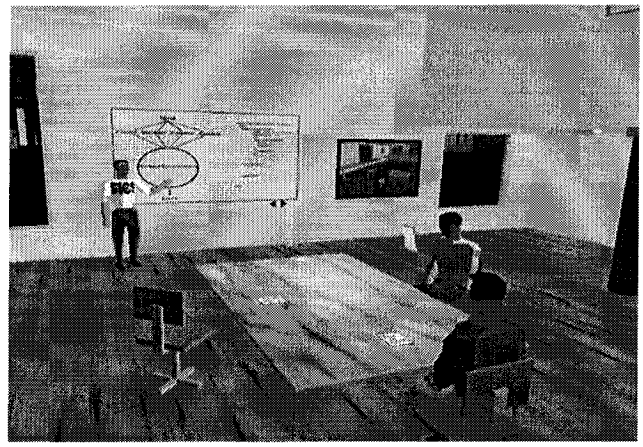


Figure 2.2 DIVE Environment Screenshot. DIVE has a homogeneous distributed dynamic database and uses reliable multicast protocols to replicate new objects [SZ99]

continued viewing service of the virtual environments. However, dynamic simulation, especially the issue of interactive and accurate collision detection over the distributed environment, is not tackled in the abovementioned works.

2.2 Haptic Interaction in Virtual Environments

Haptics, a term which was derived from the Greek verb "to touch", introduce the sense of touch and force in human-computer interaction. Haptics enable the human operator to manipulate the environment in a natural and effective way, enhance the sensation of "presence", and provide information such as stiffness and texture of objects, which cannot be described. Haptic interaction, as an augmentation to visual and auditory displays, enriches the perception and understanding of both force fields and world models populated in synthetic environments.

Early work was accomplished over three decades ago for tele-robotics applications. The technology has already been explored in contexts as diverse as modeling & animation, geophysical analysis, dentistry training, virtual museums, assembly planning, mine design,

surgical simulation, design evaluation, control of scientific instruments, and robotic simulation. Haptic interaction in virtual environments involves augmentation of a client station. These applications are typically implemented upon non- or soft- real time operating systems. The true potential of the technology in interactive virtual reality, tele-presence, tele-medicine and tele-manipulation applications has not been well studied [SZE+04, ZSG04, ESG04]. These tele-haptic interactions impose more stringent requirements. They demand hard real time guarantees [SZE+04].

Multiple tasks, such as haptic sensing/actuation, visual updates must be accomplished in a synchronized manner in haptic applications. It becomes commonplace to separate tasks into computational threads or processes, to accommodate different update rates, distribute computation load, and optimize computation. Conventionally, multithreading and multiprocessing software architectures are applied to develop effective multimodal VEs to optimize the usage of CPU capabilities [HBS99, GGT99]. A disadvantage of the conventional architectures is that the rate of force display is determined by the scheduler algorithm of the underlining operating system. For a soft real time operating system, the

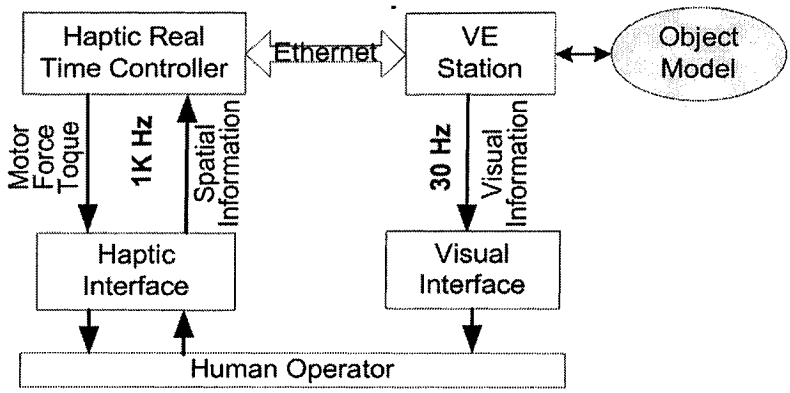


Figure 2.3 Multi-machine Architecture

actual rate may not satisfy the requirement.

Multi-machine solutions for a haptic application were addressed in [SZE+04, SBN03], as shown in Figure 2.3. The multi-machine architecture is comprised of three parts: haptic device, Haptic Real Time Controller (HRTC) and Virtual Environment (VE) graphics station. The HRTC communicates with its VE station through a local Ethernet connection and relies on a hard real time operating system (eg. QNX Neutrino, VxWorks or Windows CE) to guarantee the stability of the control loop. The separation of functionalities of haptic and graphic rendering makes this architecture easier to extend to existing applications. Unlike conventional multithreading or multiprocessing approaches for haptics, this multi-machine model solution applies a hard real-time operating system for haptic control, while using a mainstream OS such as Win2K or WinXP for the application and graphics. It also provides the potential for tele-haptic applications to switch between multiple protocols. For example, one for large-scale distributed simulations and one adapted to collaboration when several users meet and need to perform a collaborative task. It also requires an architecture that supports those different protocols.

Two major tasks in haptic interaction paradigms are collision detection and collision response. Studies of human tactile perception of contact information have shown that a desired force update rate is preferably 1 kHz. Although there is a huge volume of literature in the area of collision detection, many proposed algorithms targeted graphics applications which require a relatively low collision query rate (desirably 30Hz).

2.3 Mesh Coding

Highly detailed geometric models are becoming necessary to fulfill a growing expectation for realism in computer graphics. Consequently, objects are often scanned or tessellated into polygonal meshes at very high resolution to accommodate this need for detail. However, the computational cost of manipulating a model is directly related to its complexity. Detailed models result in large storage space, expensive transmission, and slow rendering. In addition, the full complexity of such models is not always required. Therefore, it is useful to have simpler versions of complex models. Several mesh coding techniques have been proposed to address these problems.

2.3.1 Traditional Meshes

A common representation in 3D computer graphics is the polygonal surface mesh because meshes can model objects of arbitrary shape and can be easily constructed from sensed 3D data. The resolution of a surface mesh is the overall spacing between vertices that comprise the mesh. A mesh whose 3D points are evenly distributed is in uniform resolution. Most often, surface meshes are represented by triangle meshes. A traditional mesh M is composed of four components, as illustrated in Figure 2.4. It can be represented as (V, K, D, S), where V is a set of vertex positions, K is a set of simplicial complexes, D represents discrete appearance attributes, such as material identifier, and S represents scalar appearance attributes, such as color, normal, texture coordinate. Each vertex is represented by three floating-point numbers. Mesh compression is needed in applications where there are hundreds of complex models in a scene.

2.3.2 Mesh Compression

Mesh compression methods are aimed at two complementary tasks. *Compression of geometry*: efficient encoding of numerical information attached to the vertices (position, surface normal). The compression is performed by discretizing coordinates in floating-point real numbers and expressing them on a predefined number of bits (lossy compression). Some methods are proposed for predicting vertex position from its predecessors in the bit stream. Tree-based prediction [TR98], parallelogram rule [TG98] used in the MPEG-4, and prediction based on triangle strips [Dee95] are examples. Algorithms that quantize vertex positions and normals, and then apply Huffman or entropy encoding can achieve compression in the range from 15:1 to 65:1, depending on the nature of the model. *Compression of mesh connectivity*: efficient encoding of the mesh topology. In general, an implicit encoding of Triangle-Vertex and Triangle-Triangle relations is needed. Techniques based on triangle strips have been used in graphics libraries (GL, OpenGL) for rendering applications. Techniques based on graph traversal utilize adjacency information and obtain high compression for connectivity information. Topological surgery [TR98], cut border [GS98], and Edge-breaker [Ros98] are examples.

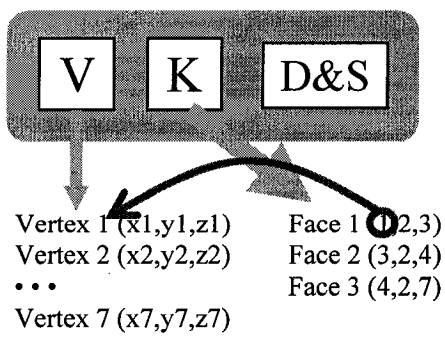


Figure 2.4 Traditional Triangle Meshes

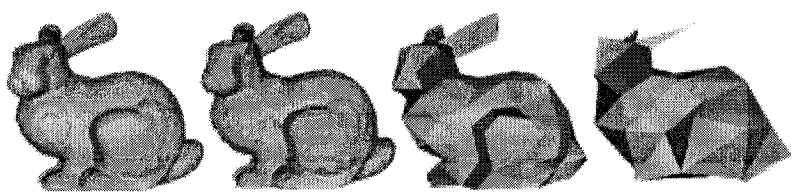
2.3.3 Mesh Simplification and Optimization

Mesh simplification is the construction of coarse meshes from non-optimal scanned meshes. It usually involves some preprocessing of the input data which may take a substantial amount of time. The primary goal of mesh simplification is to reduce the problem's complexity. The primary goal of *mesh optimization* is to construct a simplified mesh that accurately fits the model with a smaller number of vertices. Error metrics are derived to help decide the best geometry to approximate the shapes.

Mesh simplification algorithms generate several simplified versions of a geometric model, rather than compress the model in its original resolution. Most works in this area focus on the simplification of polygonal models. Previous mesh simplification works fall into two categories: multiple *static level-of-details (LODs)* or single LOD with *multi-resolution.* A general reference to LOD and mesh simplifications can be found in [LRC02].

2.3.4 Static LOD

The static LOD mesh simplification creates several versions of a mesh at various discrete level-of-details off-line (Figure 2.5). Initially it was proposed for accelerating the graphics rendering of polygonal models. There are two possible objectives in choosing an LOD: gain



69,451 polys 2,502 polys 251 polys 76 polys

Figure 2.5 Static LODs

the cheapest rendering time within a given error bound; and gain lowest error bound within a given rendering time. Typically 5 to10 LODs are created for each model. One LOD typically has half the resolution of the other. The LODs are switched based on projected area on screen or viewing distance. Models far from a view point are rendered in low LODs, those close to the view point are rendered in high LODs (Figure 2.6). Large projected area indicates closer distance, therefore high LODs are required to render [FS93]. Static LODs have some advantages over a highly detailed mesh in a single LOD. They decouple the simplification from rendering and simplify the programming model. Run-time rendering needs only pick the appropriate LOD. However, switching between discrete LODs brings some disadvantages. First, in a distributed environment, when LODs are switched at the client, multiple LODs are transmitted from the server over a network which leads to delay at the client. Second, instantaneous switching of LODs leads to perceptible "popping" artefacts. Third, multiple discrete LODs requires larger storage space at the server in comparison with the original mesh. Finally, this does not work for large models, where some parts may be very close, while some other parts are far away. Applications such as flying over a terrain require changing the level-of-detail locally, based on viewing direction. Since static LODs are created off-line, no viewing directions are considered in the mesh simplification process.

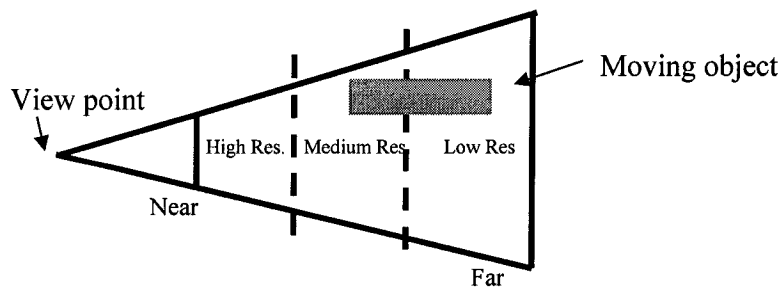


Figure 2.6 Switch LODs by Viewing Distance

Therefore, a static LOD is not a view-dependent LOD, but a view-independent LOD.

2.3.5 Multi-resolution and Continuous LOD

Multi-resolution: Multi-resolution mesh simplification creates a data structure that can be employed to dynamically produce a mesh with any desired resolutions lying between the highest and the lowest number of polygons from the original mesh at run-time. Whereas the static LOD mesh simplification creates several versions of a mesh at various discrete LODs [FS93]. Some multi-resolution mesh simplification algorithms refine or simplify meshes globally based on the distance to the view point and projected area on the screen. They are called *continuous LOD* [GSG96, Hop96, RB93, MBF+04]. Compared to the static LOD, the continuous LOD has the following advantages: (1) It allows a geometric model to be efficiently delivered over a network in a progressive manner; (2) Objects use no more polygons than necessary, which frees up polygons for other objects; (3) In rendering, the visual effect is smoothed with multiple resolutions at a finer granularity; (4) The storage requirement is significantly reduced due to the single multi-resolution data structure; (5) View-dependent mesh refinement is supported.

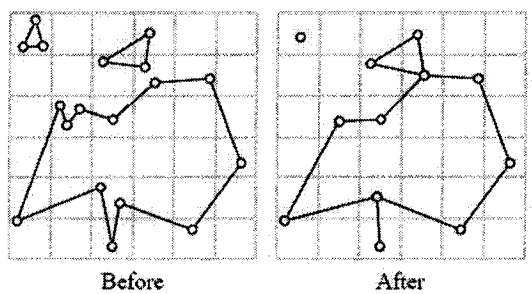


Figure 2.7 Vertex Clustering. All vertices within a grid cell are merged into one vertex.

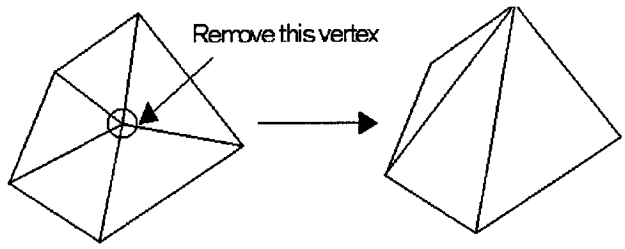


Figure 2.8 Vertex Decimation. At each iteration, a vertex is selected, its adjacent faces and itself is removed, the "hole" in the mesh is re-triangulated.

Multi-resolution Case Study

Mesh simplification for highly detailed geometric models is used in rendering, storage and transmission. There are also a lot of other uses for mesh simplification: collision detection at a different resolution than the one used for rendering; editing a model not in its highest resolution, but at an arbitrary level of detail, automatically applying the changes to the entire multi-resolution structure, and so on. Over the past few years, a tremendous amount of work has been done on mesh simplification and multi-resolution data structures. The following list contains some of the work that has been very important and influential, each being a representative of a distinct approach to solving the problem.

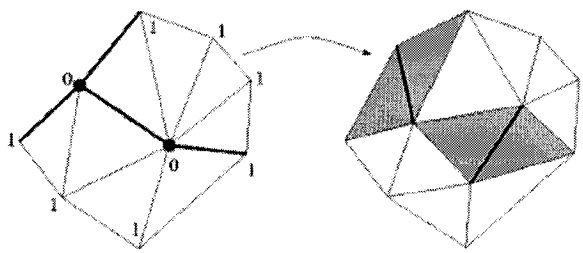


Figure 2.9 Example of Vertex Split. Vertices to be split are marked with 0.

Vertex Clustering [RB93]: Grouping vertices into clusters and determining a single representative vertex. Clusters are iteratively merged into larger clusters until only a single cluster/vertex remains. Vertices are clustered by simple coordinate quantization, octrees, etc. This approach is fast and simple. However, the output is often a very bad approximation of the original mesh. The degree of simplification is only controllable indirectly by quantization parameters (Figure 2.7).

Vertex Decimation [SZL92]: Vertices that pass a certain distance/angle criterion are iteratively removed. Resulting holes are patched by local re-triangulation (Figure 2.8). The approach is primarily developed for processing the output of Marching Cubes. It is limited to a manifold surface.

Edge Collapse [HDD+93]: Vertices connected by an edge are collapsed into a single representative vertex (Figure 2.9). This is the most often used primitive operation. The efficiency depends on the error metric employed.

Wavelets [EDD+95, GSG96]: This approach hierarchically decomposes arbitrary meshes to multi-resolution form. The method is based on the approximation of an arbitrary initial mesh by a mesh that has subdivision connectivity and is guaranteed to be within a specified tolerance. However, it has many constraints and requires many mesh preparation steps which may introduce error into the highest level of detail. Typically, it is very costly and difficult (Figure 2.12).

Contact LOD: Miguel A. Otaduy and Ming C. Lin proposed contact LOD in [OL03], a multi-resolution data structure for collision detection between two interacting 3D objects (Figure 2.10). Given a polyhedral model, the algorithm automatically builds a "dual hierarchy", both a multi-resolution representation of the original model and a convex hull BVH for accelerating collision queries (Figure 2.11). The generation of CLODs employs a local simplification operation that filters edge collapses subject to local and global convexity constraints. A cluster of triangles forms a part of a convex hull surface. The major benefit of using the convex hull as a BV is that when the refinement reaches certain CLOD, the contact information obtained from the BVs at that level approximates the contact information from the triangles of that CLOD. This is especially important when contact normals and contact points are required to compute a plausible collision response. The loss is that convex hull requires that the input models be 2-manifold oriented surfaces. At runtime, the algorithm uses error metrics to adaptively select the appropriate levels of detail independently at each potential contact location. Various error metrics are proposed which include object-space errors, velocity dependent gap, screen-space errors and their combinations. Compared to the existing exact collision detection algorithms, significant performance improvement is obtained on some benchmarks. However this algorithm does not refine a mesh at runtime

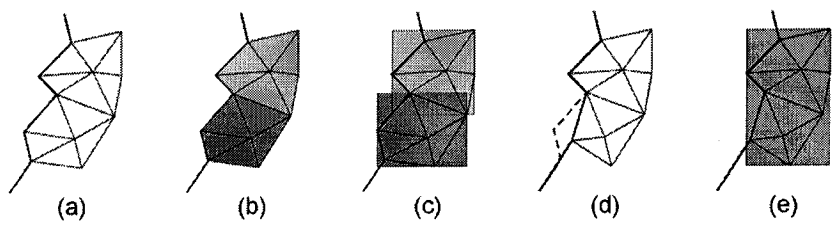


Figure 2.10 Creation of CLODs. (a) Initial surface; (b) Clusters of triangles; (c) Bounding volumes for each cluster (in this case AABBs); (d) Mesh simplification; (e) Bounding volume of the union of clusters, after some conditions are met.

using perception-based simplification criteria. Therefore, it does not provide a simplified mesh that can satisfy a growing requirement of perceptual realism very well. This is one of the problems we address later in this thesis. In addition, CLODs do not support progressive transmission. In a distributed environment, simulation can not be performed at the client until the whole CLODs data have been received. Therefore, CLODs are not appropriate for interactive distributed applications. In section 3, we propose a collision detection algorithm which performs collision queries on PM, a multi-resolution data structure supporting both progressive transmission and local refinement.

Quadric Error Metrics (QEM): This is a surface simplification algorithm that can be used in rendering systems for multi-resolution models [GH97]. It assumes surface models are represented as triangle meshes, points do not move far from their original positions, and the topology of the models need not to be maintained. A simplified mesh is a good approximation to the original mesh. The algorithm iteratively contracts vertex pairs (a generalization of edge collapse). As the algorithm proceeds, a geometric error approximation is maintained at each vertex of the model which is represented using quadric matrices. The algorithm proceeds until the simplification goals are satisfied.

The error approximation at a vertex v is defined as $\Delta(v) = v^T Q v$, where Q is a symmetric 4x4 matrix. The initial error approximation for each vertex is computed as follows. Let p = [a, b, c, d]^T represents the incident face planes of v. $ax + by + cz + d = 0$, and $a^2 + b^2 + c^2 = 1$. K_p is defined as 'fundamental error quadric'.

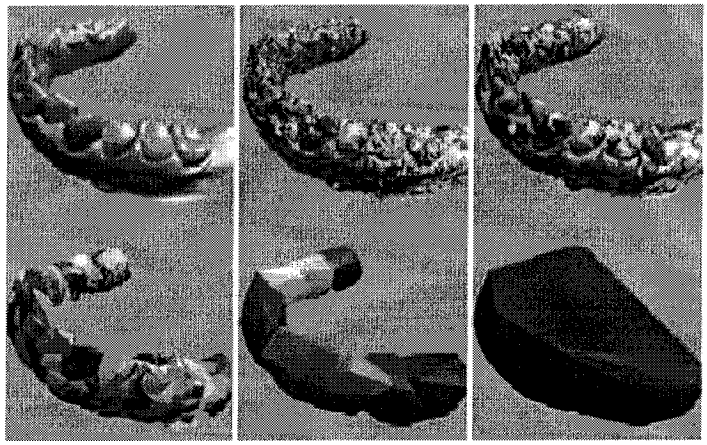


Figure 2.11 BVH of the Lower Jaw. From left to right and from top to bottom. LOD_0, and convex decompositions of LOD_0, LOD_3, LOD_6, LOD_{11}, and LOD_{14} (courtesy of M.A. Otaduy and M.C.Lin).

$$K_p = pp^T = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

The error approximation of v is defined as the sum of squared distances to the planes.

$$\Delta(v) = \Delta([v_x, v_y, v_z, 1]^T) = \sum_{p \in planes(v)} (p^T v)^2 = v^T \left(\sum_{p \in planes(v)} K_p \right) v$$

For the following pair contractions (v_1, v_2) -> v', the matrix Q' for v' is derived by Q'=Q_1+Q_2, where Q_1 is the matrix for v_1, Q_2 is the matrix for v_2.

Quadric Error Metrics have a nice geometric interpretation. Each quadric corresponds to a quadric surface centered at the point of minimal error. The surface itself (an ellipsoid in the non-degenerate case) has the same error everywhere, that is, it is an iso-surface for a given error.

Progressive Mesh: "Progressive Mesh" [Hop96] introduces a very efficient data structure that can be used to represent a triangle mesh at multiple levels of detail. It also allows progressive refinement and transmission. Hoppe posed the mesh simplification task as an energy minimization problem. He introduced a very high quality error metric using an energy function which combines a measure of the geometric similarity between an approximating mesh and the original model with a cost proportional to the number of data points in the approximating mesh. Many works for general polygonal models simplification are related to a progressive mesh (PM). The PM format is well suited to accelerating graphics rendering. A triangle mesh can be represented at multiple LODs by performing a series of refinement operations. The operations include vertex split and edge collapse. A triangle mesh is encoded as a base mesh plus a sequence of n vertex split records. The mesh can be refined from the coarsest level to the finest level or vice versa by performing the two operations. Another paper by the same author focuses exclusively on the view-dependent refinement of progressive mesh [Hop97] and defining efficient refinement criterion.

2.3.5.1 Progressive Techniques

Progressive techniques are aimed at speeding up transmission of geometric data. They

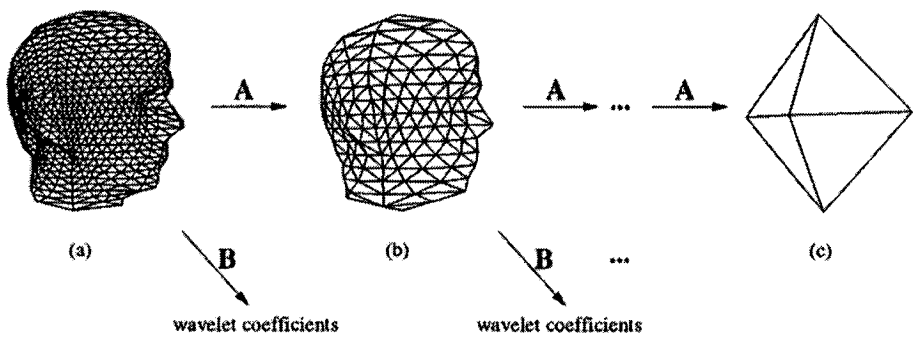


Figure 2.12 Wavelet Mesh Decomposition

provide progressive transmission of meshes. A coarse mesh approximation must be transmitted in a short time. The received initial approximation is refined through subsequent incremental modifications as more and more data are received. The techniques can be classified as to whether they are based on signal processing (or wavelet methods) and geometric techniques.

Wavelet methods: Wavelets, with their roots in signal processing and harmonic analysis, have had a significant impact in several areas of computer science. The development of wavelets has been motivated primarily by the need for fast algorithms to compute compact representations of functions and data sets. They exploit the structure (if any) in the data or underlying function, reorganizing it in a hierarchical fashion. The wavelet approach to progressive transmission of 3D surface meshes is basically the same as that used for curves, images, or any signal. The basic idea is to treat the model as an approximation, which can be updated with wavelets. The approach iteratively decomposes the original, or high-resolution, surface into a low-resolution part and a detail part. It end up with the coarsest representation of the original surface, along with a sequence of details which can be used to reconstruct the original. At runtime, a base mesh is sent first then the wavelet additions are sent until the desired level of detail is achieved at the receiver (Figure 2.12). This approach can provide very simple base mesh and efficient compression of the mesh. However, the wavelet mathematics is not trivial.

2.3.5.2 Geometric Techniques

There are two classes of geometric techniques. *Fine-grained* progressive techniques: based on modifications that update a few triangles at one time producing more levels of detail. Progressive Meshes (PM) [Hop96] is an example. *Coarse-grained* progressive techniques:

based on batches of modifications which are encoded at the same time; they usually lead to more compact bit streams. Progressive forest splits [TGH+98], progressive technique based on vertex insertion [CLR99], and Compressed Progressive Meshes (CPM) [PR00] are examples.

Progressive Meshes (PM): A triangle mesh is encoded as a coarsest mesh (or base mesh) plus a sequence of n vertex split records ($M_0, \{vsplit_0, \ldots, vsplit_{n-1}\}$). At runtime, a PM is transmitted progressively with a M_0 followed by a sequence of $vsplit_i$ as shown in Figure 2.13. The receiver refines the mesh progressively as more and more *vsplit* records are received. The mesh is refined from the coarsest level to the finest level and forms a sequence of meshes called a *PM sequence*, $M_0 \xrightarrow{vsplit_0} M_1 \xrightarrow{vsplit_1} \ldots \xrightarrow{vsplit_{n-1}} (M_n = \hat{M})$. Performing a vertex split requires inserting two new vertices v_s, and v_t, and adding two triangles (v_s, v_t, v_l) and (v_s, v_r, v_t). An edge collapse is an inverse operation of vertex split as shown in Figure 2.14. The coarsest mesh is obtained by performing a sequence of edge collapses on the full mesh. PM is a fine-grained technique since at each refinement at most two triangles are inserted in

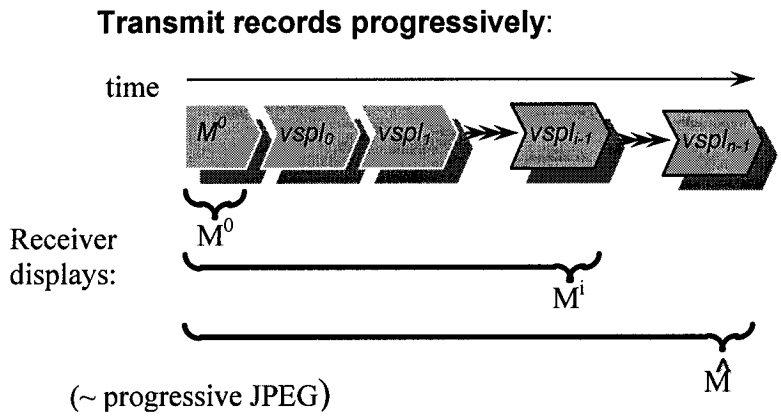


Figure 2.13. Progressive Mesh Transmission

a manifold domain. A traditional representation of mesh geometry with n vertices requires storage of $3n$ coordinates or $96n$ bits with IEEE single-precision floating point. In [Dee95], vertex coordinates are quantized to 16-bit fixed precision values without significant loss of visual quality, thus reducing storage to $48n$ bits. Further compression by delta-encoding the quantized coordinates and Huffman compressing the variable-length deltas can reduce storage to $35.8n$ bits. PM uses a similar approach. Experiments from Hoppe show that the space requirement is $30n$ to $37n$ bits.

Compressed Progressive Meshes (CPM): A triangle mesh is encoded as a coarse mesh M_0 plus a sequence of sets of vertex splits, called batches, which can affect the whole mesh. In the PM approach, $\log n$ bits are used to encode the index of the vertex which must be split. In the CPM approach, at the i-th refinement step, each vertex of the current mesh M_i is marked with one bit to specify whether it must be split or not. The number of bits used to encode the index of a vertex is relatively small due to large refinement steps.

2.3.6 View-dependent LOD

View-dependent LOD: Some multi-resolution mesh simplification algorithms extend continuous LOD to support local refinement of the meshes based on viewing directions or moving directions relative to other models in a scene. They are called *view-dependent LODs*.

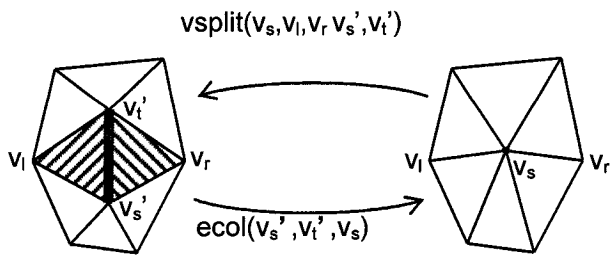


Figure 2.14 Vertex Split and Edge Collapse

Compared to view-dependent LOD representations, continuous LOD representations have more evenly distributed 3D points. Continuous LOD representations are in uniform resolutions and view-dependent LOD representations are in non-uniform resolutions. Comparing with continuous LOD representations, view-dependent LOD representations have some advantages. First, they have better granularity. View-dependent LODs allocate polygons where they are most needed, within as well as among objects, therefore enabling even better fidelity. Second, they enable drastic simplification of very large objects, such as the stadium model and terrain model (Figure 2.15), etc.

Normally, a multi-resolution representation needs more space than a single resolution representation, because not only does the original mesh need to be stored, but also additional information for all other levels of detail. However, if the only mesh stored in its entirety is the lowest-resolution mesh (or base-mesh) and all other levels are stored as deltas from the base-mesh, then it can also be used for compression: delta encoding. At least, the storage

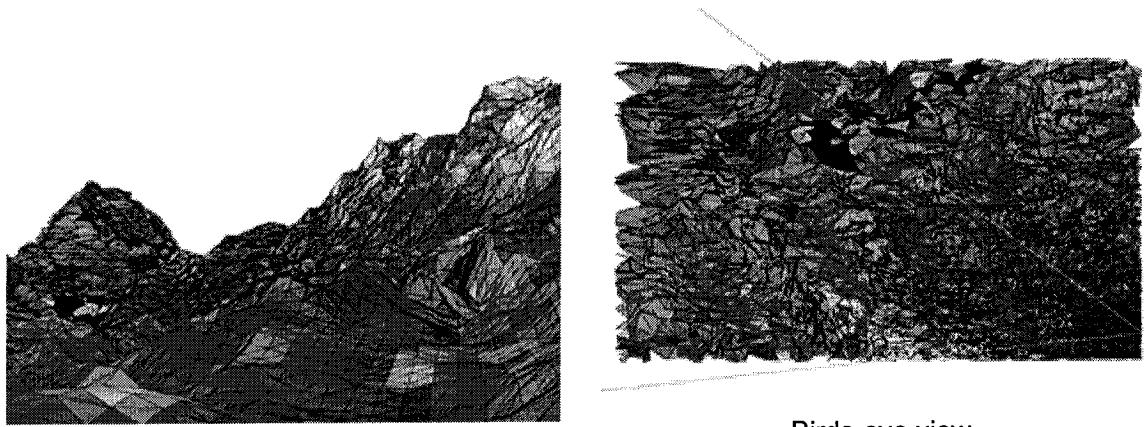


Figure 2.15 Illustration of View-dependent LOD. Nearby portions of object is at higher resolution than distant portions

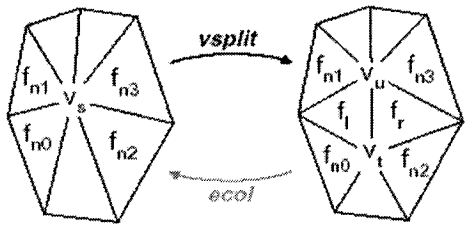


Figure 2.16 Redefinition of vsplit and ecol Transformations.

requirements of a multi-resolution representation need not be higher than those of a high-resolution mesh. Several mesh simplification techniques and multi-resolution data structures are introduced in the following section.

Recent works on view-dependent simplification take into account viewing parameters in mesh simplification to speed-up graphics rendering further. Xia et al [XEV97] introduced a merge tree which is built upon progressive meshes to enable real-time view-dependent rendering of objects. The vertex split and edge collapse operations for an object are organized in a hierarchical manner. Hoppe [Hop97] developed a view-dependent simplification algorithm for progressive meshes which use screen space projection and viewing orientation to guide the runtime simplification. Luebke and Erikson [LE97] defined a tight octree, called vertex tree, over a given model to generate hierarchical view-dependent simplifications. When the screen-space projection of a cell in the octree is too small, all the vertices in that cell are collapsed to one vertex. Taubin et al [TG98] demonstrated a surface partition scheme for progressive encoding of surface. Schilling and Klein [SK98] introduced a texture dependent refinement algorithm.

2.3.6.1 View-dependent Refinement Case Study

View-dependence solves the problem of fast rendering of large models by non-uniformly refining the meshes. It is integrated into the PM framework by only performing those vertex-split operations required for a certain screen-space error. Because the mesh operations cannot

be performed independently, a PM hierarchy tree needs to be considered instead of a simple list of vertex split records (Figure 2.17). It has been proven useful to redefine the ecol/vsplit transformation, as shown in Figure 2.16. The transformation *vsplit*(v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3}), replaces a parent vertex v_s by two children v_t and v_u. Two new faces f_l and f_r are created between the two pairs of neighboring faces (f_{n0}, f_{n1}) and (f_{n2}, f_{n3}) adjacent to v_s. The edge collapse transformation *ecol*(v_s, v_t, v_u, ...) has the same parameters as *vsplit* and performs the inverse operation. To support meshes with boundaries, face neighbors f_{n0}, f_{n1}, f_{n2}, f_{n3} may have a special *nil* value, and vertex splits with f_{n2}= f_{n3} =*nil* create only the single face f_l. A set of preconditions for *vsplit* and *ecol* to be legal is defined and a set of properties of the view-dependent mesh refinement is given. The correctness of the preconditions and the properties has held for the numerous experiments, but they have not been proved formally as yet.

Preconditions

A *vsplit*(v_s, v_t, v_u) is legal if

(1) v_s is an active vertex, and (2) the faces $\{f_{n0}, f_{n1}, f_{n2}, f_{n3}\}$ are all active faces.

A *ecol*(v_s, v_t, v_u) is legal if

(1) v_t and v_u are active vertices, and (2) the faces adjacent to f_l and f_r are $\{f_{n0}, f_{n1}, f_{n2}, f_{n3}\}$.

Note that a vertex or a face is active if it exists in the selectively refined mesh.

Some criteria have been proposed in [Hop97] to decide whether a vertex should be split or not, which include viewing volume, surface orientation, and screen-space error. A vertex obviously doesn't need to be split if it lies outside the viewing volume or belongs to a back-facing region. If the vertex is visible from the current viewpoint, an estimation of the screen-space error introduced by the corresponding edge collapse is needed. Hoppe extends the metric used in [LKR+96] and defines an error volume called *deviation space,* whose

projection to screen is used for the screen-space error test. All these conditions are packed into a single function *qrefine* which returns *true* if a vertex should be split and *false* otherwise.

2.4 Collision Detection

Collision detection is a fundamental problem in computer animation, physically-based modeling, computer simulated environments and robotics. In a geometry context, collision detection is a system which samples object motions in a time interval, and then queries and reports information about the relative configuration or placement of two objects. The system should be easily interfaced with a variety of applications. Two common examples of collision queries are:

Interference detection: Checks whether two objects overlap in space or whether their boundaries intersect.

Separation distance computation: Computes the minimum Euclidean separation distance between the boundaries of two objects and reports if it is more or less than a given value.

Each collision query can be augmented by adding the element of time. If the trajectories of two objects are known, then the next time to sample can be determined at which a particular query (interference, or separation distance) will become true. These queries are called

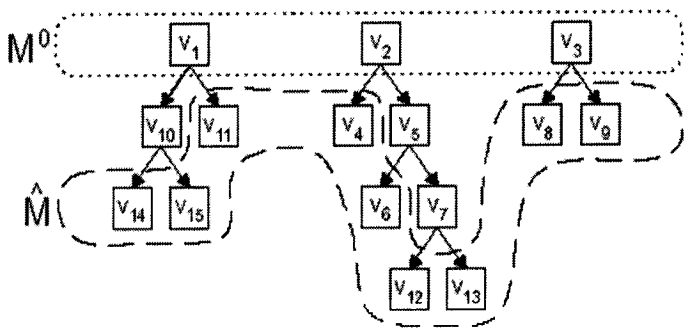


Figure 2.17 Illustration of the Vertex Hierarchy for PM. The vertex hierarchy on the set of vertices in all meshes of the PM sequence forms a "forest", in which the root nodes are the vertices of the coarsest mesh (base mesh M_0) and the leaf nodes are the vertices of the most refined mesh (original mesh $\hat{M}$).

dynamic queries. The ones that do not use motion information are called *static queries*. In the case where the motion of an object cannot be represented as a closed-form function of time, the underlying application often performs static queries at specific time steps.

2.4.1 Large-scale Virtual Environments

The abovementioned collision queries only apply to pairs of objects. In a large-scale virtual environment which consists of multiple objects, collision detection becomes a $O(n^2)$ problem where n is the number of objects. Some methods have been proposed to overcome this bottleneck. A common solution to interactive applications is to divide collision detection into two phases [Hub96]: a broad phase, in which pairs of objects on which collision potentially will happen are identified, and a narrow phase, in which exact pair-wise queries are performed (Figure 2.18).

For the broad phase, a simple approach is *space division*. Space is equally divided into cells and an object is assigned to one or more cells. Collisions are checked between object pairs belonging to the same cell. In [Tur90, ZOM+93] a regular grid is used to identify objects that are close to each other. Choosing a near-optimal cell size is difficult, and failing to do so results in large memory usage and computational inefficiency. Overmars uses a hash table to efficiently perform point location queries in fat subdivisions [SHO91, Ove92]. Subdivisions

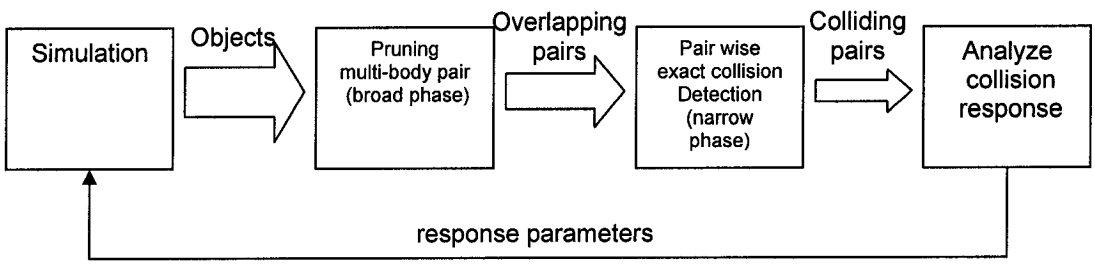


Figure 2.18 Two phases multi-body collision detection

are called fat if no cell has long and skinny parts. This approach does not work well in environments where objects are not uniformly distributed in the space. A recent approach is *sweep-and-prune* [Bar92, CLM+95], which exploits inter-frame coherence in a dynamic environment to efficiently sort bounding boxes, identifying those that intersect. It incrementally computes an axis-aligned bounding box (AABB) for each object and checks them for overlap by computing the projection of the bounding boxes along each dimension, and sorting the interval endpoints using insertion sort or bubble sort. The number of pair wise collision queries is reduced to $O(n+m)$ where n is the number of objects and m is the number of objects very close to each other. This approach is limited to environments where objects undergo small movements. Another approach is *swept volume* [Cam90]. It operates on a 4D volume which is swept out by object motion over time. If the swept volumes for a pair of objects do not intersect, then no collision will occur between them. Due to the computationally expensive generation of the swept volume, many works deal with it by using convex approximations and piecewise translation motions. In cases where object trajectories are expressed as functions of a parameter (time), collision instants can be determined analytically. However, when the degrees of the polynomial functions are arbitrarily high, the determination of the collision instant becomes computationally very expensive. In [Can86], the problem is tackled by designing a trajectory connecting two arbitrary configurations for a moving polyhedron, so that the obtained polynomials are of degree 3. It is the lowest degree supporting translation and rotation simultaneously. The exact points of collision are computed. In the narrow phase, all pairs of objects identified in the broad phase require applying exact collision detection between either 3D volumes or 4D ones. Different set of algorithms can be classified based on query type (interference

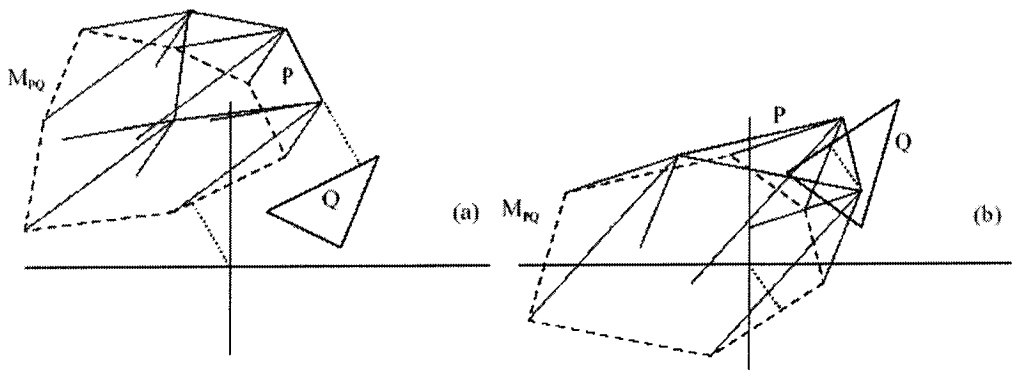


Figure 2.19 The Minkowski Difference M_{PQ} (dashed line) can be constructed by taking the convex hull of the points resulting from subtracting the vertices of Q from the vertices of P. (a) The distance between P and Q (dotted line) is the same as that from the origin to M_{PQ}. (b) If P and Q are interfering, the origin will be within M_{PQ}.

detection, separation distance), temporality (static, dynamic), and representation (convex polytope, general polygonal model, curved model represented by implicit, parametric, or NURBS surface). Object representation plays a very important role in the performance of collision queries. Therefore it is used in the following sections as a criterion to classify the algorithms.

2.4.2 Convex Polyhedra

Most algorithms used to detect interferences between convex polyhedra rely on the computation of the minimum distance. When the separation turns out to be zero, the interference is detected implicitly. A number of algorithms with good asymptotic performance are given below.

Linear programming: It is proved in [LC91] that intersection detection for two convex polyhedra can be done in linear time, in the worst case by reduction to linear programming. Two convex polyhedra do not overlap if and only if there is a plane separating them. The three parameters that define the plane are treated as unknown. The algorithms check whether

there is a feasible solution to the constraints that all vertices of one polyhedron lies in one half space of the plane and those of the other polyhedron lie in the other.

Dobkin-Kirkpatrick (DK) hierarchy: Convex polyhedra can be properly preprocessed to make the complexity of intersection detection drop to $O(\log n \log m)$ [Dom90], where n and m are the numbers of features of a pair of convex polyhedra. Preprocessing takes $O(n+m)$ time to build a DK hierarchy to each polyhedron. If the topology of the polyhedra does not change, the hierarchies need to be computed only once.

Minkowski difference: It is defined as $M_{P,Q}=\{p-q \mid p \in P, q \in Q\}$ where P and Q are two polytopes. $M_{P,Q}$ has been used in distance computation in [Cam86, Cam97], exploiting the fact that the distance between P and Q is equal to that from $M_{P,Q}$ to the origin (Figure 2.19). While the complexity of computing the $M_{P,Q}$ is $O(n^2)$, a fast algorithm similar to DK that exhibits linear time performance in practice is proposed in [GJK88].

Tracking closest features: Separation distance is computed by navigating along the boundaries of involved polyhedra in the direction of decreasing distance. An incremental algorithm for distance computation by using Voronoi regions is proposed in [LC91]. Every

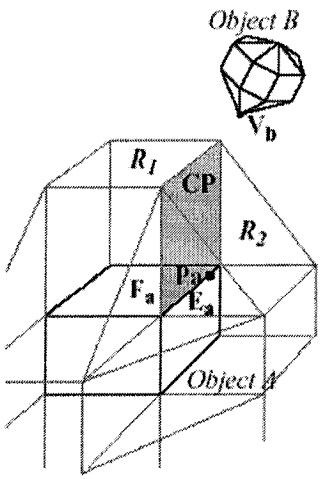


Figure 2.20 A Walk Across the External Voronoi Region of Object A. Vertex Vb of Object B lies in the Voronoi region of Ea.

feature of a polyhedron has associated one such region, consisting of all the points that are closer to it than to any other feature. The algorithm first pre-computes the Voronoi region for each external feature in linear time. At each time step, it starts with a pair of features and checks whether they are the closest features based on whether they lie in each other's Voronoi region. If not, it performs a local walk to neighbor features until it finds the closest features (Figure 2.20). In applications with high motion coherence, the local walk typically takes roughly constant time in practice. Ehmann and Lin proposed the "multi-level Voronoi Marching" algorithm in [EL00], which modified this algorithm and used an error bounded LOD hierarchy to accelerate collision query, using a progressive refinement framework.

2.4.3 General Polyhedra

General non-convex polyhedra are more difficult to handle. A few works try to cope directly with non-convex polyhedra without decomposing them [BEG+04]. Most researchers resort to decomposing them or their boundaries into convex parts (convex polyhedra and polygons respectively) and to apply collision queries to those parts. In any case, a number of additional fictitious entities are created which have to be considered in the intersection tests. Typically, decomposition is performed in a preprocessing step, and has to be computed only once, if the topology of the polyhedra does not change. The minimum decomposition problem is known to be NP-hard [BD92]. The solutions can be classified based on whether the object is represented as a closed polyhedral model which forms a solid, or is represented as a collection of polygons without assumptions related to the connectivity among different faces, or whether they represent a closed set. The strategy of the latter is to focus the search for collisions on relevant portions of the objects. The approach is to apply a hierarchical bounding to approximate a volume or a boundary representation. *Space partitioning* and

bounding volume hierarchies (BVHs) representations for collision detection are introduced next.

2.4.3.1 Space Partitioning

The spatial partitioning is a recursive subdivision of the space occupied by an object. Octrees [HH96], kd trees and their variants [Sam89], BSPs [NAT90], and regular grids are examples of spatial partitioning hierarchies. Octrees and BSPs are most widely used. Octrees recursively partition cubes into octants. The octree representation makes it possible to avoid checking for collisions in those parts of space where octants are free of objects. BSP-trees recursively cut the space by hyperplanes. This can be considered a cross between octrees and boundary representations, in which partitioning is not restricted to be axis-aligned, as in octrees. Therefore, transformations in space can be simply applied to each hyperplane without rebuilding the whole representation.

2.4.3.2 Bounding Volume Hierarchies (BVHs)

BVHs are based on a recursive partitioning of the primitives of an object. BVHs use bounding volumes (BVs) to bound or contain sets of geometric primitives, such as triangles, polygons, curved surfaces, etc. In a BVH representation, the geometry BVs are stored at the nodes of a tree structure. The root BV bounds all the primitives of a model, and children BVs each bound separate partitions of the primitives enclosed by the parent. Leaf nodes' BVs typically contain one primitive each. In some variations, one may place several primitives at a leaf node or use several volumes to contain a single primitive. BVHs are used to perform interference and separation-distance queries. Sphere-trees [Hub96, BO04], oriented bounding boxes (OBB) trees [GLM96], axis-aligned bounding boxes (AABB) trees [Ber97], swept sphere volume [LGL+00], spherical shell-trees [KPL+98], discrete orientation

polytopes (k-dops) trees [KHM+98], SSV-trees [LGL+99], multi-resolution hierarchies [OL03], convex-hull trees [EL01], and CHPM [YSL+04] are examples of BVH representations.

Collision queries:

Interference queries are performed by traversing the BVHs. Two models are compared by recursively traversing their BVHs in tandem. Each recursive step tests whether BVs A and B, one from each hierarchy, overlap. If they do not, the recursion branch is terminated. If A and B overlap, the enclosed primitives may overlap and the algorithm is applied recursively to their children. If A and B are both leaf nodes, the primitives within them are tested directly.

Separation-distance queries are performed similarly to the interference queries. As the query proceeds, the smallest distance found from comparing primitives is maintained in a variable δ. At the start of the query, δ is initialized to ∞, or to the distance between an arbitrary pair of primitives. Each recursive call with BVs A and B must determine if some primitive within A and some primitive within B are closer than, and therefore will modify δ. The call returns trivially if BVs A and B are farther than the current δ, as this precludes any primitives within them being closer than δ. Otherwise the algorithm is applied recursively to its children. For leaf nodes it computes the exact distance between the primitives, and if the new computed distance is less than δ, it updates δ.

Performance of BVHs:

The abovementioned approach is based on the divide-and-conquer paradigm. There are two main advantages in the approach: (1) In many cases, interference or non-interference situations can be easily detected at the first level in the hierarchy; (2) the refinement of the

representation is only necessary in the parts where collision may occur. The creation of a BVH representation must follow three criteria: (1) the bounding volumes need to tightly approximate the input primitives of an object; (2) an intersection test between two bounding volumes needs to be fast; (3) the refitting of the bounding volumes and the hierarchical structure needs to be fast when the geometric primitives are rotated or translated, or when the topology of the geometric representation is changed.

The performance of BVHs on collision queries is governed by a number of factors (techniques to build the trees, the maximum number of children per node, the choice of BV type). Some of the commonly used algorithms assume that the BVHs are binary trees and each primitive is a single triangle or a polygon. The cost of performing a collision query is given in [GLM96] as:

$$T = N_{bv} \times C_{bv} + N_p \times C_p ,$$

where T is the total cost for a collision query, N_{bv} is the number of BV pair operations, and C_{bv} is the total cost of a BV pair operation. N_p is the number of primitive pairs tested for proximity, and C_p is the cost of testing a pair of primitives for collision. Typically, for tight-

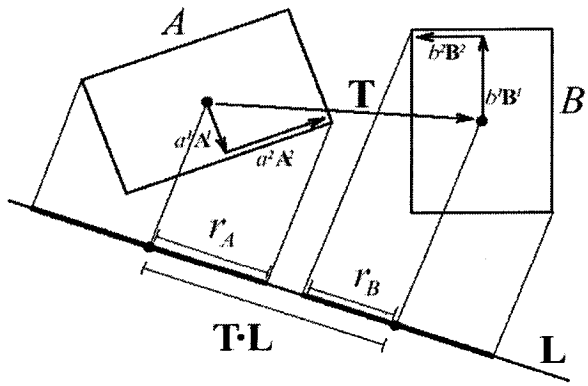


Figure 2.21 OBBs Overlap Test. Under projection onto L, A and B become disjoint intervals, therefore L is a separating axis for OBBs A and B

fitting bounding volumes, e.g., OBBs, N_{bv} and N_p are relatively small, whereas C_{bv} is relatively high. In contrast, C_{bv} is low while N_{bv} and N_p may be larger for simple BV types such as spheres and AABBs. Due to the opposing trends, no single BV yields optimum performance for collision queries in all situations.

Queries on BVs:

- In cluttered environments, OBB trees perform better than AABB trees or sphere trees. OBBs fit objects tighter, therefore fewer intersections between bounding volumes are reported. In [GLM96] an efficient algorithm to test two OBBs for overlap based on the separating axis theorem (SAT) has been presented. It computes the projection of each OBB along 15 axes in 3D. The 15 axes are computed from the face normals of the OBBs (6 face normals) and by taking the cross products of the edges of the OBBs (9 cross products). It is shown that two OBBs overlap if and only if their projection along each of these axes overlap (Figure 2.21). In practice, it can take anywhere from 80 to 240 arithmetic operations to check whether two OBBs overlap. Routines for building OBB trees, as well as for performing fast overlap tests between them can be found in the RAPID interference detection package.
- Although AABBs cannot fit some primitives like long-thin oriented polygons tightly, the overlap test of AABBs performs better than that of OBBs. A BVH based on AABBs has the advantage that the overlap test between each pair of AABB trees is not orientation dependent, as is the case of OBB trees. In other words, the boxes in AABB trees need to be projected on the coordinate axes only once. For each pair of boxes of OBB trees undergoing an overlap test, one box has to be projected onto the axes of the other one. The separation distance between them can be computed based on the

separation along each axis. Furthermore, AABB trees need less memory and are faster to build, and are even faster to update [Ber97], which makes them suitable for deformable models. SOLID is a library for collision detection between polygonal models using AABB trees. Hierarchies of AABBs are also used in the context of self-intersections and collisions between deformable models [HDL+96, VT94, LM01].

- k-dops are BVs that are convex polytopes whose facets are determined by half-spaces with outward normals coming from a small fixed set of k orientations. The overlap test for k-dops is similar to the test for AABBs, in which the projections of a k-dops are checked along the k fixed axis. A k-dops tree is a type of BVH representation. This is a compromise between the relatively poor tightness of AABBs and bounding spheres and the relatively expensive overlap tests and updates of the OBBs and convex hulls. We will employ both AABBs and OBBs data structures in our approach and study their performance.

2.4.4 Real-time Collision Detection

In Computer Graphics, research emphasis has been placed on the possibility of detecting collisions in real-time, even if speed is gained at the cost of losing precision. This problem has been well studied in the literature. Some strategies have been developed to lower the cost of collision detection.

The first is to exploit a BVH. The BVH provides higher precision as one moves down the hierarchy and allows output precision to be adapted to the computing time available [Hub96, BO04]. However, spatial inaccuracy is inherent due to the lack of exact collision queries on primitives. This becomes a significant limitation when applying the BVH CD to time-critical

applications in which contact normal and contact points are required to compute a plausible collision response in physics simulation.

The second is to determine forefront features in the direction of motion. *Back-face culling*, which has been used in Computer Graphics to speed up the rendering of polyhedra, can also be used in the collision detection context to avoid unnecessary checking of boundary elements for collision. The basic idea is to cull those faces of a pair of polyhedra whose normal vectors have negative projections on the relative motion vectors. Half of the faces are eliminated on the average. *Applicability constraints* [Don87] permit detecting the vertex-face and edge-edge pairs of two polyhedra that can actually come into contact under translational motions.

The third is to exploit the motion coherence to keep track of the closest feature points. At a given instant, the boundary elements of the polyhedra that realize the minimum distance must be close to those realizing it at the previous instant, which are therefore taken as initial points for the search [JTT01]. This is known as temporal and geometric coherence or motion coherence. The incremental minimum distance calculation technique [LC91] has been discussed in section 2.4.2, in which a neighborhood criterion is used to save computational effort. A collision detection library *I_COLLIDE* based on this technique has been developed. The *Separating vector* algorithm efficiently determines whether there exists a separating plane between two convex polyhedra [CW96]. If so, they do not collide. Otherwise, the situation at the previous instant is examined. The motion coherence is exploited, so that the separating plane can be determined in expected constant time. The abovementioned two techniques are limited to convex polyhedra.

2.4.5 Other Collision Detection Methods

Most of CD algorithms support static LOD, which means that the mesh geometry and topology are static at runtime. In DVEs, they are predetermined by the lowest available network bandwidth and the accuracy of CD required by the application. A few recent works have been focused on using multi-resolution representations in haptic CD. Pai and Reissel [PR97] investigated the use of multi-resolution image curves for 2D haptic interaction. El-Sana and Varsheny [SV00] introduced a multi-resolution hierarchy. A detailed mesh is used for regions around probe pointer and a coarser mesh is used elsewhere. A locally refined method is introduced in [LSG+05] to perform fast collision queries among probe pointers and other objects in a scene. Otaduy and Lin [OL03] developed an algorithm to decompose a surface model into convex pieces and construct a multi-resolution BVH for fast collision queries of an object-object pair. However, that algorithm does not refine a mesh at runtime following perception-based simplification criteria. The above mentioned algorithms do not support progressive transmission. Therefore they do not support interactive DVE in low bandwidth network.

Collision detection on deformable objects is important to some applications such as clothes simulation [BFA02, CD97, RKL+04], hair simulation [WFL02] and articulated models [GNR+02, JP04, KAA+03, LC02, LSH+02]. A summary of previous works can be found in [TKH+05]. Using graphics hardware to prune the number of objects that are in close proximity and check for overlapping triangles between objects is proposed by Govindaraju et al.[GRL+03]. It works well for non-rigid objects. CULLIDE is extended to perform intra-object or self-collisions between complex models in [GLM05]. However, accuracy of hardware collision detection is limited by the resolution of the rasterization hardware.

2.5 Summary

Collision detection is considered to be one of the major bottlenecks in building interactive and realistic virtual environment. Some strategies have been exploited to reduce the cost of collision queries on polygonal surface models. One is to introduce BVH representation of models in time-critical collision detection. Another is to perform collision queries on simplified approximation of the models. Some recent surveys of these works can be found in [JTT01, LG98]. However, no good real-time collision detection systems are known for distributed and interactive virtual environments. We are seeking to overcome the shortcoming of the existing strategies and propose new strategies to speed up collision detections in the context of distributed virtual environment.

Chapter 3 Multi-resolution Collision Detection

Collision detection is accelerated to an interactive rate by setting the meshes to a low resolution. Performing collision detection on meshes in low resolution saves storage space and transmission bandwidth, but loses accuracy. On the other hand, the graphics display of computers is more and more powerful with the integration of Graphics Processing Units (GPUs) into computer graphics cards. GPUs are capable of manipulating and displaying millions of polygons per second. The detail and the complexity of a scene can increase without affecting the performance of CPU. In practice, coarse meshes for collision detection are decoupled from fine meshes for visual rendering, which leads to an inconsistency between realistic visual display and inaccurate simulation. Therefore, setting the resolution of the meshes for collision detection in a virtual environment application becomes an engineering trade off between speed and accuracy. In this thesis we propose a multi-resolution collision detection approach for distributed virtual environments which benefits from recent advances in mesh coding techniques.

The *continuous LOD* technique makes it possible to change the cost of collision detection dynamically. At runtime, the full meshes, which are stored in secondary storage locally or remotely, are progressively loaded to main memory and dynamically refined to higher or lower resolutions. Certain refinement criteria have to be set in order to maintain the collision detection at interactive rate. The *progressive techniques* make it possible to transmit the meshes for collision detection on demand of accuracy. In a distributed environment, clients may have varied computing power. More powerful clients can subscribe more detailed

meshes from the server and gain more accuracy in physics simulations. Instead of performing ubiquitous non-realistic physics simulations on all clients, the performance of both high ends and low ends is maximized.

A multi-resolution collision detection approach is composed of three tasks:

1. Multi-resolution mesh modeling.

2. Definition of a multi-resolution data structure for mesh refinement.

3. Creating a BVH collision query algorithm for multi-resolution models.

The following sections discuss the design and implementation issues of the three tasks. How they are fitted into the DVE frame work is explained later.

3.1 Multi-resolution Mesh Modeling

3.1.1 CDPM Representation

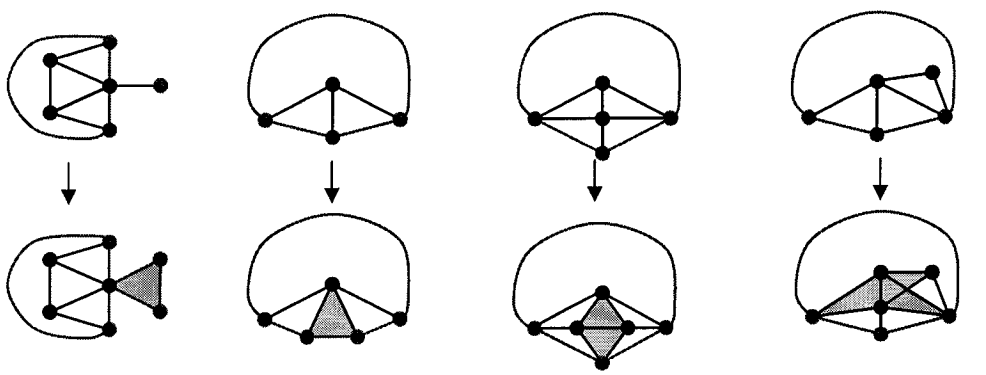


Figure 3.1 Non-manifold (a, d) and Manifold Triangle mesh (b, c) Vertex Split and Edge Collapse. (Faces in grey are generated by vertex split operations).

The proposed CDPM modeling method involves the use of the progressive mesh and the Quadric Error Metrics (QEM). The QEM is a method we use to efficiently generate CDPM from traditional triangle meshes in arbitrary topology by performing two operations, vertex split and edge collapse [GH97]. A vertex split operation splits one vertex v_s' to two vertices v_s and v_t, and adds several new faces F_1, F_2, ... F_n (Figure 3.3). The edge collapse operation is an inverse of the vertex split. In this thesis, a vertex split operation is denoted as VSP. A set of VSP is denoted as VSPs. The *i*th VSP is a sequence of VSPs is denoted as VSP*i*. The input to the QEM algorithm is non-manifold triangle meshes. A manifold polygonal surface is such that the neighborhood of every vertex can be continuously deformed to a disk (to a half disk at the boundary). Many real-world polygonal meshes are *non-manifold*, that is, contain topological singularities, (e.g., edges shared by more than two triangles) [GBT+99]. Figure 3.1 illustrates mesh refinement operations. Meshes in (a) and (d) are non-manifold while those in (b) and (c) are manifold.

The CDPM representation is similar to PM. An advantage of PM representation over some other multi-resolution representations is that PM supports progressive transmission. A CDPM mesh can be streamed over network by progressively transmitting vertex split records

Header (*H*)	**Coarse mesh(M_0)**	**Vertex split record *i* (VSP_i) (*i* from 1 to *n*)**
# vertex in full mesh # face in full mesh # vertex split record	vertex list: $\{v_1, \ldots v_c\}$ primitive index list: $\{f_1, v_{11}, v_{12}, v_{13}, f_{1leaf}\} \ldots$ $\{f_d, v_{d1}, v_{d2}, v_{d3}, f_{dleaf}\}$	i $\{v_s, v_t\}$ $\{dx_s, dy_s, dz_s\}$ $\{dx_t, dy_t, dz_t\}$ $\{v_1, v_2, \ldots, v_m\}$ $\{leaf_1, leaf_2, \ldots, leaf_m\}$ $\{p\}$ $\{f_1, f_2, \ldots f_p\}$ $\{f_1', f_2', \ldots f_q'\}$

Figure 3.2 CDPM Mesh Structure

in sequence. The format of CDPM is illustrated in Figure 3.2. Statistical information of the mesh is stored in *Header*. Here *#vertex* records the number of vertices in the mesh of full resolution, denoted by M_n, *#face* is the number of faces in the M_n, *#vsplit* is the number of vertex split operations required to refine the mesh from the coarsest mesh M_0 to the finest mesh M_n. Finally, *coarse mesh* stores the vertex array for c vertices and primitives index list of d primitive (polygonal faces) of M_0. Each primitive contains a list of vertex indices and the code of the primitive in the BVH which is explained later. It is followed by a sequence of vertex splits records. In *vertex split record i,* VSP_i, v_s and v_t are the indices of two new vertices split from v_s'. i is from 1 to n, where n is the total number of vertex split record. In this case $\{dx_s, dy_s, dz_s\}$ and $\{dx_t, dy_t, dz_t\}$ represent the differences in coordinates of the two new vertices from v_s',$\{v_1, v_2, \ldots v_m\}$ are vertex indices of the generated new faces besides v_s and v_t. Finally $\{f_1, f_2, \ldots f_p\}$ and $\{f_1', f_2', \ldots f_q'\}$ are the indices of adjacent faces to vertex v_s' which are split by the new generated faces to two groups. Faces in the first group are adjacent to v_s while faces in the second group are adjacent to v_t. P is the number of faces in the first group. The structure of the BVH built on M_n is encoded in the vertex split records. This means that $leaf_1, \ldots, leaf_m$ are the indices of leaves in the BVH tree structure which contain a BV of the newly generated faces.

3.1.2 Mesh Generation

It is proposed to use the CDPM for BVH collision detection of polygonal models in DVEs. The process of mesh generation can be divided into three phases.

3.1.2.1 Mesh Simplification

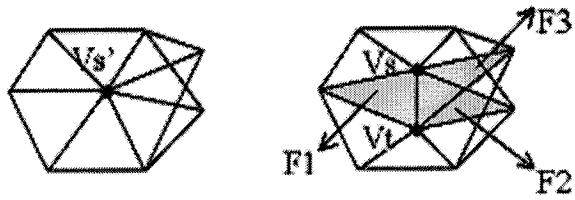


Figure 3.3 Vertex Split Operation. One vertex Vs' is split to two new vertices Vs and Vt. Left: before; right: after.

In phase 1, a QEM mesh simplification algorithm is applied to polygonal meshes in traditional single LOD representation. In this paper, we refer to such a mesh as a full mesh M_n. The full mesh is simplified to a coarse mesh M_0 by performing a sequence of edge collapse operations which are the reverse of vertex split operations. This process continues until certain application dependent condition, such as the ratio of the number of vertices from the simplified mesh to the full mesh, is met. The simplified mesh M_0 is defined by a vertex array and a polygonal faces index list. Edge collapse operations are recorded in vertex split records, in which the fields $leaf_1, \ldots, leaf_m$ are populated in the following phase.

3.1.2.2 BVH Construction

In phase 2, a BVH tree T is constructed top-down in a binary tree structure by recursively subdividing primitives of mesh M_n. The operation first defines all primitives stored in M_n as a set. Then the smallest bounding volume (BV) of all primitives in the set is computed and stored in the root node of the tree. The primitives in the set are then split into two subsets by a well chosen partitioning plane. The smallest BVs for the two subsets are computed and stored in the children of the root node. The two subsets of primitives are then further partitioned. This recursive process terminates when the subsets contain one element each. The nodes that bound one primitive are leaves. The partitioning plane is chosen orthogonal to the longest axis of the BV and intersecting at a partitioning coordinate on the axis. Such a

space partitioning process is illustrated in two-dimensions in Figure 3.4. The heuristic behind choosing a partitioning coordinate is to subdivide the set of primitives into two sets of equal size. In such a way, the constructed BVH tree becomes an AVL tree. An AVL tree is a binary tree in which the difference between the height of the right and left subtrees (or the root node) is never more than one. Another heuristic proposed for space partitioning is to choose the median of a BV as the partitioning coordinate [Ber97]. However, when the geometric primitives are not distributed uniformly within their BV, this approach may construct an unbalanced tree. Even if the number of vertices is the same for different models the constructed tree structures may be different. This means more information is needed for a BVH tree structure to be constructed in comparison to the space partition method used in this work. It is shown in the following section that an unbalanced BVH tree does not perform very well in BVH collision query and updating process. BVHs constructed by the abovementioned method have the following property.

Theorem 1

If an original mesh M_n has a set of m primitives, then the constructed BVH binary tree T has m leaves and the tree is always built into a unique structure, regardless the topology and the geometry of the mesh.

Proof

Let T be a BVH tree constructed from M_n, rooted at node x, and let m be the number of primitives in M_n. The primitives are bounded by the leaves of T. Let T_a be a tree rooted at node a. Let left[a] be the left child of a, right[a] be the right child of a, and d_a be the depth of T_a.

When $m = 1$, the root x of T is a leaf node which bounds the primitive.

$$d_x = d + 1, \text{where } m > 2^d \text{ and } m \le 2^{d+1}, d = -1$$

When $m = 2$, T has two leaves.

$$d_x = d + 1, \text{where } m > 2^d \text{ and } m \le 2^{d+1}, d = 0$$

Assume that when $m = k$,

$$d_x = d + 1, \text{where } k > 2^d \text{ and } k \le 2^{d+1}, d \ge -1.$$

When $m = 2k$, the leaves in $T_{\text{left}[x]}$ bound k primitives. The depth of $T_{\text{left}[x]}$ is $d_{\text{left}[x]}$. The leaves in $T_{\text{right}[x]}$ bound k primitives. The depth of $T_{\text{right}[x]}$ is $d_{\text{right}[x]}$.

$$d_{left[x]} = d + 1, \text{where } k > 2^d \text{ and } k \le 2^{d+1}, d \ge -1.$$

$$d_{right[x]} = d + 1, \text{where } k > 2^d \text{ and } k \le 2^{d+1}, d \ge -1.$$

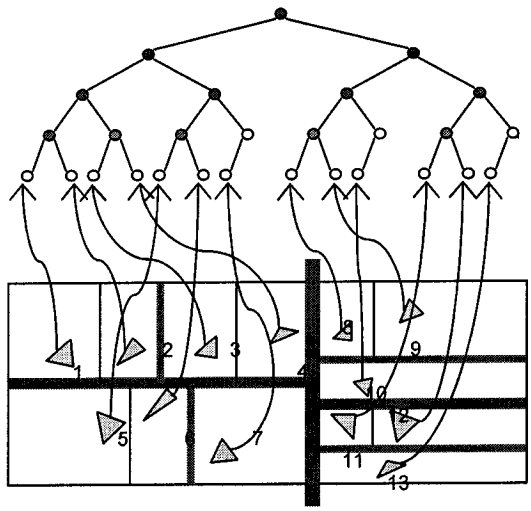


Figure 3.4 A BVH Tree Structure of a Triangle Mesh

$$
\begin{aligned}
d_x &= \max(d_{right[x]}, d_{left[x]}) + 1 \\
&= (d+1)+1, \text{where } 2k > 2^{(d+1)} \text{ and } 2k \le 2^{(d+1)+1}, (d+1) \ge -1 \\
&= d'+1, \text{where } 2k > 2^{d'} \text{ and } 2k \le 2^{d'+1}, d' = d+1,\ d' \ge -1
\end{aligned}
$$

When $m = 2k+1$, the leaves in $T_{\text{left}[x]}$ bound $k+1$ primitives. The depth of $T_{\text{left}[x]}$ is $d_{\text{left}[x]}$.

$$
d_{left[x]} = b+1, \text{where } k+1 > 2^b \text{ and } k+1 \le 2^{b+1}, b \ge -1.
$$

The leaves in $T_{\text{right}[x]}$ bound k primitives. The depth of $T_{\text{right}[x]}$ is $d_{\text{right}[x]}$.

$$
d_{right[x]} = d+1, \text{where } k > 2^d \text{ and } k \le 2^{d+1}, d \ge -1.
$$

If $k < 2^{d+1}$, then $b = d$.

$$
\begin{aligned}
d_x &= \max(d_{right[x]}, d_{left[x]}) + 1 \\
&= (d+1)+1, \text{where } 2k+1 > 2^{(d+1)} \text{ and } 2k+1 \le 2^{(d+1)+1}, (d+1) \ge -1 \\
&= d'+1, \text{where } 2k+1 > 2^{d'} \text{ and } 2k+1 \le 2^{d'+1}, d' = d+1,\ d' \ge -1
\end{aligned}
$$

If $k = 2^{d+1}$, then $b = d+1$ and $2k+1 = 2^{(b+1)}+1$.

$$
\begin{aligned}
d_x &= \max(d_{right[x]}, d_{left[x]}) + 1 \\
&= (b+1)+1, \text{where } 2k+1 > 2^{(b+1)} \text{ and } 2k+1 \le 2^{(b+1)+1}, (d+1) \ge -1 \\
&= d'+1, \text{where } 2k+1 > 2^{d'} \text{ and } 2k+1 \le 2^{d'+1}, d' = b+1,\ d' \ge -1
\end{aligned}
$$

It is proved that the depth of tree T is a function of m where $m > 0$. The tree structure is uniquely determined by the number of geometry primitives in M_n. □

For a CD algorithm, this property makes it easy to build a BVH tree given that the number of primitives of the mesh M is known even though the primitives of the mesh M haven't been loaded yet. This explained why the field #vertex is in CDPM header H and the mesh header

is always transmitted first. The fields #face and #vertex split record are used for memory management of BVH and vertex hierarchy at the client. At runtime, the BVH tree is constructed at the client when the mesh header is received from a remote host. The BVs in the tree nodes are calculated after the coarse mesh is received.

3.1.3 BVH Structure Coding

One assumption of the proposed CD algorithm is that a CDPM is refined gradually instead of being refined in a drastic way. This means only a few primitives of the mesh are affected in each animation frame. Therefore only a small number of BVs in a BVH need to be recalculated for collision detection per frame. To quickly locate these BVs, a bottom–up approach is much faster than a top-down approach given that the links from affected primitives to the leaves of the BVs are known from the received CDPM vertex split data. This requirement is fulfilled by coding the BVH structure and saving the code in the mesh refinement data. The primitives bounded by the leaf BVs are numbered in the order that the BVH tree is traversed. For a primitive f in M_0, its code is recorded in the primitive index list in the course mesh. For those primitives generated during the mesh refinement operation, their codes are stored in the field $leaf_k$ of vertex split records. The simulation results shown in

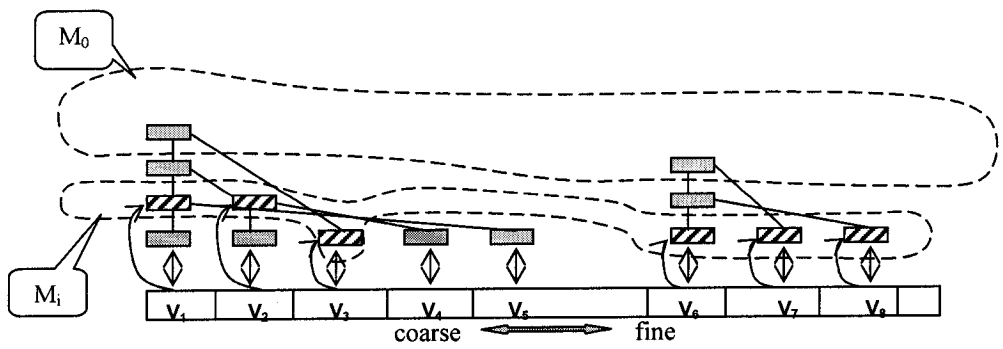


Figure 3.5 CDPM Vertex Hierarchy

Chapter 4 demonstrate that the cost to construct a BVH is negligible when the volume of the transmitted CDPM header H plus coarse mesh M_0 is much smaller than that of the entire mesh.

3.2 Multi-resolution Data Structure

3.2.1 Vertex Hierarchy

A mesh model in multi-resolution comprises a single data structure, *vertex hierarchy*. The vertex hierarchy is a forest. Each node of the trees represents a vertex and contains a record of a vertex split operation. The hierarchy is constructed at the client upon receiving the header H and the base mesh M_0. The root nodes of the trees represent vertices in M_0. The trees in the hierarchy grow upon loading vertex split records. Since one old vertex is split to two new vertices, this parent-children relationship establishes the tree structure. The data structure contains information about the geometry and topology of the model in a continuous LOD. Only leaf nodes are extended dynamically upon receiving vertex split records. A leaf may be extended to have two children to split the leaf's represented vertex into two vertices and generate new faces in the mesh. On the other hand, the extended nodes may be folded into their parent to collapse the split two vertices to one and removed the generated triangles from the mesh. The hierarchical data structure records the history of vertex split and edge collapse operations on the multi-resolution mesh, which enables fast mesh split and merge. A *vertex list,* as shown at the bottom of the hierarchy in Figure 3.5, is designed to record the order in which vertices are collapsed and split in order to quickly track the nodes that are to be folded or unfolded. The hierarchy includes the basic structure described below (explanations of the individual fields follow):

```
struct hierarchy_tree {
        tree_node *         rootPtr;
        index_type          treeIdx;
};
```

- rootPtr: a pointer to a tree node that represents a vertex in mesh M_0.
- treeIdx: index of trees in the hierarchy.

```
struct hierarchy_tree       treeList[NUM_M0_VERTICES];
```

- treeList: a list of trees which compose a vertex hierarchy.

```
struct tree_node {
        triangle *          neighborFaces;
        tree_node *         left;
        tree_node *         right;
        tree_node *         parent;
        vertex_item *       vsplitPtr;
        float               deltaLeft;
        float               deltaRight;
};
```

- neighborFaces: a pointer to a list of triangles that indent to the represented vertex.
- left: a pointer to left child tree node
- right: a pointer to right child tree node
- parent: a pointer to parent tree node
- vsplitPtr: a pointer to a vertex_item in the vertex list
- deltaLeft: the difference in coordinate from the vertex of current node to the vertex of left child
- deltaRight: the difference in coordinate from the vertex of current node to the vertex of right child

```
struct vertex_item {
        tree_node *         treeNode;
        index_type          vsIndex;
};
```

- treeNode: a pointer to a tree node

- vsIndex: index of the vertex that is represented by the tree node pointed by treeNode.

```
struct vertex_list {
        vertex_item *       headPtr;
        vertex_item*        activeItem;
        size_type           numVertices;
};
```

- headPtr: a pointer to the first vertex item in the vertex list
- activeItem: a pointer to the most recent split or collapsed tree node
- numVertices: the number of vertices in the vertex list which is equal to the number of vertices in M_n.

3.2.2 Primitive List

A primitive list in a simple form is a sequence of active triangles in the order the triangles are generated. The primitive list contains more triangles when the mesh is refined to higher LOD. The list is maintained in the current implementation as an array of triangle structures for fast data access. Each item has the following basic structure:

```
struct triangle {
index_type vertex[3];
};
```

- vertex[3]: indices of triangle vertices.

The vertex field represents the indices of triangle vertices. The values are changed only when the represented vertices are split or collapsed.

3.2.3 Mesh Refinement Operation

The fundamental operations associated with nodes in the hierarchy are EdgeCollapse() and VertexSplit(). These functions add or remove triangles from the mesh by updating the *primitive list*, *vertex list* and *hierarchy trees*:

Procedure **VertexSplit**

1. Obtain active vertex item (A)

2. Obtain the tree node to be split (B) from the active vertex item

3. IF received a vertex split record (C) which split v_s' to v_s and v_t THEN

4. Allocate memory for left child (B_L) and right child (B_R) of B

5. Allocate a new vertex item (D) in the vertex list after A

6. ELSE

7. Obtain left child (B_L) and right child (B_R) of B, and the vertex item (D) after A

8. ENDIF

9. Store faces indent to vertex v_s in B_L

10. Store faces indent to vertex v_t in B_R

11. Store the difference of vertex coordinates from v_s' to v_s in B_L

12. Store the difference of vertex coordinates from v_s' to v_t in B_R

13. Set the treeNode field of A to B_L and the treeNode field of D to B_L

14. Set active vertex item to D

15. Add split new faces in primitive list

16. Update vertex indices of deformed faces in primitive list

17. RETURN an index list of new faces and an index list of deformed faces

Procedure **EdgeCollapse**

1. Obtain active vertex item (A)

2. Obtain the indices (v_t) and (v_s) of the vertices to be collapsed from A and the previous vertex item (C) of A

3. Obtain the tree node (B_R) which represents v_t from the treeNode field of A

4. Obtain the parent tree node (B) of B_R which represents the vertex (v_s') to be merged from v_s and v_t)

5. Obtain the tree node (B_R) which represents v_s from the treeNode field of C

6. Set the treeNode field of C to B

7. Set active vertex item to C

8. Update vertex indices of deformed faces in primitive list

9. RETURN an index list of merged faces and an index list of deformed faces

As illustrated in Figure 3.6, a CDPM vertex hierarchy maintains two meshes, *refined mesh* and *loaded mesh*. The refined mesh defines a sub vertex hierarchy in which the vertices represented by leaf nodes are those in the mesh at current resolution. The loaded mesh defines another sub vertex hierarchy which represents a mesh generated by performing a sequence of vertex split operation. The vertex split records of the operations have been subscribed by the client and loaded from the server to the local cache. The hierarchy of the refined mesh is always a sub vertex hierarchy of the loaded mesh because the mesh may be refined from high LOD to low LOD. Initially only the header and the coarse mesh are sent from the server to the client. The number of trees in the hierarchy equals the number of vertices in M_0. Each tree has one node which represents a vertex in M_0. At this stage, the two sub hierarchies overlap (Figure 3.6a). After that, the client subscribes to refinement data from the server. The amount of data requested is proportional to the accuracy of CD required by the application. It is calculated by the LOD selection method introduced in section 4.1. Upon receiving the data, the mesh is refined to a higher LOD. The two sub vertex hierarchies are extended top-down as shown in Figure 3.6b. When the mesh is refined to lower LOD, the corresponding nodes in the hierarchy for the refined mesh are folded. Therefore, the space between the two sub hierarchies is increased (Figure 3.6c).

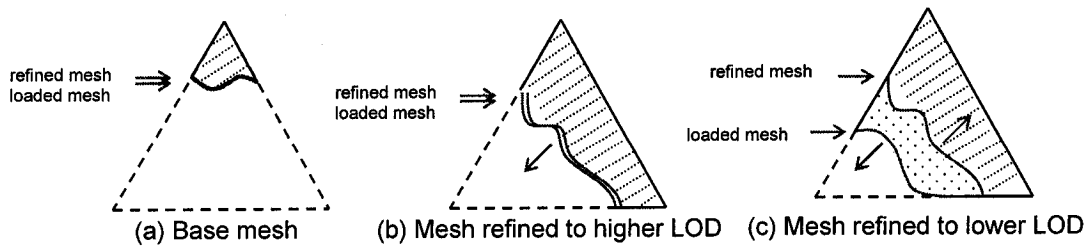


Figure 3.6 Dynamic Construction of CDPM Vertex Hierarchy

3.3 Algorithm for Fast BVH Collision Query

The algorithm to generate a CDPM mesh from a traditional triangle surface mesh was introduced in section 3.1. In DVEs, CDPM are loaded either from the server upon request or from a local secondary storage to the client. The coarse meshes M_0 are loaded into the multi-resolution data structure introduced in section 3.2. The BVHs for collision query are constructed upon receiving H and M_0 of the meshes. Collision queries on the meshes start from this point. The BVH data structure which is introduced in the following section consists of two parts, an AB-tree and a primitive index list. Whenever the meshes are refined, the BVH are updated by refitting the BVs of the affected triangles in tree nodes. The accuracy of collision queries is increased upon splitting more vertices and updating the BVHs accordingly. The accuracy is decreased by reversely performing edge collapses on mesh and updating the BVH. In section 3.3.2, algorithms for fast BVH updating and collision queries are introduced. Although the algorithms are demonstrated under CDPM, they can be extended to accommodate other multi-resolution mesh representations.

3.3.1 Active Bounding Tree Definition

An AB-tree, denoted as T, is a binary tree augmenting an AABB tree with additional data structure. An AABB tree is a BVH algorithm introduced in [Ber97] which provides a fast way to perform exact collision detection between complex models. However, the algorithm does not apply to multi-resolution meshes. An AABB tree node, x, has four fields: an AABB which bounds all AABBs of the leaves in the subtree rooted at x, $range[x]$; a link to the left child of x, $left[x]$; a link to the right child of x, $right[x]$; and a label indicating the type of the node x (leaf or internal), $label[x]$. The AABB of a leaf bounds a geometric primitive. The

following property allows the cost of refitting an AABB in an AABB tree to be independent of the number of nodes in the tree.

Property 1: Let S be a set of primitives. $S+$ and $S-$ are subsets of S. $S+\subseteq S$, $S-\subseteq S$, $S+\cup S-=S$. If $B+$ and $B-$ are the smallest AABBs of $S+$ and $S-$ respectively, and B is the smallest AABB enclosing $B+$ *and* $B-$, then B is the smallest AABB of S. (Proved in [Ber97]).

Assume that *range*[x] is defined as [t_1, t_2], with $low(x) = t_1$ (the low endpoint coordinates) and $high(x) = t_2$ (the high endpoint coordinates). Then following property 1, we have

$$range[x]=[\min(low(left[x]),low(right[x])),\max(high(left[x]),high(right[x]))]$$

An AB-tree augments an AABB tree by a primitive index list, which contains an index of a triangle face in the full mesh M_n and a link to a leaf node which bounds the face. In the current implementation this list is stored as an array. An element f has a link to a leaf in the tree, *leaf*[f] and the index of f is an index of a triangle face. The four fields defined for AABB tree nodes, internal nodes contain two additional fields, a link to the parent of x, *parent*[x]; a status of the node x, *status*[x] where

$$status[x] \in \{active,\ inactive,\ deformed\}.$$

Leaves also contain another field, a link to an element in the primitive index list, *index*[x]. Each leaf is pointed to from exactly one element in the primitive index list (Figure 3.7).

A multi-resolution mesh is refined at runtime. The mesh primitives may be inserted, removed, or deformed. Their geometry and topology are changed dynamically. Therefore, an accelerated algorithm is required to recalculate the BVs which bound the changed primitives. A method is proposed to quickly locate a set of leaves and all of their ancestors in the BVH,

given that the primitives bounded by the leaves are known. These primitives are obtained from the output of the procedures VertexSplit and EdgeCollapse, which are introduced in section 3.2. In an AABB tree, this operation may require searching several paths top-down due to the possible overlaps between the BVs of the sibling nodes. However, in an AB-tree, the primitive indices are used as key to the primitive index list so that a leaf which bounds a primitive can be located in a constant time. The parent field is then used to find all the ancestors of the leaf in a bottom-up tracing. Compared with an AABB tree, it is clear that an AB-tree has better performance.

Hybrid tree is proposed in [LM01] for performing collision detection on deforming bodies. The hybrid tree can be pre-built and updated during simulation by a combination of top-down and bottom-up update strategy. Meshes connectivity information is stored in both internal and leaf nodes cost more memory than our AB-tree. The AABB bounding volume of node is updated by traversing the shared vertices of the faces in the node. The updating algorithm bottom-up updates the tree in half the depth of the entire tree. Thus, time cost is O(n) where n is the number of vertices on the mesh. The analytical result in section 3.3.3 shows that AB-tree has better performance than hybrid tree when the mesh does not refine drastically.

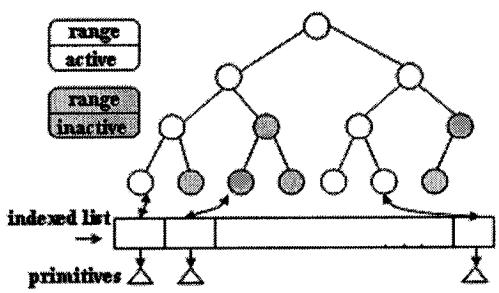


Figure 3.7 AB-tree Structure

For CD on multi-resolution meshes, the time to recalculate an affected AABB BV is guaranteed to be independent of the number of nodes in the tree by property 1. In addition, the primitives of a refined mesh and their BVs need to be maintained efficiently such that the complexity of collision queries performed on the BVH is proportional to the mesh complexity. In other words, when the mesh is refined to be coarser, the collision queries run faster. The *status* field works for the second requirement. This is explained in the next section.

3.3.2 Operations

As discussed in section 3.1.3, only a small number of BVs in the BVH are affected in mesh refinement per animation frame. To quickly identify these BVs, a bottom–up approach has been explored. The primitive index list and the parent pointer in tree nodes are designed for this purpose. Once the nodes are located, the *status* field is used to identify those affected BVs. A status of *deformed* indicates that the BV needs to be recalculated because its bounded primitive is deformed or is generated during vertex split; *inactive* indicates that the BV is temporarily not used for collision query because its bounded primitive has not be generated or has been removed from the mesh; *active* indicates that there is no change to the bounded primitive. Initially only the leaf nodes in the AB-tree whose BVs bound the primitives in M_0 and their ancestors are set to *active*. All the other nodes are set to *inactive*. The first step to update a BVH is to mark all the affected leaves and their ancestors as deformed. As a result, the *deformed* nodes form a subtree of the AB-tree. Then a top-down traverse is performed on the subtree, the recalculation of BVs is applied to *deformed* nodes. Compared to the AABB tree which requires a traversal of the whole tree, our BVH algorithm

is much faster. Before we describe the detailed operations, we need to introduce some notations.

Let M_n denotes a full mesh, P a CDPM generated from M_n, H the header of P, α the number of primitives in M_n, T and T' two AB-trees, x a node in T and T_s a subtree of T. Here F is the primitive index list of T which contains α pointers to leaf nodes while I, R, and D are three sets of indices of the primitives in M. $R \cap D=\emptyset$. $I \cap D=\emptyset$. $I \cap R=\emptyset$. Assume a refinement is to be performed on P. In this case I contains the indices of the primitives to be inserted into the mesh, R contains the indices of the primitives to be removed from the mesh, and D contains the indices of primitives to be deformed in the mesh.

The diagram in Figure 3.8 illustrates the state transition of an AB-tree node.

An AB-tree supports the following three operations. Part of the pseudo-code is given.

1. Pre-construct the AB-tree

Procedure: ACTIVE-PRECONSTRUCT

Upon receiving H, T is pre-constructed in a method similar to the one introduced in section 3.1.2.2. T is constructed top-down in binary tree structure by recursively subdividing the number of primitives α into two sets of equal size and creating a node for each set of the primitives. The process terminates when the size is equal to 1. For a node representing a primitive set in size β, its left subtree contains $\lceil \beta/2 \rceil$ leaves and its right subtree contains $\lfloor \beta/2 \rfloor$ leaves. It is easy to prove that T is an AVL tree. It can be proved further by Theorem 1 that the tree structure is the same for all meshes that have α primitives. When the tree is

constructed, the *range* field is not populated and the *status* field is initialized to *inactive* for all nodes because no mesh primitive data has been received at this point.

2. Update the BVH upon mesh refinement

Procedure: ACTIVE-UPDATE

It calls procedure ACTIVE-MARK to mark the AB-tree T. Then it calls operation ACTIVE-WALK to recalculate BVs in the marked subtree T_s.

Input: T is an AB-Tree. In a mesh refinement, I is a set of indices of the primitives to be inserted into the mesh, R is a set of indices of the primitives to be removed from the mesh, and D is a set of indices of primitives to be deformed in the mesh.

ACTIVE-UPDATE (I, R, D, T)
1 $T_s \leftarrow$ ACTIVE-MARK(I, R, D, T)
2 $x \leftarrow root[T_s]$

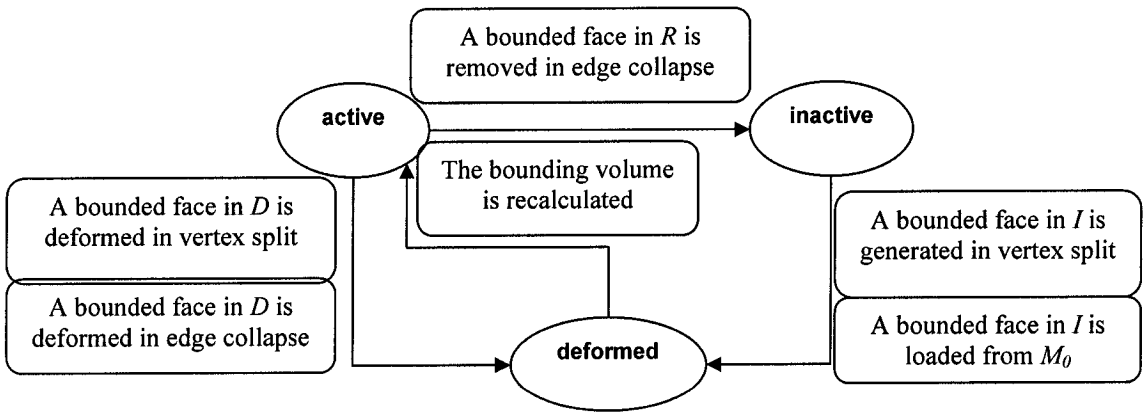


Figure 3.8 State Transition of an AB-tree node

3 ACTIVE-WALK(x)

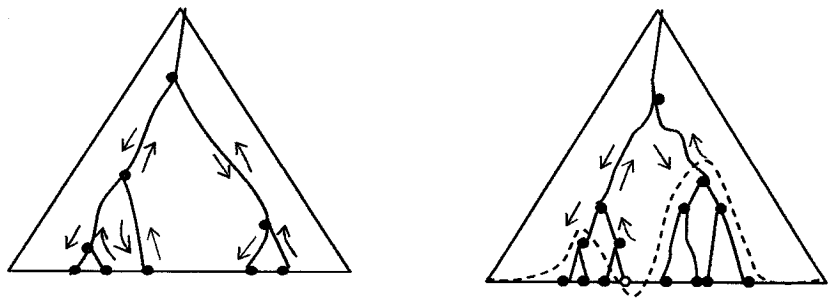
Procedure: ACTIVE-MARK

Upon mesh refinement, this procedure is called to mark the tree T bottom-up and generates a subtree T_s as shown in Figure 3.9. The logic behind this procedure is that when no one primitive bounded by a node or by its descendants is in the mesh at the current resolution, the BVs of the nodes need not be tested in collision queries. Therefore, the nodes are set to *inactive*. When a primitive bounded by a node or by its descendants is to be inserted into or deformed in the mesh at the current resolution, the BVs of the nodes needs to be re-calculated before a collision query is performed. Therefore, the nodes are set to *deformed.*

Input: T is an AB-Tree. In a mesh refinement, I is a set of indices of the primitives to be inserted into the mesh, R is a set of indices of the primitives to be removed from the mesh, and D is a set of indices of primitives to be deformed in the mesh.

Output: T_s is a subtree of T which is composed of all nodes in T that are set to *inactive* or *deformed* state.

ACTIVE-MARK(I, R, D, T)



(1) Mesh is refined to high LOD. (2) Mesh is refined to low LOD.

Figure 3.9 BVH Updating. Upon refining the mesh, mark the tree bottom-up and refit the subtree of deformed nodes in post-order. Active nodes are in white, inactive nodes are in green, deformed nodes are in red

```
1 FOR each x in R
2   WHILE x ≠ nil[T]
3         status[x] ← inactive
4       IF x = right[parent[x]] THEN
5             IF left[parent[x]] = nil[T] OR status[left[parent[x]]]  = inactive THEN
6               x ← parent[x]
7             ELSE
8               x ← nil[T]
9             ENDIF
10     ELSE IF right[parent[x]] = nil[T] or status[right[parent[x]]]  = inactive THEN
11            x ← parent[x]
12     ELSE
13            x ← nil[T]
14     ENDIF
15  ENDWHILE
16 ENDFOR
17 FOR each x in D and I
18   WHILE x ≠ nil[T]
19     status[x] ← deformed
20     x ← parent[x]
21   ENDWHILE
22 ENDFOR
23 output Ts
```

Procedure: ACTIVE-WALK

This procedure post-order traverses the marked subtree T_s. For a node x in T_s, if *status*[x] is *deformed*, ACTIVE-REFIT is called to calculate the BV of x. If *status*[x] is *inactive*, the subtree of T_s rooted at x will not to be traversed. This procedure returns an updated BVH in which the status of a node is either *active* or *inactive*. The tree formed by active nodes is a subtree of T. The BVH collision detection only applies to the active nodes. When the resolution of the mesh becomes coarser, the number of nodes whose status fields are *active* is decreased and the depth of the tree applied to collision query is decreased. Thus the running time of the collision detection is reduced.

```
ACTIVE-WALK(x)
1 IF x ≠ nil[T] and status[x] = deformed THEN
2       ACTIVE-WALK (left[x])
3       ACTIVE-WALK (right[x])
4       ACTIVE-REFIT(x)
5 ENDIF
```

Procedure: ACTIVE-REFIT

This procedure calculates the smallest BV of a node x, *range*[x], which encloses *range*[*left*[x]] and *range*[*right*[x]], and sets *status*[x] to *active*.

3. BVH collision query

This operation is similar to the collision queries method introduced in [Ber97] except that for each visited pair of nodes, the two BVs are tested for overlap only if the *status* of the two nodes are both *active*. The nodes for which the BVs overlap are then further traversed. When both nodes are leaves and their primitives intersect, a collision is detected. A procedure ACTIVE-INTERSECT is performed to query collision between two BVHs T and T'.

3.3.3 Complexity Analysis

The time cost of procedure ACTIVE-PRECONSTRUCT is O(n), where n in the number of nodes in an AB-tree. The number of leaves in an AB-tree is α, where α is the number of primitives in mesh M_n. Since an AB-tree is an AVL tree, the time cost becomes O(α). The space cost is determined by the number of node in the tree. It equals O(α).

The performance of the BVH updating algorithm is governed by the complexity of operation ACTIVE-UPDATE. The complexity of operation ACTIVE_REFIT(x) is not affected by the total number of nodes in T but the type of BV. Hence, we consider this procedure costs a

constant time. The following paragraphs give a proof that ACTIVE-UPDATE takes $O(\log n)$ time in worst-case to update a AB-tree.

Lemma 1

For a binary tree, T_c, which has n leaves and the height is $O(\log n)$, marking k randomly chosen leaves and all of their ancestors requires marking $O(k\log n - k\log k + 2k)$ nodes in worst-case.

Proof

Let the height of T_c be H_c. If T_c's longest path from root to leaves has h nodes, then $h \leq H_c + 1$. Let $b = \log_2 k$, b is a non-negative integer. $T_c(k)$ is the number of marked nodes in T_c. Let S be a set containing k leaves. $S = \{l_1, l_2, \ldots, l_k\}$.

In the worst case, marking the 1st leaf l_1 and its ancestors top-down from the root node to l_1, requires marking at most h nodes. Marking the 2nd leaf l_2 and its ancestors top-down from the root node to l_2, requires marking at most h-1 nodes. For the 3rd and 4th leaves, this marking process requires marking at most h-2 nodes respectively. For the 5th to 8th leaves, this marking process requires marking at most h-3 nodes respectively.

By induction, marking k leaves requires marking $T_c(k)$ nodes of T_c in the worst case, where

$$T_c(k) \leq 2^b h - \sum_{i=0}^{b} 2^{i-1} \times i$$

By differentiating a finite geometric series $\sum_{i=0}^{b} x^i$, we get

$$(\sum_{i=0}^{b} x^i)' = \sum_{i=0}^{b} (x^{i-1} \times i)$$

Let x=2, since

$$(\sum_{i=0}^{b} x^i) = \left(\frac{x^{b+1}-1}{x-1}\right),$$

$$\sum_{i=0}^{b}(2^{i-1} \times i) = \sum_{i=0}^{\log_2 k} 2^{i-1} \times i = k(\log_2 k - 1) + 1$$

$$T_c(k) \le k(\log_2 n + 1) - \sum_{i=0}^{\log_2 k} 2^{i-1} \times i$$

$$T_c(k) \le k(\log_2 n - \log_2 k + 2) - 1$$

$$T_c(k) = O(k \log n - k \log k + 2k)$$

□

Theorem 2

Let T_a be an AB-tree built upon a CDPM mesh P. F is the primitive index list of T_a. I, R, and D are three subsets of indices in F, $R \cap D = \emptyset$, $R \cap I = \emptyset$, and $I \cap D = \emptyset$. The procedure ACTIVE-UPDATE (I, R, D, T_a) takes $O(k (\log n - \log k + 2))$ time where n is the number of leaves in T_a, k is the number of primitives to be deformed in, removed from, or inserted into P at an instant resolution. Furthermore, if k is set to be in a range [1, K], where K is a constant and $K << n$, then the procedure takes $O(\log n)$ time.

Proof

We consider the maximum number of calls to ACTIVE-REFIT as the worst-case running time of ACTIVE-UPDATE. The main idea of this proof is to find out the maximum number of nodes whose field *status* is set to *deformed.*

Let k_1 be the size of R, k_2 be the size of I, and k_3 be the size of D, then $k = k_1 + k_2 + k_3$. Let $T(n, k)$ be the running time of ACTIVE-UPDATE(I, R, D, T_a). The total number of loops in ACTIVE-MARK(I, R, D, T_a) is k. In the second **FOR** loop of pseudo-code at lines 17-22, when k_1=0, the maximum number of loops is k. Since an AB-tree is an AVL tree and n is the number of leaves in T_a, T_a is a binary tree and the height of T_a is $O(\log n)$.

It can be proved by lemma 1 that for an AB-tree T_a the maximum number of operations in line 19, measured as the running time of ACTIVE_UPDATE (I, R, D, T), is

$$T(n,k) = O(k\log n - k\log k + 2k)$$

If k is set to be in a range [1, K], where K is a constant and $K<<n$, then the call to ACTIVE_UPDATE(I,R,D,T) takes $O(\log n)$ time. □

The procedure ACTIVE_INTERSECT performs collision queries on AB-trees. Its complexity is given in [GLM96] as:

$$T_q = N_{bv} \times C_{bv} + N_p \times C_p,$$

where T_q is the total cost for a collision query, N_{bv} is the number of BV pair operations, and C_{bv} is the cost of a BV pair operation. Here N_p is the number of primitive pairs tested for proximity, and C_p is the cost of testing a pair of primitives for collision. Typically, for tight-

fitting bounding volumes, e.g., OBBs, N_{bv} and N_p are relatively small, whereas C_{bv} is relatively high. In contrast, C_{bv} is low while N_{bv} and N_p may be larger for simple BV types such as spheres and AABBs. Due to the opposing trends, no single BV yields optimum performance for collision queries in all situations. In the AB-tree algorithm, T_q is controlled by adjusting the LODs of CDPM meshes. When a mesh is refined to a lower LOD, both the number of BV pair-wise overlap tests between actives nodes N_{bv} and the number of primitive pair tests N_p become smaller. Therefore the cost of collision queries is reduced.

The framework of multi-resolution collision detection for distributed virtual environment applications is introduced in chapter 4.

Chapter 4 Collision Detection in Distributed Virtual Environments

The accuracy of computed collision response is directly dependent on the LOD of meshes to which the collision detection algorithm is applied. LOD selection (or LOD management) for real-time rendering of large-scale virtual environment and related perceptual issues have received a lot of attention. It deals with assigning LODs to objects based on a series of criteria. Funkhouser and Sequin proposed some criteria, such as semantics, focus, motion, and hysteresis, to calculate the visual importance of objects [FS93]. Reddy uses distance and projected screen space to manage LODs [Red01]. Brown et al. introduced an attention model for LOD selection [BCP03]. Gaze direction is another criterion to LOD management [LHN00]. O'Sullivan and Dingliana investigated different factors that affect collision perception, including eccentricity, separation, distracters, causality, and accuracy of collision response [OD01]. A higher LOD results in more perceivable consistency in physics simulation. A lower LOD brings in more inconsistency. For example, one of the possible consequences of reducing the accuracy of CD is that objects bounce off each other at a distance, leaving a perceivable gap. Therefore, maintaining the level of collision detection

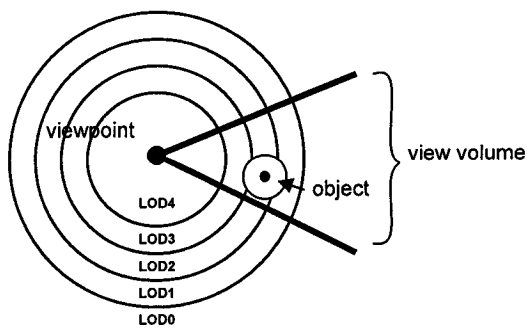


Figure 4.1 Mapping Distance to LOD

detail as close to the LOD for rendering as possible can generate a visually more realistic simulation. In a distributed environment, object models are stored in database servers remotely. The CDPM allows a server to progressively transmit models at LODs required by a client.

4.1 LOD Selection

4.1.1 Selection Criteria

The LOD of an object model required by a client at a particular moment is determined by several factors that are observed to highly affect the realism of simulation.

The first factor, *application context function,* is determined per application context. The idea is to classify all objects in the virtual scene by their context (a measure of importance) and assign higher LODs to more important models. For example, in a virtual museum, items on display have the highest LOD, while static background scenes, such as walls and ceiling, have the lowest LOD.

The second factor is *viewing distance function* which is based on the perceptual capacity of the human visional system. It seeks to give a model a low LOD as it moves away from the viewpoint and give a high LOD as it moves toward the viewpoint (Figure 4.1).

The third factor, *movement velocity,* is based on the following observations. Objects that are moving quickly across the screen appear blurred and can be seen for only a short amount of time. A viewer may not be able to see them clearly. Therefore, lower LODs are allocated to fast moving objects and higher LODs are allocated to static and slow moving objects.

The cost of transmitting models in a distributed environment and the usage of network bandwidth are directly affected by the LOD selection algorithm at the client. Overestimating the required LODs increases the overhead of data transmission and decreases the frame rate at the client, underestimating the required LODs brings inconsistency in the simulation. In our LOD selection method, the lowest LODs are used in initialization. In each frame, the LODs of the objects are dynamically adjusted upon the changes of the factors. First, the application context is used to determine the LOD requirement at the client. Second, if the time allocated to the collision detection in the current frame is not exhausted, the LODs of all objects are increased. Otherwise, the LODs of all objects are decreased. The increasing and decreasing steps are calculated by the viewing distance and movement velocity. Finally, if the LOD of an object is increased, the client subscribes to more mesh refinement data from the server. Upon receiving the data, the client refines the models to a higher resolution. If the LOD of an object is decreased, the model is refined to lower resolution. In this way, the accuracy of the collision detection is maximized in an interactive application.

4.1.2 LOD Selection Model

It is expensive to determine the LOD of every object in the environment based on these continuously changing factors. In order to simplify the selection of LODs, we propose an LOD selection model to map these factors to a series of discrete LODs. The model is proposed based on the abovementioned observation in the form of a heuristic. The aim is to find the appropriate LODs for all objects in a scene that produces the most realistic collision simulation within a target animation frame. The model is formulated as follows.

In a virtual environment, the estimated available time T for collision detection per frame can be expressed as

$$T = T_R + T_B + T_Q + T_{LOD}$$

where T_R is the time for mesh refinement, T_B is the time for BVH updating, T_Q is the time for collision queries, and T_{LOD} is the time for LOD selection. T_R is proportional to the number of vertex split (or edge collapse) operations needed to refine a mesh from one LOD to another. When LODs do not change drastically between neighboring frames, T_R is roughly a small constant and T_B is in the order of the logarithm of the size of the full meshes which is proved by theorem 2. T_Q is determined by not only mesh resolution but also collision configurations. T may be set to either fixed or variable depending on the stability of the costs of other tasks in an animation frame. The LOD for each object has to be selected during T_{LOD}. The LOD selection model calculates LODs in linear time to the number of objects in the scene.

An object is defined as O (α, β, γ, *LOD1*, *LOD2*), where α is its current viewing distance, β is its application context, γ is the length of 2D projection of its motion path in previous frames, *LOD1* is the LOD of previous frame, and *LOD2* is the LOD of current frame. In preprocessing, each object is classified by its application context into three categories: most important (Q1), where $\beta=1$, less important (Q2), where $\beta=2$, and not important (Q3), where $\beta=3$. In each frame, the measured time for CD T' in previous frame is compared with T. The result falls into one of the following three cases.

1. $T' - T \leq -t$
2. $T' - T > 0$

3. $T'- T \leq 0$ and $T'- T > -t$, where t is a threshold parameter

In case 1, the LODs for the current frame are to be increased exponentially. In case 2, the LODs are to be reduced exponentially. In case 3, the LODs remain unchanged. We deal with the first two cases differently. Let (0, D) be the visible range, W be the diagonal length of screen space, and ΔLOD be the number of vertex split for case 1 and edge collapse operations for case 2. In our implementation, ΔLOD equals the total number of the operations of a model divided by 50 and t equals 10% of T. A smaller ΔLOD means that visually the accuracy of the CD process changes more smoothly. In case 1,

$$LOD2 = LOD1 + \Delta LOD \times f(\alpha,\gamma) \times g(i,j,\beta).$$

In case 2, the time cost for CD needs to be reduced quickly in order to guarantee an interactive rate. Therefore, we simplify the calculation of LODs and accelerate the reduction of LODs in comparison with case 1.

$$LOD2 = LOD1 + \Delta LOD \times g(i,j,\beta).$$

$$f(\alpha,\gamma) = \min(\frac{D-\alpha}{D}, \frac{W-\gamma}{W})$$
$$g(i,j,\beta) = \eta_1 \times g_1(i,j,\beta) + \eta_2 \times g_2(i,j)$$

where $\eta_1+\eta_2=1$. In case 1, $\eta_1=1$ and $\eta_2=0$. In case 2, $\eta_1=0$ and $\eta_2=1$.

$$g_1(i,j,\beta) = \begin{cases} 2^{i-j-\beta}, i-j \geq \beta, i = 1,2,3,...,n, and\ j = 1,2,3,...,n \\ 2^0, i-j < \beta, i = 1,2,3,...,n, and\ j = 1,2,3,...,n \end{cases}$$
$$g_2(i,j) = \begin{cases} 2^{2j-i-1},\ 2j \geq i+1, i = 1,2,3,...,n, and\ j = 1,2,3,...,n \\ 2^0, 2j < i+1, i = 1,2,3,...,n, and\ j = 1,2,3,...,n \end{cases}$$

where i is the index of current frame, n is the total number of frames, and j is the index of the closest previous frame of i with a condition that in j+1 frame, the case is changed from 1 or 3 to 2, or from 2 or 3 to 1. Function $g(i, j, \beta)$ which is a factor of ΔLOD is illustrated in Figure 4.2 in which j is 5.

In order to maintain the time cost of collision queries in a frame to be below a threshold, the resolution for each model has to be changed dynamically. In system initialization phase, each model is refined to the coarsest resolution. Then, if the computational resources are available, each model is refined to higher resolution by following the LOD selection method introduced in this section. In such a way the gap between the LOD for rendering and the LOD for collision detection is reduced.

4.2 Multi-resolution Collision Detection Framework

The proposed multi-resolution CD approach mainly addresses the issues of interactive collision detection in distributed and complex virtual environments. To meet the requirement of streaming object models over low bandwidth networks at runtime the new approach must

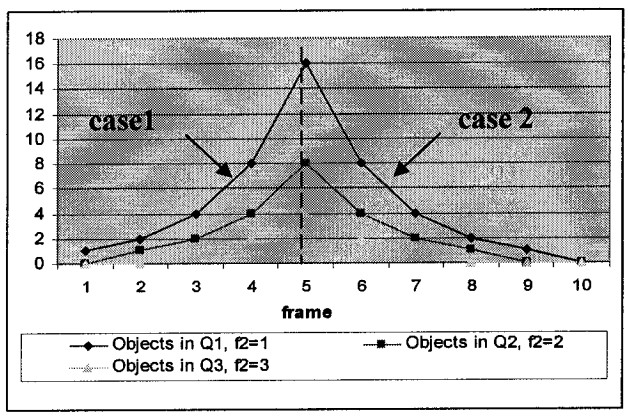


Figure 4.2 Illustration of Function $g(i,j,\beta)$

be able to create a new multi-resolution CDPM mesh representation for transmission, and a vertex hierarchy for mesh refinement at runtime, and a new BVH algorithm, along with a new BVH algorithm based on the AB-tree. In a DVE, multi-resolution models are stored in database servers. The servers are allowed to transmit the models at resolutions on-demand for exact collision detection at the clients. This could save network bandwidth from transmitting extra details for those objects that are out of the end users' view volume, or that rarely collide with other objects since in these cases highly accurate collision detection is not necessary. In addition, clients may run on computers that have different computing capabilities and memory spaces. More powerful clients can obtain more detailed models from servers and gain more accuracy in physics simulations, instead of achieving uniformly low accuracy for CD on all clients for the purpose of maintaining an interactive rate at the least powerful client. Due to transmission delay, the geometric data are received at the clients in a progressive manner. Utilizing the proposed algorithms and data structures, the accuracy of collision detection performed at a client increases smoothly when more and more data are received.

A multi-resolution collision detection system is composed of two parts: a server subsystem

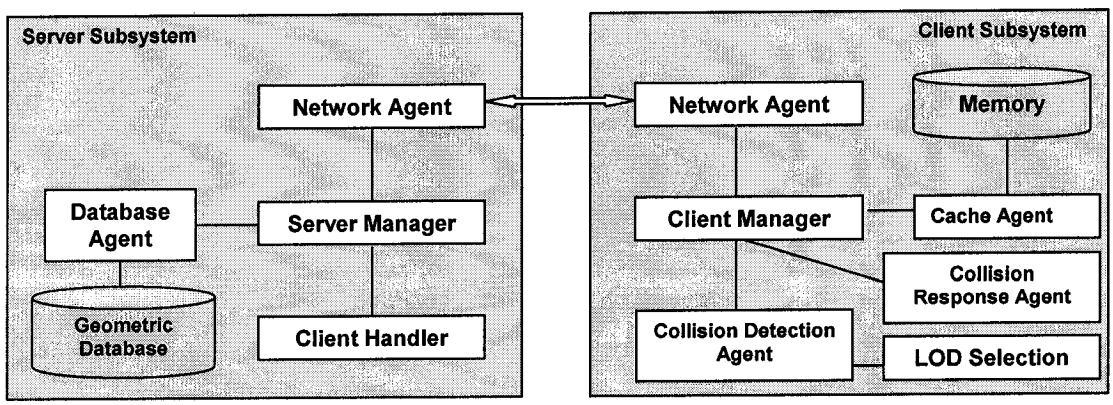


Figure 4.3 Architecture of Multi-resolution Collision Detection

and client subsystem as shown in Figure 4.3. The client subsystem is responsible for LOD selection of models, subscription of mesh data from the server, management of data transmission between client and server, mesh refinement, and collision detection. The server subsystem is responsible for handling the subscriptions from the clients, loading mesh data and progressively transmitting the data. A protocol for CDPM subscription between client and server is illustrated in Figure 4.4. The multi-resolution collision detection approach has two phases, the preprocessing phase and the runtime phase.

Preprocessing Phase

1. Client subscribes models in CDPM representation from server
2. Client loads received mesh models to vertex hierarchy
3. Client constructs AB-tree BVHs of the models

Runtime Phase

For each animation frame

1. Select LOD for each model
2. Subscribe mesh refinement data from the server
3. Mesh refinement and BVH updating
4. Broad phase collision detection
5. Narrow phase pair-wise collision query
6. Rigid body animation

In the preprocessing phase, the volume of transmitted mesh data is relatively small compared to that of the entire meshes. The BVH structures are encoded in the CDPMs. Therefore, the cost of the first phase is negligible. In the runtime phase, the time left per frame on collision detection is estimated as determined by the application's performance goals and the set of operations performed during the frame. Initially BVHs are built on the coarsest meshes. Then some meshes are selectively refined globally to lower or higher resolution, based on the available time and memory, and the configuration of the objects relative to others in a

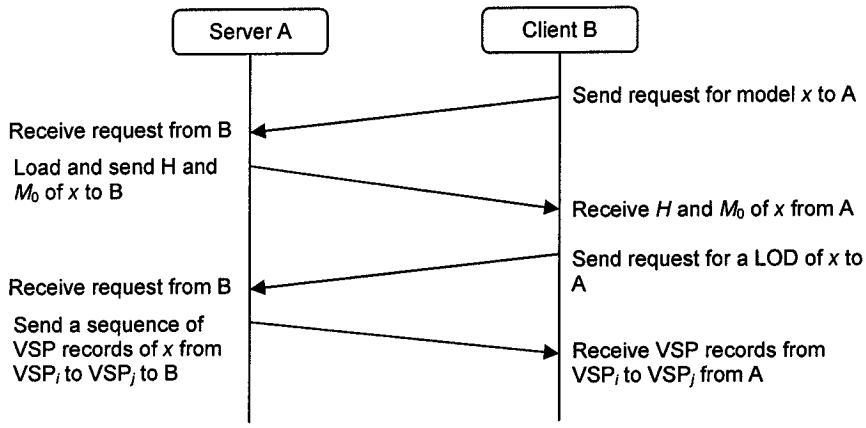


Figure 4.4 CDPM Subscription Protocol

scene. Then, the BVHs of the models are updated. Finally, broad phase and narrow phase pair-wise collision detection are performed on the models. The experimental results introduced in the following section demonstrate significant performance improvement over existing algorithms for static LOD meshes in distributed environments.

4.3 Implementation and Experimental Results

In this section, we describe our implementation in a distributed environment setting and demonstrate rigid body dynamic simulation results.

Table 4.1 Parameter Settings for Models

Models	#faces in M_n (Original Mesh)	#vertices in M_n (Original Mesh)	#faces in M_0 (Base Mesh)	#vertices in M_0 (Base Mesh)	#vertex split records	AB-tree height
Sphere	496	2050	50	27	2023	12
Cow	5804	2904	704	355	2549	12
Bunny	37576	20000	500	1406	18594	16
Dragon	50760	25418	500	258	25160	16

4.3.1 System Implementation

A variety of models have been processed with this CD approach. Some of them are listed in Table 4.1. We have demonstrated the approach in a local area network which provides 100MB bandwidth connection. A prototype application runs on a Pentium4 2.8GHz processor PCs with 510MB of RAM and Windows XP OS. The real-time graphic rendering is achieved with NVIDIA® GeForce™ FX5200 Graphics Cards. The physics simulation was implemented in C++ and the OpenGL library was used for graphics rendering. In our performance study, we focused on analyzing the complexity of updating BVHs, measuring the cost of mesh refinement in terms of network delay and CPU time, and measuring the CPU time of collision queries on the BVHs in various mesh resolutions. The following section provides the analysis of the experimental results from different perspectives.

4.3.2 Runtime Performance

We have successfully applied our approach to interference detection on the benchmark models given in section 4.3.1. Screen shots of the demonstrations are shown in Figure 4.5 and Figure 4.6 and experimental results are given in Figure 4.7 and Figure 4.8.

The performance of collision detection is determined by the complexity of the geometric models and the granularity of the continuously refined LODs. We have performed a measurement in CPU time of the collision detection task on both the static LOD meshes and the CDPMs at the same resolution. We do not observe a performance loss on the CDPMs, which require a complex data structure in comparison with a simple data structure used by the static LOD. On the other hand, when the resolution of a CDPM decreases, the performance of the CDPM interference detection is improved significantly. When a CDPM

gets coarser, the number of pair wise BV interference detections and the number of pair wise primitive interference detections are decreased. Therefore, the running time is decreased. We have also performed a complexity analysis on the dynamic BVH refitting algorithm described in section 3.2 and compared the theoretical result with experimental results. Figure 4.7 shows a comparison between the time cost given in theorem 2 and the CPU time of the BVH updating algorithm upon refining the meshes at different granularity. The measured CPU time increases in proportional to the analytical results when k, the number of leaves to be marked in the BVH increases. The only exceptions are the cases when k is very close to n. Typically k is set to be a constant that is much smaller than n. Therefore we believe that there is strong agreement between the theoretical and experimental results. Furthermore, we observe that when k is smaller than 10% of the total number of faces in the original mesh, the updating operation requires less than 15% of the query time. In some computer simulation and animation applications, the context of the models does not change very often and the movement configuration of the models does not change swiftly. The granularity of the continuously refined LODs is not too large relative to the number of faces in a mesh, which means k is relatively small. Therefore, the proposed collision detection approach performs quickly and robustly in these applications. In comparison to existing time-critical approaches for static LODs, the new approach has the following advantages. (1) Physically more accurate simulations are generated because collision response calculation is on instantaneous impulse mesh models rather than BVs of the mesh primitives; (2) The computational cost is substantially reduced because partially refitting the BVH when the LODs of objects do not change drastically has a logarithmic worst-case complexity; (3) Since the BVHs are dynamically refitted according to the adaptively changed LOD, perceptual inconsistency is

successfully removed. In addition, the approach can handle non-manifold triangle meshes in an arbitrary topology.

In a distributed environment, the time for collision detection can be expressed as

$$T = T_D + T_B + T_Q$$

where T_D represents the network round trip delay which is assumed to be a constant, T_B represents the time to receive mesh refinement data, refine the mesh and update its BVH, and T_Q represents the time for collision queries. Assume that the collision query frame rate is fixed and the available network bandwidth is measured by the number of vertex split records received per frame. T_D increases in proportion to the volume of transmitted data per frame. Figure 4.8 (a) and (b) show the time for T_B. We can see that T_B, the time it takes for mesh refinement and BVH updating increases in a linear relation to the mesh data received at the client. The time it takes to transmit a mesh model in the lowest LOD through networks of various bandwidths is almost the same due to the small data volume. When the required LOD increases, the value of T_B becomes relatively low in high bandwidth networks. The time taken to refine the mesh to the highest LOD and update the BVH in a low bandwidth network is 1.5 times that of the high bandwidth network, as shown in Figure 4.8 (a). This ratio becomes 3 in Figure 4.8 (b). Hence, the time cost for T_B does not increase in a ratio greater than a relatively small constant. In terms of T_Q, we find that the time for collision queries in a low bandwidth network is 10% more than the time cost in a high bandwidth network, and the difference approximates zero when the highest LOD is reached as shown in Figure 4.8 (c) and (b). This relatively small difference in T_Q is caused by the loosely fitted BVHs generated during mesh refinement. However, since the BVH structures encoded in

CDPM meshes are based on a tightly fitted BVH of the finest mesh, the difference in T_Q will converges to zero when the highest LOD is reached.

In conclusion, the performance of the proposed multi-resolution collision detection is slightly affected by the network bandwidth. However, the initial waiting time at a client is significantly reduced. We propose to segment one heavy computing task into many subtasks. The subtasks in the client are performed in parallel with the data transmission tasks over the network. As more and more mesh data are received, the accuracy of the collision detection is increased accordingly. From an end users' point of view, they can start running a virtual environment without waiting for a complex environment to be received. A realistic physics simulation is smoothly achieved in a few seconds.

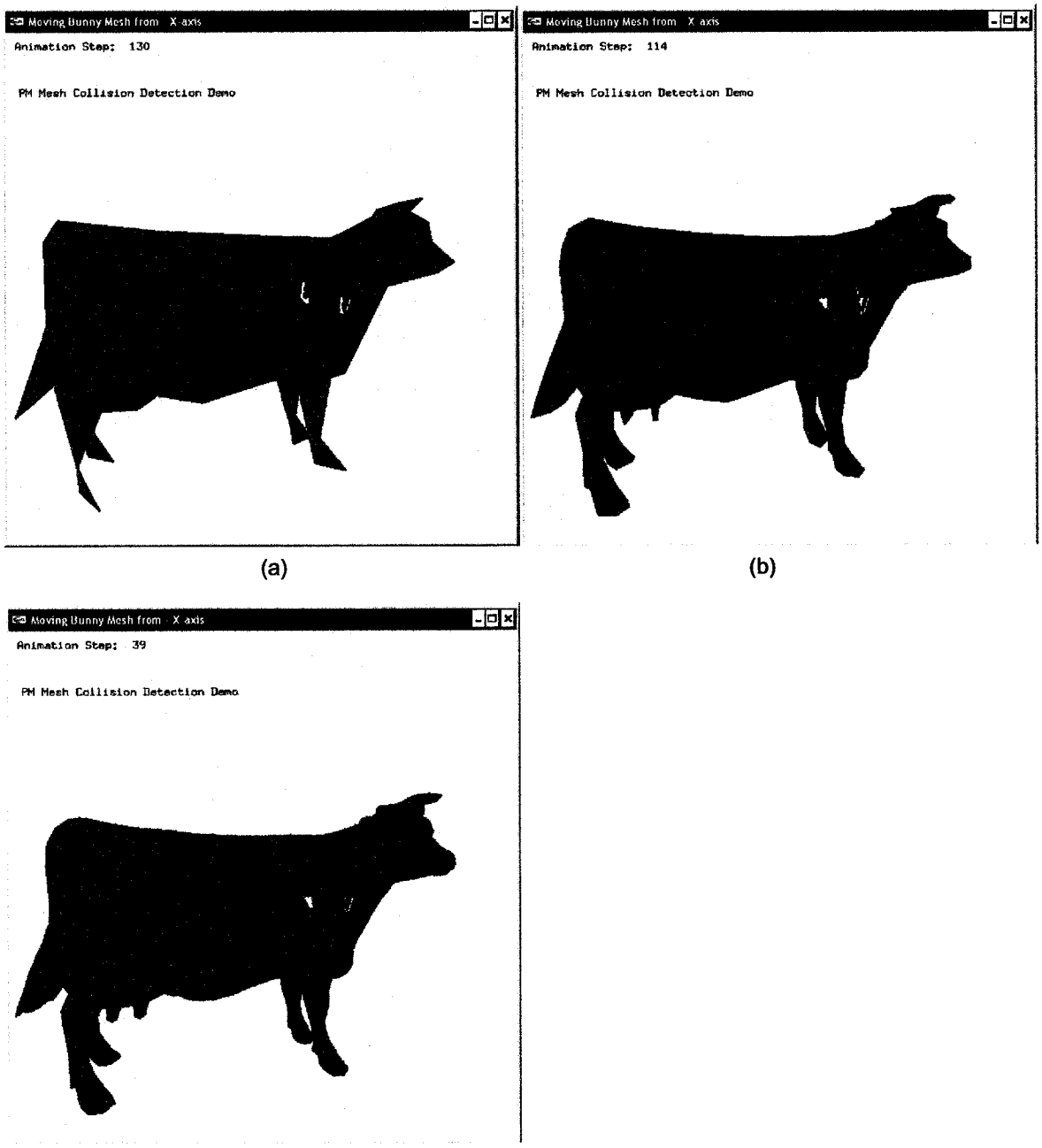
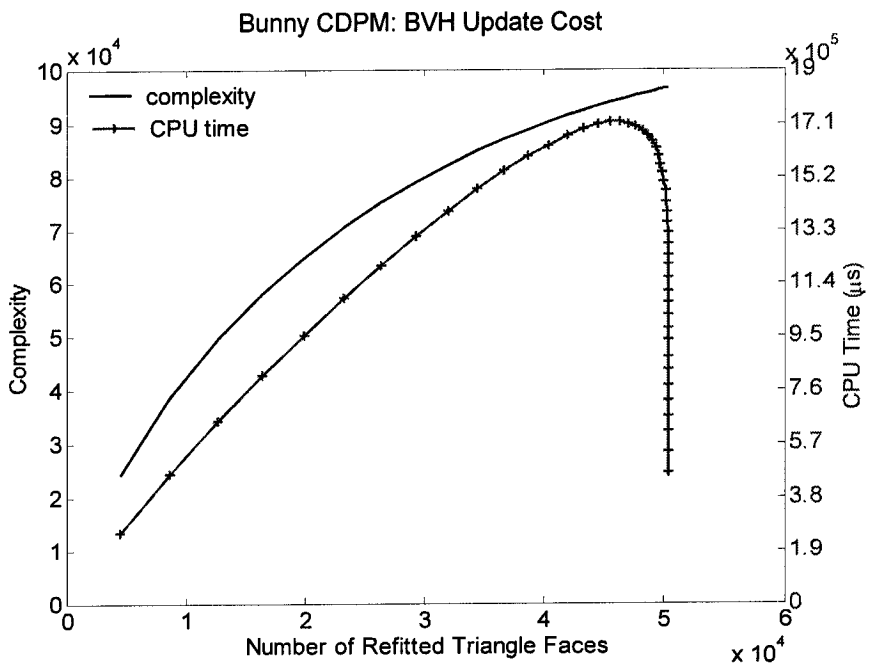
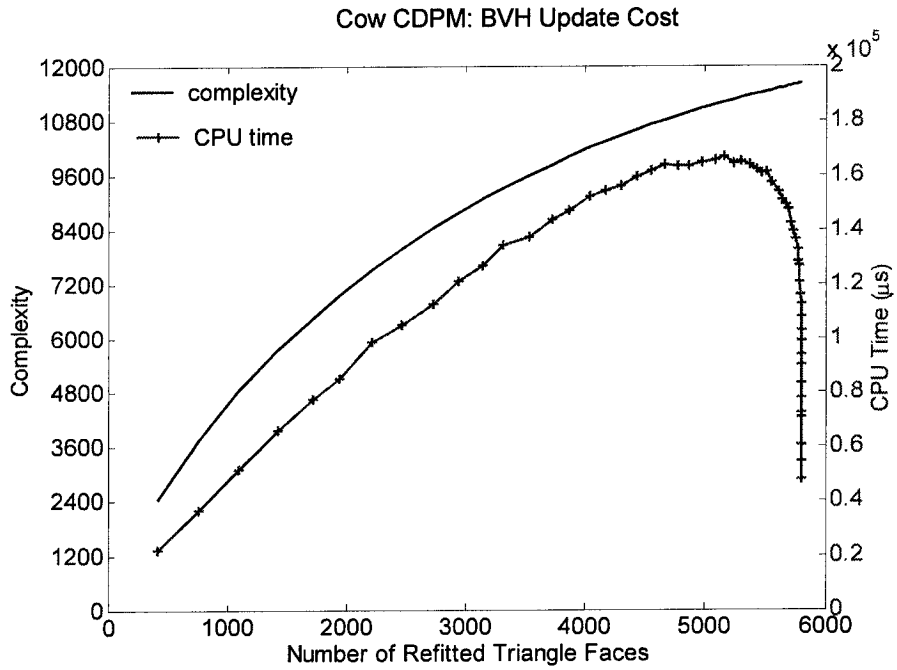


Figure 4.5 LOD Selection and Mesh Refinement for Cow Model. From (a) to (c), the cow model is refined from low LOD (704 faces, 355 vertices) to high LOD (5804 faces, 2904 vertices).

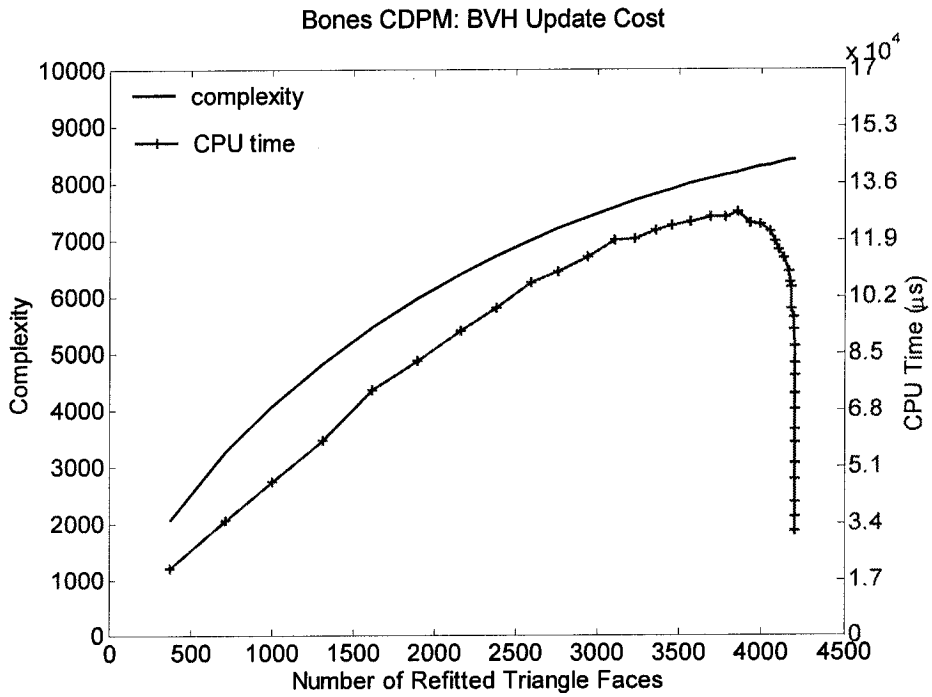


(a) Bunny model

Figure 4.7 Comparison of CPU Time and Complexity of the BVH Updating.

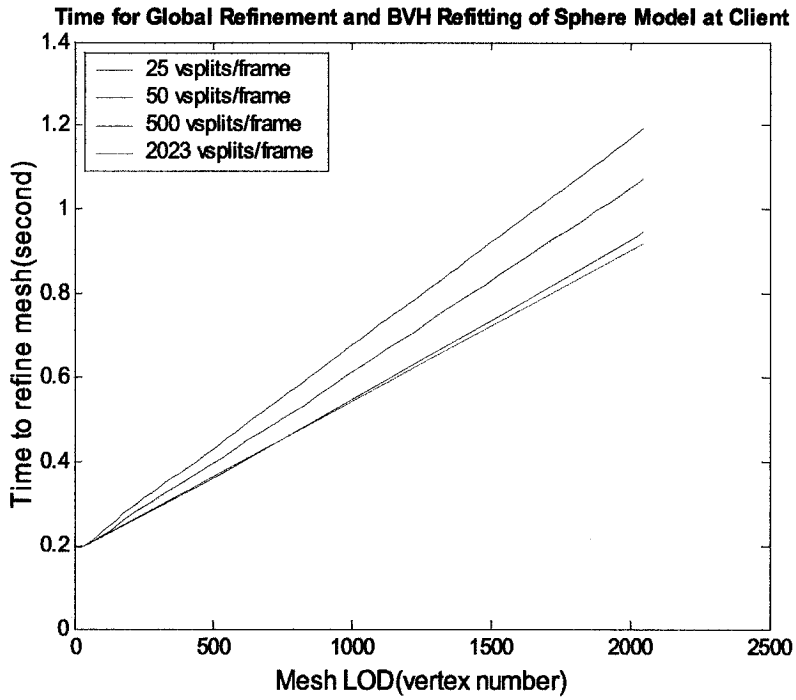


(b) Cow model

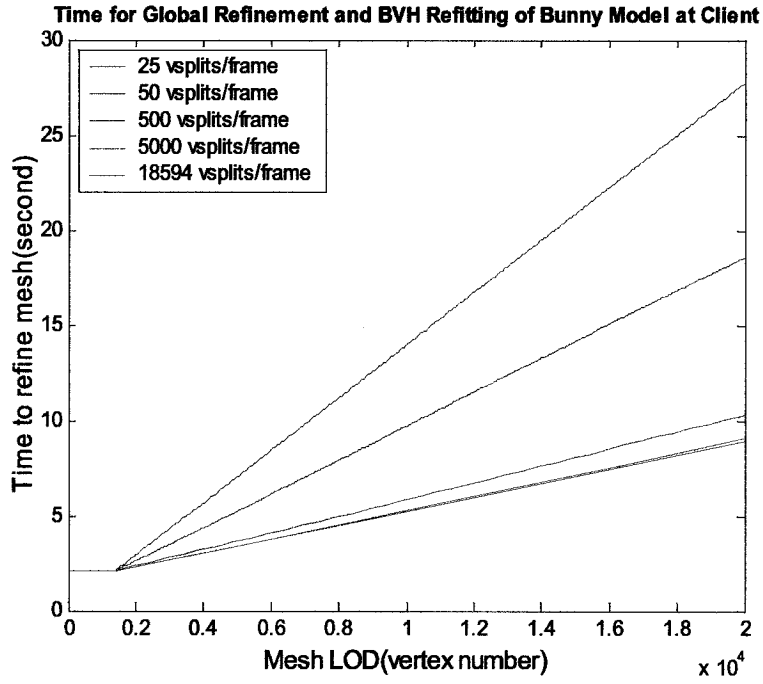


(c) Bones model

Figure 4.7 Comparison of CPU Time and Complexity of the BVH Updating.



(a)



(b)

Figure 4.8 Measurements of CPU Time in Walkthrough Application. (a) Time for mesh refinement and BVH updating on sphere model; (b) time for mesh refinement and BVH updating on bunny model.

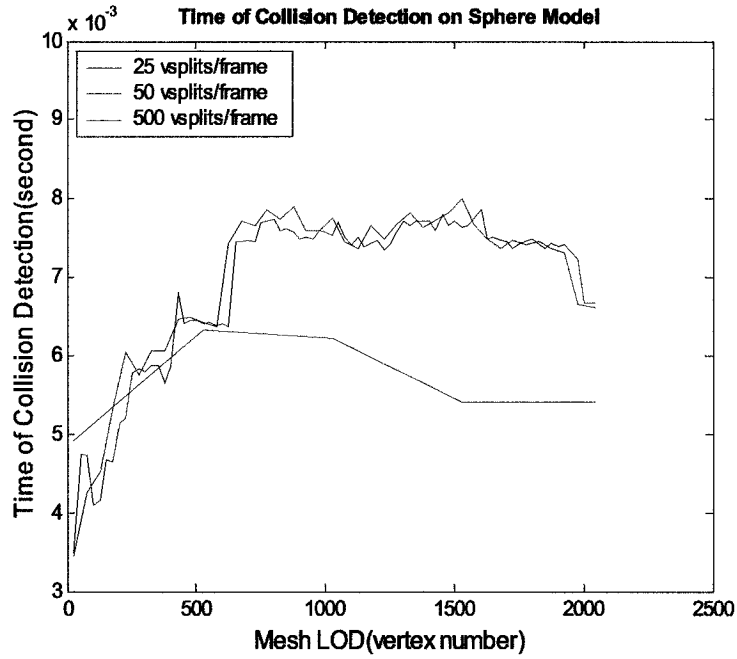


Figure 4.8 Measurements of CPU Time in Walkthrough Application. (c) Time for collision queries on sphere model; (d) time for collision queries on bunny model.

Chapter 5 Locally Refined Collision Detection

Locally refined collision detection is a new approach that is applied to large scale models, such as terrain models, where the collision typically occurs in a small area of the entire mesh surface. The main idea of the new approach is to perform collision queries only in those areas of the meshes where collisions are most likely to happen. In a flight simulation, for instance, as an aircraft flies over a terrain mesh the regions near the aircraft which are at high risk of collision would be selected for a collision query. The major benefit of this approach is that the mesh geometry of the other regions may not be required by the CD algorithm. Therefore for models with millions of polygons only a small portion of the data needs to be loaded from server to client. This can significantly reduce network traffic, offload the work load of the server, and accelerate simulation at the client. In addition, the accuracy of CD can be increased if the mesh in the selected regions is refined to a higher resolution. When relative position and orientation among objects changes, the selected regions are changed dynamically. The un-selected regions only need to be refined to the coarsest resolution. Thus, when refinement data can not be received by the client in time, CD is still supported, although at low accuracy. The AB-tree algorithm introduced in chapter 3 is used for collision query. A new space partitioned mesh representation (SPM) is presented in this chapter to fulfill the requirement of local mesh refinement. The complexity of a collision query on triangle meshes in the highest accuracy is affected mainly by two factors as introduced in section 2.4.3.2: N_{bv} and C_{bv}. N_{bv} is the number of BV overlap tests and C_{bv} is the cost of a BV overlap test. For AABB BVs, the cost of C_{bv} is constant. For simple BV types such as AABB and sphere, one primitive can be bounded by more than one BV and one BV can be

bounded by other BVs besides the BV in its parent node. Thus redundant BV overlap tests occur among these overlapped BVs. In my experience, N_{bv} is determined by not only the height of the BVH tree, but also the number of nodes in the tree. The simpler the BVH tree is, the fewer the redundant BV overlap tests that occur. For an AB-tree, N_{bv} is determined by not only the height of the active subtree but also the number of nodes inside the tree. If a collision occurs on the predicted contact area, the depth of the tree a collision query searches down equals the height of the active subtree. Since the mesh is locally refined, the number of active primitives on the mesh is very small relative to the total number of primitives of the entire model. Thus, the number of active leaves in the tree is small and the number of redundant BV overlap tests is small. In other words, the data volume of the predicted contact regions on the meshes affects the complexity of a collision query instead of the data volume of the entire meshes.

H-Collide is a framework for fast and accurate collision detection for haptic interaction. It's modeling method for complex models is similar to the Space Partitioned Mesh introduced in this chapter. A polygonal mesh is decomposed to uniform grids to efficiently deal with large storage requirements. An OBB tree is pre-computed for the polygons of each grid. The OBB BVH algorithm is used for collision query. The overlap test between the path swept out by the probe tip and an OBB is relatively simple. The performance of an implemented H-Collide system shows that the collision detection algorithm is 2 to 20 times faster than earlier algorithm. However, this approach does not take the limitations of cache memory space into consideration. Thus it does not apply to system in which the entire mesh data is not available in local storage but at a remote site.

One problem with the proposed multi-resolution collision detection is that in the approach described in Chapter 3 the collision queries only apply to globally refined meshes. The resolutions of the meshes cannot be adjusted locally in specified regions on the surface model. The complexity of a BVH collision query algorithm is mainly determined by the number of BV overlap tests while the accuracy of a collision detection algorithm is determined by the resolution of the meshes at contact location. The proposed CDPMs are globally simplified from traditional triangle meshes using the QEM algorithm. They are refined from coarse to fine in a uniform fashion. For such meshes the complexity of collision detection in the highest accuracy is proportional to the size of the entire meshes.

The proposed *local refinement collision detection* is an extension of the multi-resolution CD approach which supports CD on locally refined meshes. At the server, vertex split records for local refinement on the client subscribed areas on the mesh are collected and sent to the client. At the client, only those parts of the mesh that are most likely to collide with other objects are refined to full resolution at runtime. This means that the input size of the collision detection process does not changed significantly when the meshes are locally refined. When a comparable accuracy is achieved on the same large model, the locally refined collision detection approach runs faster than the global approach.

Locally refined collision detection approach is composed of five tasks:

1. Define the conditions to legally perform local refinement and prove their correctness;

2. Extend the CDPM representation to support local refinement;

3. Create a method to determine the collision detection regions;

4. Create a BVH collision query algorithm for multi-resolution models;

5. Define a "local refinement on demand" protocol between client and server.

Before discussing the design and implementation issues of these five tasks and how they are fitted into the DVE framework, some notations are given in the first section of this chapter.

5.1 Notation

M_n is an arbitrary mesh. M_0 is the base mesh of M_n. H is the header of a SPM.

$\{VSP_1, ..., VSP_n\}$ is a sequence of vertex split operations, denoted as g_0, which $i \leq n$ and $i > 1$.

VSP_i represents the ith vertex split operation in the sequence. VSP^* represents a set of vertex split operations.

g is a subsequence of vertex split operations g_0. g starts with VSP_1.

G is a set of all such sequences. $g \in G$.

M_G is a set of all meshes refined from M_0 by applying a subsequence g in G.

$(H, M_0, \{VSP_1, ..., VSP_n\})$ is a SPM representation of M_n.

$VSP(i, v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3})$ represents a parameterized VSP_i.

S is a set of all subsequences of legal vertex splits that starts from VSP_1.

M_s is a refined mesh obtained by applying to the base mesh M_0 a subsequence in S.

s is an element in S. G is a subset of S. $s \in S$ *and* $G \subseteq S$.

M_S is a set of all meshes refined from M_0 by applying a subsequence s of VSP operations.

M_G is a subset of M_S. M_G is called the set of globally refined meshes because the VSP operations are performed in the reverse order of edge collapse operations in the mesh simplification process. The meshes are simplified in such a process uniformly so that the primary features of the models are reserved even after significant simplification.

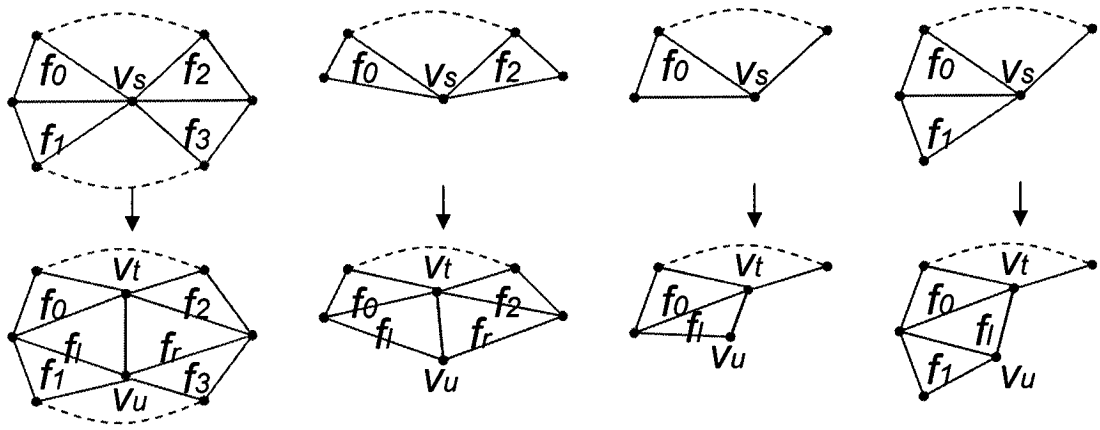


Figure 5.1 Vertex Split Configurations

Let M_L be a subset of M_S. $M_G \subseteq M_L$. Each mesh in M_L is produced from M_0 by applying a sequence of legal VSP operations. These sequences form a subset L of S. $L \in S$. Meshes in M_L are called locally refined meshes. A VSP is legal if the current mesh satisfies some conditions which will be introduced shortly.

$|V_0|$ represents the number of vertices in M_0. $|V_n|$ represents the number of vertices in M_n. V is the set of indices of all vertices appear in the vertex split sequence g_0. The size of V, denoted as $|V|$, is approximately twice that of $|V_n|$. $|V| = |V_0| + 2n$.

F is the set of indices of faces in a refined mesh. It is always a subset of the set of faces, F_n, in M_n.

5.2 Legal Vertex Split Operations

The idea of local mesh refinement was initially used to accelerate graphics rendering of large scale complex models by selectively refining meshes depending on viewing parameters. The proposed methods are called view-dependent refinement in general. Some earlier methods are used for height fields and parametric surfaces. The methods supporting efficient view-dependent LOD are based on hierarchical representations such as grid quadtrees [LKR+96,

TB94], quaternary triangular subdivisions [LDW97], and more general triangulation hierarchies [CPS95, DMP96, Sca90]. Because the view-dependent meshes created by these schemes have constrained connectivity, and therefore require more polygons for a given accuracy than so-called *triangulated irregular networks* (TIN's). View-dependent tessellation of parametric surfaces such as NURBS requires fairly involved algorithms to deal with parameter step sizes, trimming curves, and stitching of adjacent patches [AS94, KML95, RHD89]. The view-dependent progressive mesh proposed by Hoppe [Hop97] is an advanced method for local mesh refinement on arbitrary triangle manifold surface models. The absence of a rigid subdivision structure on the models allows more accurate approximations of object shapes than with existing schemes. He not only proposed a refinement algorithm for the progressive mesh, but also defined legal vertex split and legal edge collapse operations. Unfortunately, he didn't provide analytical proof of the correctness of the refinement algorithm. In this work, the conditions for legal vertex split and legal edge collapse proposed in [Hop97] are used in the proposed SPM refinement algorithm and the correctness of the algorithm is proved.

In the CDPM representation, an arbitrary triangle mesh M_n is simplified through a sequence of n edge collapse operations (ECOLs) to yield a much simpler base mesh M_0 (see Figure 1.5).

$M_n \rightarrow \ldots M_i \rightarrow \ldots \rightarrow M_1 \rightarrow M_0$

Because each ECOL has an inverse, called a vertex split operation (VSP), the process can be reversed as:

$M_0 \rightarrow M_1 \rightarrow \ldots \rightarrow M_i \rightarrow \ldots \rightarrow M_n$

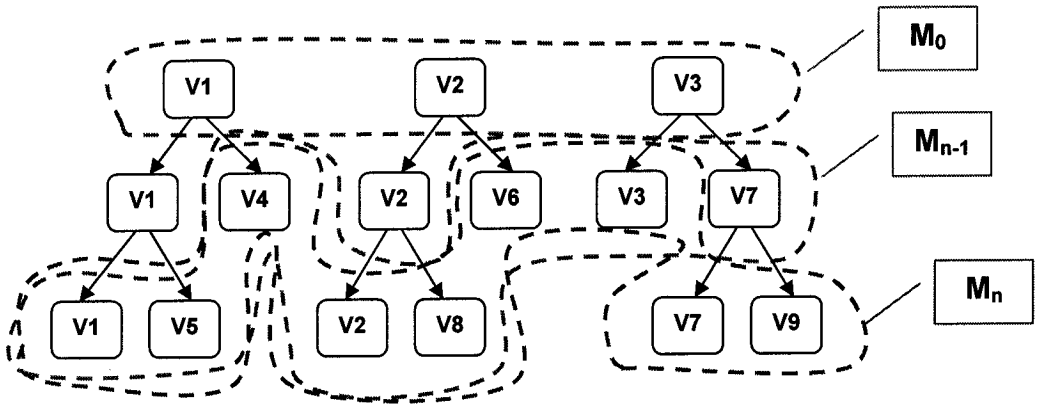


Figure 5.2 Illustration of a Base Mesh M_0, a Full Mesh M_n, and a Globally Refined Mesh M_{n-1} for CDPM

Each VSP, parameterized as VSP(i, v_s, v_t, f_l, f_2) modifies the mesh by introducing one new vertex v_t and two new faces f_l and f_2 (Figure 3.3). The resulting sequence of meshes M_0, ..., M_n is effective for global LOD control. To create localized LOD approximations, the CDPM representation has to be redefined because the information provided in each VSP record is not enough to refine the meshes to manifold mesh surfaces. The corresponding global mesh refinement algorithm for CDPM which is introduced in section 3.2.3 dose not work for local mesh refinement.

5.2.1 Define Vertex Split and Edge Collapse

The new definitions of VSP and ECOL are illustrated in Figure 5.1. Note that their effects on refined meshes are the same as CDPM VSP and ECOL operations but parameterized differently.

VSP(i, v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3}) is a parameterized VSP$_i$. A VSP transformation replaces the parent vertex v_s by two children vertices v_t and v_u. Two new faces f_l and f_r are created between the two pairs of neighboring faces (f_{n0}, f_{n1}) and (f_{n2}, f_{n3}) adjacent to v_s. To support manifold meshes with boundaries, there is at least one face and at most four faces in the set.

For manifold meshes without boundary, there are exactly four faces in the set. A vertex or a face is active if it exists in the locally refined mesh M_i. Since a $ECOL_i$ is the reverse of VSP_i, it can be parameterized similarly as ECOL(i, v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3}).

5.2.2 Vertex Hierarchy

The parent-child relation on vertices established during refining the mesh from M_0 to M_n in the vertex split sequence g_0 forms a vertex forest as illustrated in Figure 5.3. The vertices of a locally refined mesh and all their ancestors correspond to a set of subtrees of the trees in the hierarchy. The leaf nodes of the subtrees form a "refined front". This hierarchy is different from the vertex hierarchy introduced in section 3.2.3 in two aspects. See Figure 5.2 and Figure 5.3. First, an old vertex v_s is split to two new vertices v_t and v_u for SPM instead of splitting v_s to two vertices, one is a new vertex v_t the other is the old vertex itself v_s for CDPM. This difference contributes to the independence of each VSP in the SPM VSPs sequence. Second, the hierarchies for CDPM and for SPM are both constructed when M_0 is loaded. The root node of each tree inside the hierarchies represents a vertex in M_0. As more and more VSPs are loaded and performed, the trees in the hierarchies are grown. Whenever a VSP is performed, the hierarchy for CDPM generates two new nodes. Since the order VSPs are to be performed is uniquely defined by g_0, the hierarchy has only n different structures, where n equals the size of set G. A CDPM mesh can be refined to n LODs. However, for a SPM the sequences VSPs are loaded are not unique. Each sequence s is an element of set L, where L is defined as a set of all subsequences of legal vertex splits and $G \subseteq L$. The hierarchy has $|L|$ number of different structures which is no less than n. and the SPM can be refined to $|L|$ different LODs. $|L|$ is determined by the legal VSP conditions and the actual topology of the meshes.

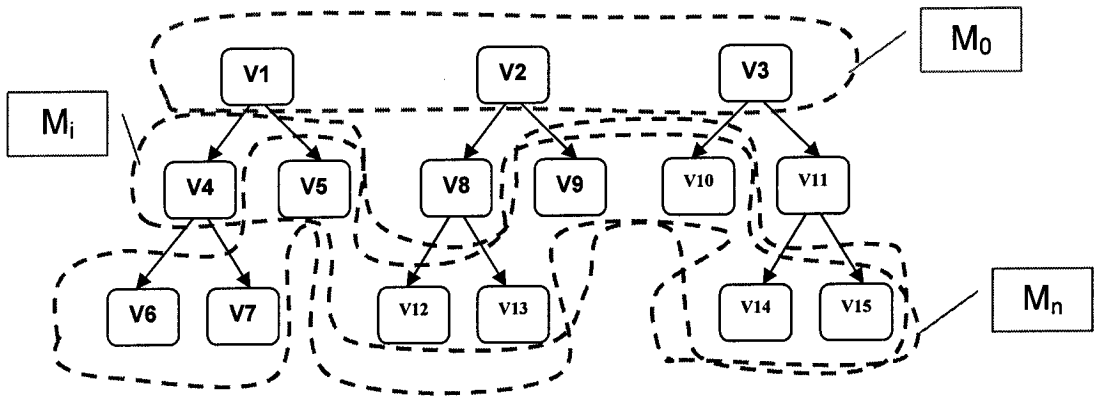


Figure 5.3 Illustration of a Base Mesh M_0, a Full Mesh M_n and a Locally Refined Mesh M_i for SPM. M_i is generated by a sequence of VSPs in a selected order.

Let a locally refined mesh M_i be defined as a mesh obtained by applying to M_0 a subsequence s of VSPs where $s \in S$. If s is an arbitrary subsequence of VSPs which has some illegal VSPs, then M_i may not correspond to a well-defined mesh. If VSPs in s are all legal, then the refined mesh must be well-defined. Some legal conditions are defined in the following section. These conditions are analogous to the vertex or face dependencies found in the hierarchical representations in [DMP96, LKR+96, XV96]. To support selected local refinement, it is necessary to consider not only vertex split, but also edge collapse. It is necessary to perform these operations in an order possibly different from that in the sequence g_0. A major concern is that a locally refined mesh obtained by performing a sequence of VSPs and ECOLs should be unique, regardless of the order the refinement operations that lead to it. Particularly, it should be a mesh in M_L.

5.2.3 Legal Vertex Split and Edge Collapse Conditions

The legal conditions have to be defined to fulfill two requirements. First, a legal VSP or a legal ECOL does not refine a mesh from manifold to non-manifold topology. This requirement is proved in theorem 3. Second, a mesh M_i is refined to a unique mesh M_j in

terms of geometry and topology no matter in which orders a set of legal VSPs and ECOLs are performed. This requirement is proved in theorem 4. It guarantees that a mesh M_0 is always refined to a mesh M_j no matter which areas of the mesh are refined first.

A vertex split operation, VSP_i or $VSP(i, v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3})$ is legal if

(1) v_s is a vertex on the current mesh, and

(2) The faces in the set $\{f_{n0}, f_{n1}, f_{n2}, f_{n3}\}$ are on the current mesh.

A edge collapse operation, $ECOL_i$ or $ECOL(i, v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3})$, which is the reverse of VSP_i is legal if

(1) v_t and v_u are vertices on the current mesh, and

(2) Faces f_{n0} and f_{n1} are adjacent to f_l, and f_{n2} and f_{n3} are adjacent to f_r in the configuration illustrated in Figure 5.1.

These conditions were first introduced in [Hop97].

Since a legal $ECOL_i$ is the reverse of a legal VSP_i, if $ECOL_i$ that performed after a sequence of VSPs s is legal and VSP_i is in s, then the mesh generated after $ECOL_i$ is the same as the mesh generated by the same sequence s without VSP_i.

5.3 Space Partition Mesh (SPM) Modeling

This section introduces the SPM representation and an algorithm used to generate SPMs.

5.3.1 SPM Representation

The SPM representation is an extension of the CDPM representation that supports local refinement. An SPM is composed of three parts (see Figure 5.4).

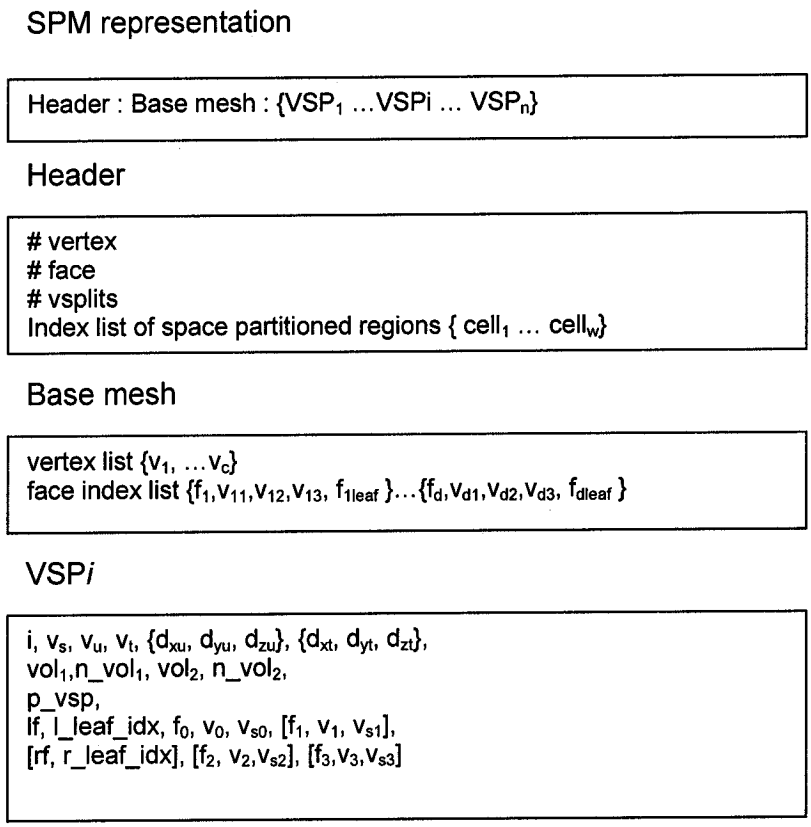


Figure 5.4 Illustration of SPM Representation

5.3.1.1 SPM Header

The SPM header, which is similar to the header of CDPM, contains mesh configuration information. The information helps receivers (or clients) to efficiently allocate memory and construct BVHs before the entire mesh is received. It also contains an index list of space partitioned regions $\{cell_1 \ldots cell_w\}$.

Space Partitioned Regions of the Mesh

In order to do local mesh refinement, an AABB BV is calculated for the original mesh. Then the BV is evenly space partitioned to 3D grids as illustrated in Figure 5.5. Each grid is indexed based on the coordinates of the partitioning planes along the axes of the BV. The set

of mesh primitives bounded by a grid is called a region. Since some grids in the BV may not be occupied by any primitive, no collision will occurs in those spaces. The collision prediction method introduced in section 5.6.2 predicts collision regions on the mesh in the current frame based on the grids in which collisions occur during previous frames. If a region bounded by a grid is predicted to collide with other objects, then the region is required to be refined to the highest resolution. An *index list of space partitioned regions* is used to collect all VSPs that split one old vertex to two new vertices in the specified region of full mesh M_n. The new vertices are represented by leaf nodes in the vertex hierarchy. In the index list {cell1, ...cell$_i$, ..., cell$_w$}, cell$_i$ represents the region on M_n that is bounded by the grid with index i. For a model that is partitioned to $D \times D \times D$ grids, w equals D^3. As illustrated in Figure 5.6, each element of the index list cell$_i$ links to a VSP$_j$ in g_0 which split one vertex v_s to two new vertices v_t and v_u. Here VSP$_j$ has two other fields vol$_1$ and vol$_2$. Each represents a region that the new vertices are located. Also n_vol$_1$ and n_vol$_2$ each links to a VSP in g_0 that has the same value in the field vol$_1$ or vol$_2$. As a result, cell$_i$ links to a list of VSP records. n_vol$_1$ and n_vol$_2$ are used to link all VSP records in the sequence g_0 which

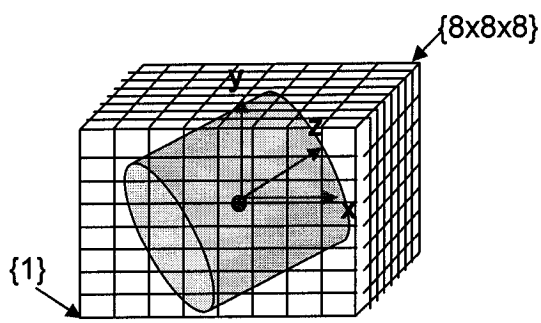


Figure 5.5 Illustration of Space Partitioning a Polygonal Model into 8x8x8 3D Grids

have the same vol_1 and vol_2 respectively.

The index list helps the server to collect VSP records that are used by the client to locally refine selected regions on the current mesh. The pseudo code of an algorithm to generate the index list is as follows. It runs on the vertex hierarchy data structure introduced in section 5.2.2. The hierarchy is constructed from the sequence of vertex split operations g_0, which is generated by the QEM mesh simplification algorithm upon the manifold mesh.

Procedure MeshPartition

1. Construct a vertex hierarchy for mesh refinement from vertex split sequence {VSP_1, ..., VSP_n}
2. Refine mesh to full resolution
3. Partition the bounding box of the mesh model into D x D x D 3D grids of equal size
3. Initialize each element of the index list of partitioned regions cell[x] to 0, where x is from 1 to w.
4. WHILE mesh is not refined to base mesh
5. Perform an edge collapse operation ECOL{i, v_s, v_u, v_t} on the vertex hierarchy in the reverse order of the vertex split sequence
6. Locate the region of Mn where v_u is contained and set the field vol_1 of VSP_i to the index of the region
7. Locate the region of Mn where v_t is contained and set the field vol_2 of VSP_i to the index of the region
8. ENDWHILE
9. FOR each node in the hierarchy corresponding to a VSP_j in the reverse order of sequence {VSP_1, ..., VSP_n}
10. SET n_vol_1 to cell[vol_1]
11. SET cell[vol_i] to j
12. SET n_vol_2 to cell[vol_2]
13. SET cell[vol_2] to j
14. ENDFOR

5.3.1.2 SPM Base Mesh

Each SPM base mesh has the same format as the base mesh of CDPM.

5.3.1.3 SPM VSPs Sequence

SPM has a sequence of VSPs. For each VSP, the fields i, $\{d_{xu}, d_{yu}, d_{zu}\}$, and $\{d_{xt}, d_{yt}, d_{zt}\}$ have the same meaning as the fields in CDPM. However, some new information is added into the VSP.

A vertex split operation, represented as VSP_i, splits one vertex v_s to two new vertices v_u and v_t. vol_1 is the index of the region containing v_u. vol_2 is the index of the region containing v_t. n_vol_1 links to the next VSP in the sequence $\{VSP_i, \ldots, VSP_n\}$, the field v_s of which is contained in the same region as v_u. n_vol_2 links to the next VSP in the sequence $\{VSP_i, \ldots, VSP_n\}$, the field v_s of which is contained in the same region as v_t. If v_u is not a vertex on M_n, than vol_1 and n_vol_1 are set to 0. If v_t is not a vertex on M_n, than vol_2 and n_vol_2 are set to 0. p_vsp is the index of a VSP which split one vertex to two new vertices, one of which is v_s. Here lf represents a generated new face and l_leaf_idx is the code of a AB-tree node. The BV of the node bounds face lf. The method to generate an AB-tree from a mesh is introduced in section 3.3. Here f_0 and f_1 represent the two pair-wised faces. v_0 and v_1 represent the vertices other than the two new split vertices on f_0 and f_1 respectively. They are in range [1, 3]. v_{s0} and v_{s1} are indices of VSPs which generate f_0 and f_1. {[rf, r_leaf_idx], [f_2, v_2, v_{s2}], [f_3,v_3,v_{s3}]} contains the information of the other two pair-wised faces. The fields inside bracket "[ ]" are optional.

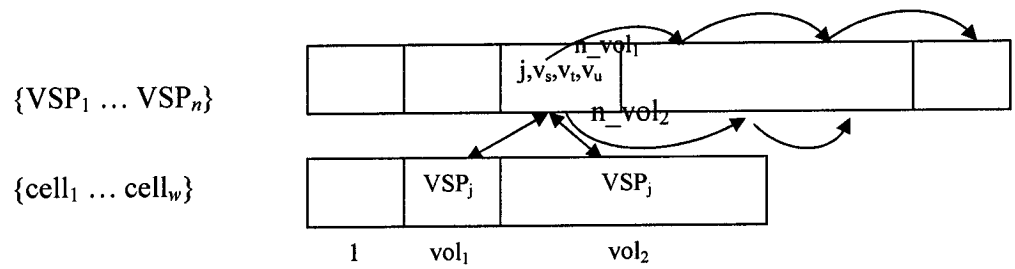


Figure 5.6 Index List of Space Partitioned Regions

5.3.2 SPM Multi-resolution Data Structure and Algorithm

There are two major tasks for the local refined collision detection at runtime: Mesh refinement and BVH CD. An AB-tree is used to complete the task of BVH CD. An introduction to the AB-tree can be found in section 3.3. The functionality and the basic operation of local SPM refinement have been discussed in section 5.2.2. This section gives a detailed description of the SPM multi-resolution data structure and the corresponding refinement algorithm.

5.3.2.1 Client

SPM collision detection is designed for distributed environments. The SPM multi-resolution data structure at the client is similar to the data structure for CDPM. It is composed of four parts: vertex hierarchy, vertex list, primitive list, and a sequence of performed VSPs. The two major differences of vertex hierarchy are discussed in section 5.2.2. The primitive list and the vertex list are the same for both CDPM and SPM, which are introduced in section 3.2. The sequence of performed VSPs records the order the nodes in the hierarchy are extended from the root nodes. The two types of mesh refinement operations for SPM, vertex split and edge collapse, have the same effect on the refined meshes comparing with the operations for CDPM. They are only different in parameterization. Each node in the hierarchy represents a vertex. Only the nodes belongs to the "refined front" are expanded or folded dynamically upon vertex split or edge collapse. The "refined front" and the primitive list define a mesh at the current resolution. The leaf nodes of the hierarchy form a "loaded front". It defined a mesh refined by all loaded VSPs. The basic structure of a vertex hierarchy is described below (explanations of the individual fields follow):

Vertex Hierarchy

```
struct hierarchy_tree{
        tree_node *         rootPtr;
        index_type          treeIdx;
};
```

- rootPtr: a pointer to tree root node that represents a vertex in mesh M_0.
- treeIdx: index of trees in the hierarchy.

```
struct hierarchy_tree       treeList[NUM_M0_VERTICES];
```

- treeList: a list of trees which compose a vertex hierarchy.

```
struct tree_node {
        triangle *          neighborFaces;
        tree_node *         left;
        tree_node *         right;
        tree_node *         parent;
        vertex_item *       vsplitPtr;
};
```

A tree node represents a vertex split operation which split v_s to v_u and v_t.

- neighborFaces: a pointer to a list of triangles that indent to v_s.
- left: a pointer to left child tree node
- right: a pointer to right child tree node
- parent: a pointer to parent tree node
- vsplitPtr: a pointer to a vertex_item in the vertex list

```
struct vertex_list {
        tree_node *         headPtr;
        size_type           numVertices;
};
```

A vertex_list is a list of links to tree node representing VSPs in the sequence {VSP_1 to VSP_n}

- headPtr: a pointer to the first tree node which represents VSP_1.
- numVertices: the number of vertices in the vertex list which is equal to the number of vertices in M_n.

Sequence of Performed VSPs

The *sequence of performed* VSPs is designed to record the order the vertices that are bounded by the refined front are split, and to quickly locate nodes on the refined front to be folded or unfolded.

Primitive List

The *primitive list* in a simple form is a sequence of active triangles. It has the same structure as a CDPM.

Vertex List

The *vertex list* is a list of coordinates of active vertices. It has the same structure as CDPM.

Local Refinement Operations

In the distributed environments, tasks at the client include collision prediction, collision region VSPs subscription, receiving VSPs data, SPM multi-resolution refinement, and AB-tree collision queries. The fundamental mesh refinement operations associated with the SPM multi-resolution data structure are LocalEdgeCollapse() and LocalVertexSplit().

Functions LocalVertexSplit is called when the predicted collision regions are different from current collision regions. It calls function CollectVSplits to collect all VSPs to be refined to from the index list of partition regions in the received mesh header. Then for each collected VSP, function VSplitLegal is called to test whether the legal VSP conditions introduced in section 5.2.3 are satisfied. If the conditions are satisfied, then function VertexSplit is called to perform the VSP on the multi-resolution data structure by calculating the coordinates of newly generated vertices, updating the vertex list, adding new faces to the primitive list, and expanding the node representing the old vertex to two new nodes. Otherwise, if an ECOL needs to be performed to make the VSP legal, then ECollapseLegal is called to test the legal conditions of ECOL. If another VSP needs to be performed to the VSP legal, then VSplitLegal is called again. Function CollectVSplits is a recursive process. Figure 5.7 illustrates how this algorithm works. It terminates when all vertices in the predicted regions on M_n become active. Function LocalEdgeCollapse() is called when the current collision

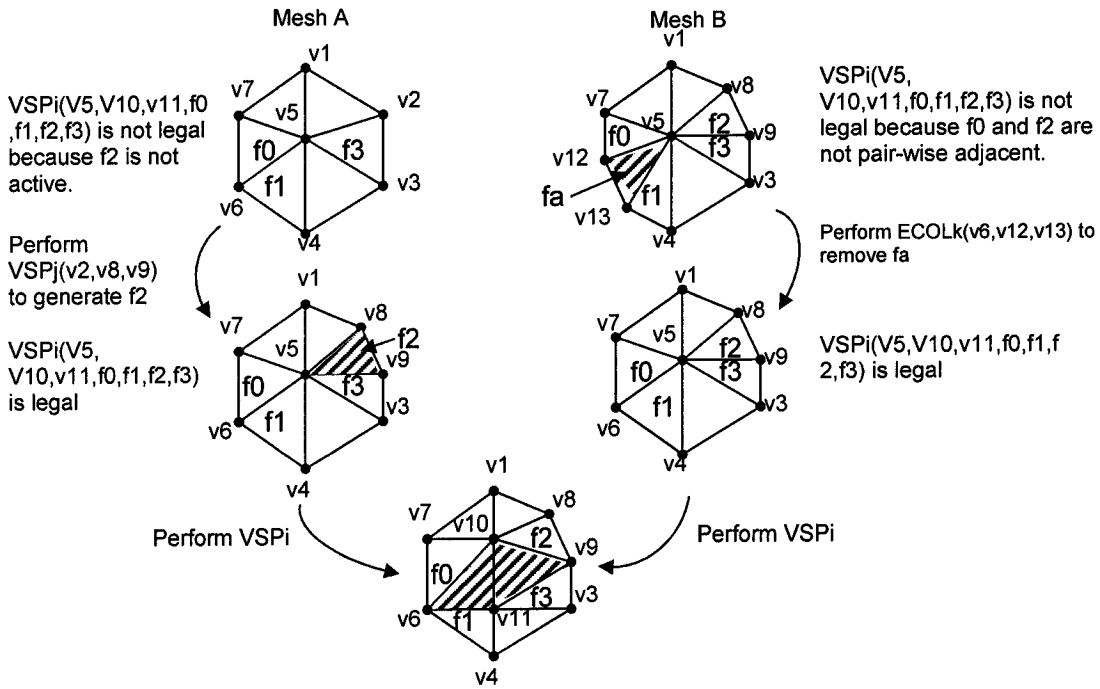


Figure 5.7 Illustration of Collecting Legal VSPs and ECOLs for Local Refinement. In mesh A, VSPj is required to be performed to make VSPi legal; In mesh B, ECOLk is required to be performed to make VSPi legal.

regions are predicted not to collide in the next frame. It calls function CollectECollapse to collect all ECOLs to be refined from the index list of partition regions in the received mesh header. CollectECollapse is another recursive process which performs all necessary VSPs and ECOLs to refine the predicted regions to the lowest LOD. Both LocalVertexSplit and LocalEdgeCollapse update the BVH for collision queries if the mesh is refined. The pseudo code of related functions for mesh refinement is give as follows.

Procedure **LocalVertexSplit**

1. CALL **CollectVSplits** with regions to be refined to higher resolution
2. Update BVH for collision queries with collected refined primitives

Procedure **CollectVSplits**

1. FOR each region to be refined
2. Walk through partition regions list inside mesh header and collect all vertex split records, in which the vertex to be split is in the region.
3. ENDFOR

4. FOR each collected VSPi
5. CALL **VSplitLegal** with VSPi
6. ENDFOR

Procedure **VSplitLegal** (vsp)
1. IF vsp is not a legal vertex split operation THEN
2. Traverse the vertex hierarchy and collect all tree nodes need to be folded or unfolded before vsp becomes legal.
3. FOR each of the collected node
4. IF need to unfold it THEN
5. CALL **VSplitLegal** with the vertex split record represented by the node
6. ELSE
7. CALL **ECollapseLegal** with the vertex split record represented by the node
8. ENDIF
9. ENDFOR
10. ELSE
11. CALL **VertexSplit** with vsp
12. ENDIF

Procedure **VertexSplit** (vsp)
1. Calculate the coordinates of split vertices v_u and v_t
2. Divide the faces indent to v_s into two sets. One set of faces indent to v_u. Another set of faces indent to v_t.
3. Update primitive list with new faces
4. Extend vertex hierarchy node representing vsp to two children and update vertex list.

Procedure **LocalEdgeCollapse**
1. CALL **CollectECollapses** with regions to be refined to lower resolution
2. Update BVH for collision queries with collected refined primitives

Procedure **CollectECollapses**
1. FOR each region to be refined
2. Walk through partition regions list inside mesh header and collect all vertex split records, in which the vertex to be split is in the region.
3. ENDFOR
4. FOR each collected VSPi
5. CALL **ECollapseLegal** with VSPi

6. ENDFOR

Procedure **ECollapseLegal** (vsp)

1. IF the reverse of vsp is not a legal edge collapse operation THEN
2. Traverse the vertex hierarchy and collect all tree nodes need to be folded or unfolded before the reverse of vsp becomes legal.
3. FOR each of the collected node
4. IF need to unfold it THEN
5. CALL **VSplitLegal** with the vertex split record represented by the node
6. ELSE
7. CALL **ECollapseLegal** with the vertex split record represented by the node
8. ENDIF
9. ENDFOR
10. ELSE
11. CALL **EdgeCollapse** with vsp
8. ENDIF

Procedure **EdgeCollapse** (vsp)

1. Calculate the coordinates of vertex v_s to be merged from v_u and v_t
2. Merge the faces indent to v_s and faces indent to v_t in one set, and remove merged faces from the set
3. Update primitive list with refined faces
4. Fold vertex hierarchy node represented by vsp from the node's two children and update vertex list.

5.3.2.2 Server

The SPM multi-resolution data structure at the server is designed for fast legal VSPs collection. It is composed of four parts: an index list of space partitioned regions defined in mesh header, a vertex hierarchy which is the same as the vertex hierarchy at the client, an index list of sent out VSPs, and a sequence of VSPs defined in the model.

A set of the regions on mesh M_n that are most likely to collide with other objects are predicated at the client. The regions are subscribed by the client by sending a list of indices to the server. The server collects all VSPs that are necessary to refine those regions on the

mesh defined by the "loaded front" of the vertex hierarchy to the highest LOD. Then the server sends the collect data to the client. The server also maintains a list of indices of VSPs that are already sent to the client and a vertex hierarchy similar to the one at the client. With the localized approach, a VSP operation is delayed at the client until it is feasible and necessary for collision query. The effect of performing the VSPs is to expand the current mesh in the subscribed local regions of interest.

Index list of space partitioned regions

```
index_type      regionList[D x D x D];
```

- regionList: each element of a list represents a partitioned regions in the bounding box of the model. The value of each element is an index of the first vsp in the vertex split sequence which split a vertex contained in the region.

Index list of sent out VSPs

```
index_type      sentVSPList[n];
```

- n: the number of VSP records in SPM.

The List of VSPs

```
struct vsplit_record vsplitList[#vsp];
struct vsplit_record {
        index_type      vsp;
        index_type      vu;
        index_type      vt;
        float           deltaLeft;
        float           deltaRight;
        index_type      vol1;
        index_type      nextVol1;
        index_type      vol2;
        index_type      nextVol2;
        index_type      parentVsp;
        vector<split_face*> adjFace;
        index_type      lr_size;
};
```

- vsp: vertex split record index

- vu, vt: index of split vertices v_u and v_t. In this VSP, v_s is split to v_u and v_t.
- deltaVu: the difference in coordinate from v_s to v_u.
- deltaVt: the difference in coordinate from v_s to v_t.
- vol1, vol2: region indices of v_u and v_t.
- next_vol1, next_vol2: region indices of split vertices of the next VSP in the sequence $\{VSP_1, \ldots, VSP_n\}$.
- parentVsp: a link to the VSP which generate v_s.
- adjFace: a link to a list of pair-wise adjacent face structures.
- lr_size: number of new faces generated.

```
struct split_face {
        index_type          lf;
        vector<index_type> f0;
        vector<index_type> vs0;
};
```

- lf: index of a generated new face
- leaf_idx: the code of the face lf in the BVH
- f0: a list of indices of pair-wised adjacent faces to lf.
- vs0: an index list of VSPs which generate the faces in f0.

Legal VSPs collection operations

In the distributed environments, tasks at the server include loading mesh header and base mesh of subscribed models from secondary storage to main memory, receiving collision regions index list from the client, collecting VSPs required for mesh refinement, and sending the collected data to the client. The fundamental operation associated with the SPM multi-resolution data structure is collecting VSPs required for mesh refinement. The algorithm of the task is designed in function SrCollectVSplits(). SrCollectVSplits() is called when an index list of subscribed mesh regions are received by the server. First, it traverses the index list of the space partitioned regions of the model and collects all VSPs that split to new vertices on the regions of mesh M_n. Then for each collected VSP_i, function SrVSplitLegal is called to collect all VSPs that have not been sent out and that need to be performed to make VSP_i legal. Function SrVSplitLegal defines a recursive process. It creates a stack and pushes

the input VSP_i onto the stack. Then it gets the top VSP_k from the stack. If VSP_k is legal, SrSplit() is called to refine the vertex hierarchy. Then VSP_k is popped from the stack. Otherwise, the VSPs that generates the missing pair-wise adjacent faces for VSP_k are pushed onto the stack. Then the algorithm gets the top VSP from the stack again. This recursive process continues until the stack is empty, which means all the VSPs that makes the input VSP legal are found. SrVSplitLegal is a simple case of VSplitLegal for the client, which does not consider edge collapse operations because the mesh maintained by the vertex hierarchy at the server is never refined to lower LOD. Mesh A in Figure 5.7 illustrates the recursive process.

The pseudo code of the algorithm is as follows.

```
Procedure SrCollectVSplits (regionSet)
1. FOR each region in regionSet
2.      Walk through partition regions list inside mesh header and collect all vertex split
records, in which the vertex to be split is in the region.
3. ENDFOR
4. FOR each collected vsp_i
5.   CALL SrVSplitLegal with i
6. ENDFOR

Procedure SrVSplitLegal (i)
1.  Stack.push ( i )
2.  WHILE Stack is not empty
3.     SET i to Stack.top
4.     SET vspi to vspList[i]
5.     IF vspi.isSent is TRUE THEN
6.        Stack.pop()
7.     ELSE IF vspi.parentVsp AND vspList[vspi.parentVsp].isSent is FALSE THEN
8.        Stack.push (vspi.parentVsp)
9.     ELSE
10.       CALL SrCheckLegal with i and &illegalVspList returning legal status  of vspi
11.       IF vspi is legal THEN
```

```
12.          Stack.pop()
13.          CALL SrSplit(i) to split vertex in vertex hierarchy
14.          SET vspList[i].isSent to TRUE
15.          sendVspList.insert( i )
16.       ELSE
17.          FOR each element j in illegalVspList
18.             Stack.push( j )
19.          ENDFOR
20.       ENDIF
21.    ENDIF
22.  ENDWHILE
```

Procedure **SrCheckLegal**(i, * illegalVspList)

1. INIT isLegal to TRUE
2. FOR each new face l generated by vspList[i]
3. FOR each of the pair-wise adjacent faces f_0 of l
4. IF VSPk who generates f_0 has not been sent THEN
5. SET isLegal to FALSE
6. illegalVspList.insert(k)
7. ENDIF
8. ENDFOR
9. ENDFOR
10. Return isLegal

Procedure **SrSplit** (i)

1. SET VSPi to vspList[i] which splits v_s to v_u and v_t
2. Calculate the coordinates of split vertices v_u and v_t
3. Divide the faces indent to v_s into two sets. One set of faces indent to v_u. Another set of faces indent to v_t.
4. Update primitive list with new faces
5. Extend vertex hierarchy node representing VSPi to two children and update vertex list.

The correctness of the local refinement algorithm introduced in this section can be proved by the following properties.

Theorem 3

If the faces f_{n0}, f_{n1}, f_{n2}, and f_{n3} of a vertex split record, VSP(i, v_s, v_t, v_u, f_l, f_r, $f_{n0}, f_{n1}, f_{n2}, f_{n3}$), are on the mesh M_i, then the faces are pair-wise adjacent to v_s.

This theorem proves that if a vertex is split and its vertex split operation is legal, then the mesh must be refined to a manifold mesh.

Theorem 4

A locally refined mesh produced by performing a set of legal VSP operations is unique, regardless of the order of the VSP operations performed which lead to the mesh. In a particular case, when all VSP records in {VSP_1, ..., VSP_n} are performed legally, the refined meshes equal M_n regardless of the order of the VSP*s* performed.

To support local refinement, it is necessary to consider performing legal VSP operation in an order possibly different from that in the M_G. Theorem 4 states that no matter in what order the VSPs in the sequence {VSP_1, ..., VSP_n} are performed, the mesh is always refined from M_0 to a unique mesh M_n. The proof for this theorem is given in the next section.

5.4 Properties and Proofs

Four properties need to be proved before proving theorem 3 and theorem 4. The properties are defined in lemmas 2, 3, 4, and 5.

Lemma 2 and lemma 4 are proved by contradiction. Lemma 3 is proved based on lemma 2 by contradiction. Lemma 5 is approved based on lemma 3 by contradiction. Theorem 3 is approved based on lemma 3 and lemma 4 by mathematical induction. Theorem 4 is proved based on lemma 5. The uniqueness of the meshes is proved from two aspects: topology (uniqueness proof) and geometry (direct proof).

Definitions:

Face f_1 is a neighbor of face f_2 if f_1 and f_2 share one edge.

A path from v_1 to v_2 on a mesh is a sequence of edges that connect vertex v_1 to vertex v_2. The length of the path is the number of edges on the path, which is the smallest among all possible sequences.

Lemma 2

Applying a sequence of legal VSPs to M_0 generates M_a. After applying VSP_i to M_i, if faces f_1 is not a neighbor of f_2, then f_1 can never be a neighbor of f_2 after any of the following VSPs.

Proof

If f_1 and f_2 are not neighbors, at most one of the paths that connect the vertices of f_1 to the vertices of f_2 has length equal to 0. Any VSP_i splits one vertex to two new vertices and generates a new edge connecting the two new vertices which refines one mesh M_i to another mesh M_{i+1}. The length of any path connecting a pair of vertices on M_i does not decrease in M_{i+1}. If a path connecting a vertex of f_1 to a vertex of f_2 in M_i is no less than 1, it can never be less than 1 after any legal VSP. Therefore, f_1 can never be a neighbor of f_2 after any of the following VSPs (proof by contradiction). See Figure 5.8. □

Lemma 3

Applying a sequence of legal VSPs to M_0 generates M_a. After applying VSP_i to M_i, if v_1 is not a vertex of face f_1, then v_1 or vertices split from v_1 directly or indirectly can never

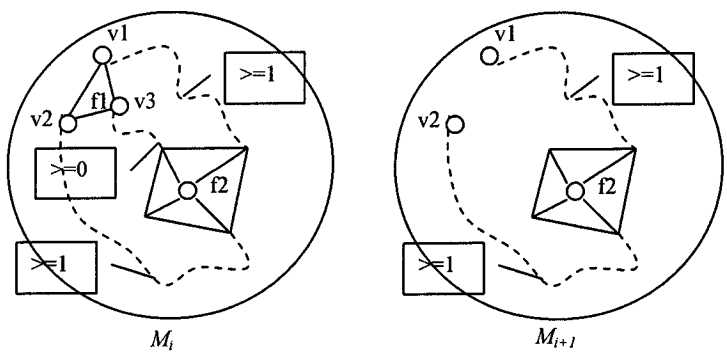


Figure. 5.8 Illustrate the Proof of Lemma 2

vertices of f_1 after any of the following VSPs.

Proof

If v_1 is not a vertex of f_1, there must be a path connecting v_1 to one of f_1's vertices, v_f, the length of which is no less than 1. As proved in lemma 2, the length of any path connecting a pair of vertices on M_i does not decrease in M_{i+1}. Therefore the path connecting v_1 to v_f can never be decreased after any of the following VSPs. If VSP_i splits v_1 to two new vertices v_1' and v_1", the length of the paths connecting v_1' and v_1" to v_f are equal to or one edge larger than the path connecting v_1 to v_f. If VSP_i splits v_f to two new vertices v_f' and v_f", the length of the paths connecting v_1 to v_f' and v_f" are equal to or one edge larger than the path connecting v_1 to v_f. Thus, v_1 or vertices split from v_1 can never be a vertex of f_1 after any of the following VSPs (proof by contradiction). □

Lemma 4

Applying a sequence of legal VSPs to M_0 generates M_a. No two VSP*s* in VSPs have the same set of pair-wise adjacent faces. Let VSP(i, v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3}) represents VSP_i and VSP(j, v_s', v_t', v_u', f_l', f_r', f_{n0}', f_{n1}', f_{n2}', f_{n3}') represents VSP_j, where $i, j \geq 1 and i, j \leq n$. $\{f_{n0}, f_{n1}, f_{n2}, f_{n3}\} \neq \{f_{n0}', f_{n1}', f_{n2}', f_{n3}'\}$.

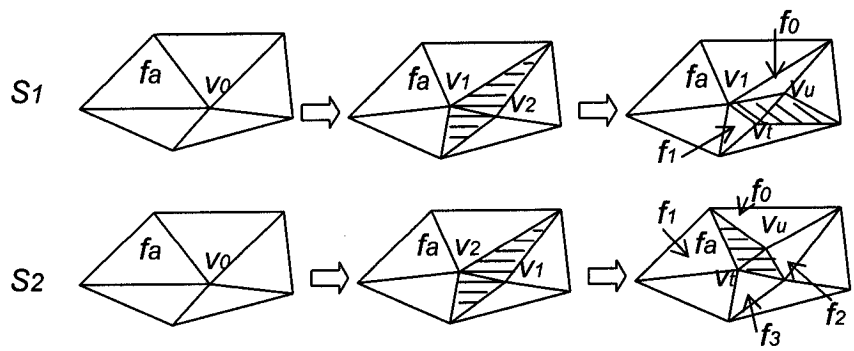


Figure 5.9 Illustration of the Proof of Lemma 4. Top: meshes generated during refinement sequence S_1; Bottom: meshes generated during refinement sequence S_2.

Proof

Let VSP_i be performed before VSP_j. VSP_i generates two new faces f_l and f_r as shown in Figure 5.9. f_l breaks the adjacency of $\{f_{n0}, f_{n1}\}$. f_r breaks the adjacency of $\{f_{n2}, f_{n3}\}$. VSP_j generates two new faces f_l' and f_r'. f_l' breaks the adjacency of $\{f_{n0}', f_{n1}'\}$. f_r' breaks the adjacency of $\{f_{n2}', f_{n3}'\}$. If $\{f_{n0}, f_{n1}, f_{n2}, f_{n3}\}=\{f_{n0}', f_{n1}', f_{n2}', f_{n3}'\}$, then $\{f_{n0}', f_{n1}'\}$ and $\{f_{n2}', f_{n3}'\}$ are not pair-wise adjacent after VSP_i. Therefore, VSP_j can not be a legal vertex split (proof by contradiction). □

Lemma 5

Let S_1 and S_2 be two legal vertex split sequences. $S_1 \in S$ and $S_2 \in S$. $|S_1| = n$ and $|S_2| = n$. Let the three vertices of a face f be denoted as $v_{f[q]}$, q=1, 2, 3. $f[q]$ represents an index of the vertices. It might change during mesh refinement. Let M_{s1} be the mesh generated at time t_1 of the refinement procedure of S_1 and M_{s2} be the mesh generated at time t_2 of the refinement procedure of S_2. If there is a face, f_a, on both M_{s1} and M_{s2}, then $v_{f[q]}$ in M_{s1} and $v_{f[q]}$ in M_{s2} have ancestor / descendant relationship, which means one vertex is split from another vertex directly or indirectly.

Proof

Let v_a be $v_{f[q]}$ when f_a was initially generated. Then v_a is split to other vertices in the order illustrated in Figure 5.10. Since the set of VSPs is the same in both S_1 and S_2, the order the vertices are split is the same too. If VSP_k splits v_0 to v_1, then let v_2 be $v_{f[q]}$ in f_a of M_{s1}', one of the meshes generated by S_1. By following the path illustrated in Figure 5.10, a legal vertex split operation $VSP(i, v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3})$ splits v_s to v_t and v_u and generates M_{s1}'', another mesh generated by S_1. Since VSPi is legal, v_s must be a vertex shared by the pair-wise adjacent faces $\{f_{n0}, f_{n1}, f_{n2}, f_{n3}\}$. Assume $VSP(j, v_0, v_1, v_2, f_l', f_r', f_{n0}', f_{n1}', f_{n2}', f_{n3}')$ splits

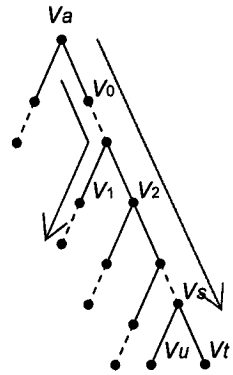


Figure 5.10 The Sequence a Vertex is Split. The right path shows the sequence v_0 is split to v_t which is unique for any legal VSPs sequence.

v_0 to v_1 and v_2, and v_1 equals $v_{f[q]}$ in f_a of M_{s2}', one of the meshes generated by S_2. VSP$_j$ splits the faces sharing v_0 to two groups, grp_1 and grp_2. grp_1 shares v_1 and grp_2 shares v_2. Faces in grp_2 can never be neighbors to any faces in grp_1 in the following VSPs in S_2 (proved by lemma 3). When VSPi happens in the sequence S_2, VSPi is legal. Therefore $\{f_{n0}, f_{n1}, f_{n2}, f_{n3}\}$ must shares v_s. Since v_s is a descendent of v_2, faces sharing v_s must not share faces in grp_1. If VSPi in S_1 is legal, then at least two pair-wise adjacent faces are missing in the faces sharing vs. Therefore, VSPi in the sequence S_2 becomes illegal. The order vertex indices $v_{f[q]}$ of face f_a appears in S_1 is the same as the order the indices appears in S_2. Thus, if there is a face, f_a, on both M_{s1} and M_{s2}, then $v_{f[q]}$ in M_{s1} and $v_{f[q]}$ in M_{s2} have ancestor / descendant relationship (proof by contradiction). □

Theorem 3

If the faces f_{n0}, f_{n1}, f_{n2}, and f_{n3} of a legal vertex split record, VSP(i, v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3}), are on the mesh M_i, then the faces are pair-wise adjacent to v_s.

Proof

Let M_k be a mesh refined from M_0 in a sequence of k legal VSPs.

1. When k =1, let the 1st VSP be VSP$_i$. $\{f_{n0}, f_{n1}, f_{n2}, f_{n3}\}$ must be pair-wise adjacent to v_s.

Since $k = 1$, v_s, f_{n0}, f_{n1}, f_{n2}, and f_{n3} must be on M_0. Since VSP$_i$ is legal in sequence g_0, f_{n0}, f_{n1}, f_{n2}, and f_{n3} must be pair-wise adjacent on M_0. Otherwise, they can never be pair-wise adjacent in any vertex split operation in any VSP sequence (proved by lemma 3).

2. When k>1, let kth legal VSP be VSPi. VSPi refines the mesh M_{k-1} to M_k. Assume {f_{n0}, f_{n1}, f_{n2}, f_{n3} } must be pair-wise adjacent to v_s on M_{k-1}. Let the k+1th legal VSP be VSP(j, v_s', v_t', v_u', f_l', f_r', f_{n0}', f_{n1}', f_{n2}', f_{n3}'). Assume {f_{n0}', f_{n1}', f_{n2}', f_{n3}'} are not pair-wise adjacent to v_s'. Since the first k VSPs are all legal and no one has the same set of pair-wise faces as VSP$_j$ (proved by lemma 4), no VSP refine the faces {f_{n0}', f_{n1}', f_{n2}', f_{n3}'} to not adjacent. Thus they must not be adjacent on M_0. Therefore, VSPj is not a VSP in the sequence g_0. It is proved by contradiction that { f_{n0}', f_{n1}', f_{n2}', f_{n3}'} must be pair-wise adjacent to v_s' (proof by mathematical induction). □

Theorem 4

A locally refined mesh produced by performing a set of legal VSP operations is unique, regardless of the order the VSP operations are performed on the mesh. In a particular case, when all VSP records in {VSP$_1$, ..., VSP$_n$} are performed legally, the refined meshes equal M_n regardless of the order the VSPs are performed.

Proof

1. Topology

If M_a and M_b are refined from M_0 by the same set of VSP, s, then sets of faces on M_a and M_b are equal because each VSP generates two new faces. It is proved by lemma 5, if a face is on two meshes, the vertices of the face on the two meshes have ancestor / descendent relationship. Let $f[q]$ be the index of a vertex of f on M_a, and $f[q]$ be the index of a vertex of f on M_b. q=1, 2, 3. If $f[q]$ on M_a is an ancestor of $f[q]$ on M_b, then there must be some VSPs in

s that split $f[q]$ in M_a to $f[q]$ on Mb directly or indirectly. Thus $f[q]$ on M_a is not an ancestor of $f[q]$ on M_b. It is easy to prove that it can neither be a descendent to $f[q]$ on M_b. Therefore the vertex indexed $f[q]$ on M_a is the same vertex indexed $f[q]$ on M_b. M_a and M_b has the same set of vertices. The topology is the same on M_a and M_b (uniqueness proof).

2. Geometry

It is proved in lemma 5, that the same vertex on any legally refined mesh is split from the same sequence of vertices. For any VSP which split v_s to v_u and v_t, the coordinates of v_u and v_t are uniquely determined by the coordinates of v_s. Thus, by following the same sequence, the coordinates of a split vertex is always the same on any refined mesh. The geometry of M_a and M_b which is determined by the coordinates of the vertices on the meshes are the same (direct proof). □

5.5 BVH Collision Query Algorithm

Although the mesh refinement algorithm for SPM is different from the algorithm for CDPM, they output the same set of data. In one animation frame, the SPM refinement algorithm outputs the faces refined during this frame, which include a set of new faces, a set of merged faces, and a set of deformed faces. The BVH collision query algorithm for locally refined meshes is the same as the AB-tree algorithm for globally refined meshes introduced in section 3.3. The AB-tree algorithm has the characteristic that it receives all the refined faces in the current frame no matter if the meshes are refined globally or locally.

5.6 Framework for Locally Refined Collision Detection in DVEs

In this section, we describe a framework for locally refined collision detection for distributed virtual environments. Then we introduce an efficient refinement criterion based on relative configurations of the objects in space.

5.6.1 Collision Detection Framework for DVEs

Our framework allows fast and exact interference detection that adapts to local surface mesh refinement and progressive transmission in distributed virtual environments. The framework consists of two parts: server and client. The server is responsible for transmitting base meshes, collecting and transmitting legal vertex split records for local refinement upon requests from the clients. The client is responsible for determining when and where to do local refinement based on a region selection function, sending refinement parameters (indices of certain regions in space that are occupied by the meshes) to the server, receiving vertex split records from the server, performing refinement, building and refitting BVHs of meshes, and performing collision queries. The CD algorithm at a client has two phases, a preprocessing phase and a runtime phase. In the preprocessing phase, the volume of transmitted mesh data is relatively small compared to that of the entire mesh. The structures of AB-Trees are encoded in the SPMs which saves the time for BVH construction. Therefore, the cost of running the first phase is negligible. In the runtime phase, the algorithm estimates the time it spends per frame on collision detection, which is determined by the application's performance goals and the set of activities it performs at each frame. Initially AB-Trees are built on the coarsest meshes. Then some mesh primitives are locally refined to higher resolution, based on the configuration of the objects in space. Then, the AB-Tree BVHs of the models are refitted. Finally, pair wise collision queries are performed on the models.

Procedure **PreprocessingPhase**

1 Subscribe to the coarsest meshes from the host

2 Load the received mesh models into the vertex hierarchy

3 Build the AB-Tree BVHs on the vertex hierarchy

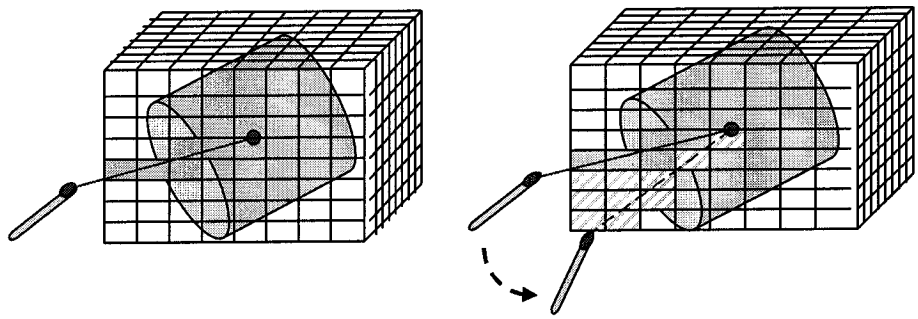


Figure 5.11 The Interaction between a Probe and a Cylinder and the Local Refinement of the Cylinder Polygonal Mesh. Top: 2D; Bottom: 3D.

Procedure **RuntimePhase**

1 **FOR** each animation frame

2 Apply collision region prediction to the current meshes

3 Load the mesh data from local cache

4 Subscribe and receive the mesh data from the host

5 Update the vertex hierarchy and refine the meshes

6 Perform AB-Tree updating and collision query

7 Perform pair-wise interference detection

8 ENDFOR

5.6.2 Collision Region Prediction

Objects in a scene are fitted by bounding boxes. The 3D space of each bounding box is partitioned evenly into smaller indexed boxes. A refinement criterion is required to adapt mesh refinement when relative configurations among objects are changed. As a result, more and more regions indices are reported at runtime, and more parts of the meshes are refined. Figure 5.11 shows a refinement process of an object.

Temporal and spatial coherence: Frames in an interactive viewing session typically exhibit only incremental shifts in contact local neighbor, so the number of potential contact regions

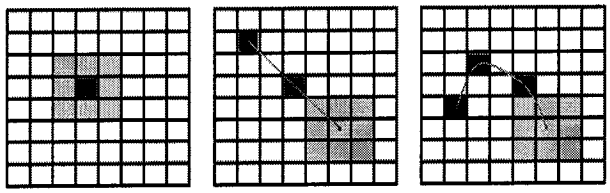


Figure 5.12 Contact Region Prediction. Left: local neighbors of contact region Middle: linear extrapolation Right: polynomial extrapolation.

remains roughly small and constant. Linear and quadratic extrapolation is considered to be at the heart of the best techniques for spatial motion prediction which requires the recording of the contact regions in previous frames. A simpler solution is to take the local neighbors on contact regions in the current frame as the contact regions for the next frame (Figure 5.12). The red areas are the contact region in the current frame. The black areas are contact regions in previous frames. The grey areas are predicted for next frame.

5.6.3 SPM Subscription Protocol

The proposed collision detection approach is performed on adaptively refined meshes. The

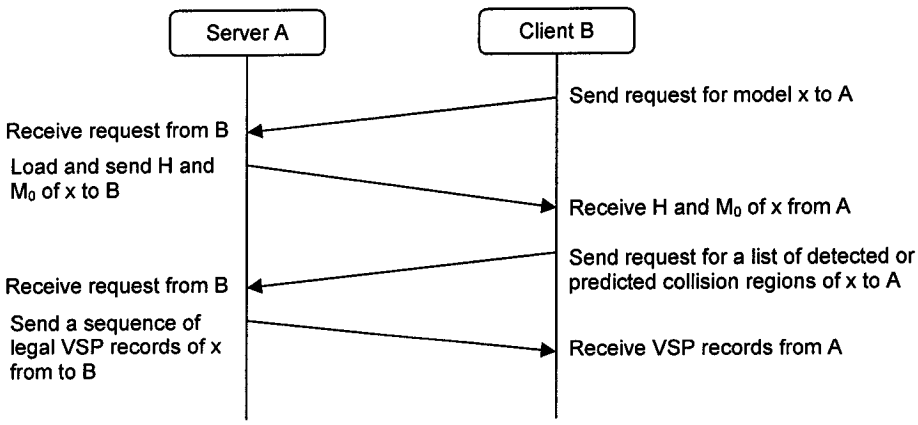


Figure 5.13 SPM Subscription Protocol

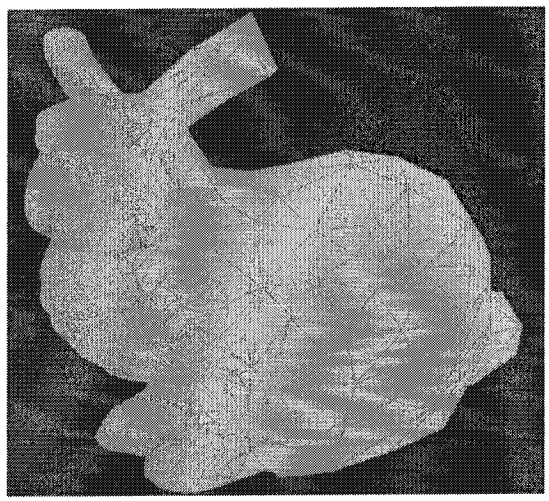


Figure 5.14 A Screenshot SPM Local Refinement Model

meshes are refined at run time only at those parts that are most likely to collide with other objects. There is a communication process between a client and a server. The client tells the server which part of the meshes are to be refined. The server sends a sequence of VSP records to the client. The client then refines the appropriate parts of the meshes by performing a sequence of VSP transformations based on the received data. This process is illustrated in the following diagram. In implementation, the protocol is run over TCP/IP due to its favorable reliability in data transmission.

5.7 Framework for Locally Refined Collision Detection in Haptic Interaction

In this section, we introduce the multi-machine model for haptic interaction. Then we propose a framework for locally refined collision detection for haptic interaction in the multi-machine environments. Then we discuss the resource management issues of the model and present an efficient refinement criterion based on relative configurations of the objects and probes in space.

5.7.1 Collision Detection Framework for Multi-machine Model

The goal of this research work is to design a CD framework based on such a multi-machine solution. The proposed framework is composed of two parts: the VE station and the HRTC. The HRTC is responsible for mesh refinement, collision prediction, and mesh subscription from the VE station, along with cache management, and collision detection, as shown in Figure 5.15. The host is responsible for handling the subscriptions from the HRTC, loading meshes into host memory, and progressive transmission of the data. The proposed framework mainly addresses the issues of how to perform real-time collision detection on complex models using limited memory space.

As mentioned in section 1.2.1, the runtime performance of a CD algorithm is directly affected by the complexity of the input models. We use the local refinement algorithm on SPMs to reduce the combinatorial complexity of the input models. The selection of refinement regions on the mesh is dependent on the local neighborhood of potential collision. The CD algorithm in the HRTC has two phases, the preprocessing phase and the runtime phase which are similar to the phases introduced in section 5.5.1.

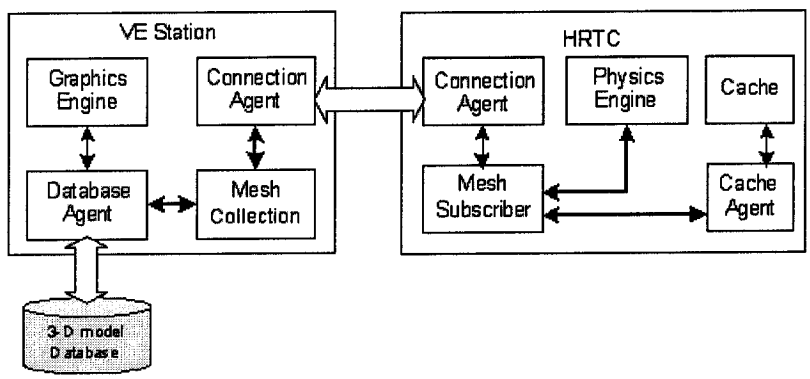


Figure 5.15 CD Framework for Multi-machine Model for Haptic Interaction

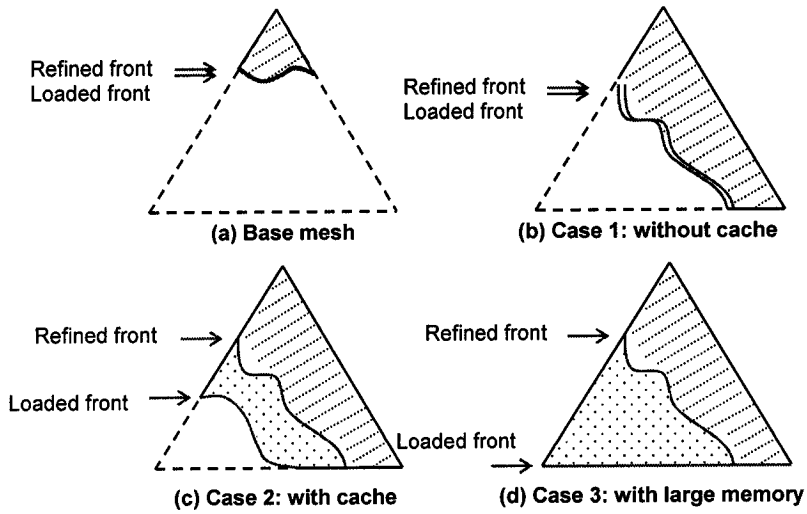


Figure 5.16 Dynamic Construction of Vertex Hierarchy

In the preprocessing phase, the volume of transmitted mesh data is relatively small to that of the whole mesh. The structures of AB-Trees are encoded in the SPMs which saves the time for BVH construction. Therefore, the cost of running the first phase is negligible. In the runtime phase, the algorithm estimates the time it can spend per frame on collision detection, which is determined by the application's performance goals and the set of activities it performs at each frame. Initially AB-Trees are built on the coarsest meshes. Then some mesh primitives are locally refined to lower or higher resolution, based on the configuration of the objects in space. Then, the AB-Tree BVHs of the models are refitted. Finally, pair wise collision queries are performed on the models. The experimental results introduced in section 5.7 demonstrate significant performance improvement over existing algorithms for static LOD meshes.

5.7.2 Mesh Refinement and Resource Management

At runtime, the vertex hierarchy for local mesh refinement is created for each loaded model during the preprocessing stage. Every node of the hierarchy stores the vertex split

information. The hierarchical data structure records the history of vertex splits and edge collapses of a mesh at an instant resolution, which enables fast mesh split and merge operations. As shown in Figure 5.16, the vertex hierarchy maintains two boundaries, *refined front* and *loaded front*. The refined front defines a sub vertex hierarchy in which the vertex split operations stored in each node are performed. The loaded front defines another sub vertex hierarchy which comprises a set of vertex split nodes that has been subscribed and loaded in local cache. In preprocessing phase, the two boundaries are overlapped (Figure 5.16a). In the runtime phase, the two boundaries continue to overlap when free cache space is not available (Figure 5.16b). The space between the two boundaries increases when more cache space is available or collisions regions on the meshes are shifted (Figure 5.16c). Ideally, the cache space is comparable with host memory. In such cases, the mesh data are not necessarily subscribed for more than once (Figure 5.16d). When the cache is small, the pre-fetching of the data from host memory to the cache is more frequent which obviously wastes host memory and increases waiting time at the HRTC. By utilizing temporal and spatial coherence, introduced in section 5.5.2, and simple motion prediction techniques, the pre-fetching of mesh data can be significantly reduced even when the cache space is relatively small.

Distance query: Calculating the distance between probe points and the labeled regions of models is required to accelerate collision detection. Mesh data contained in the regions which are closer than a predefined threshold to probe points are pre-fetched. This prediction can be applied when no contact region is detected in the current frame.

Two-phase local refinement: A frame-to-frame local refinement method for SPMs is illustrated in Figure 5.17. Let A and B be two sets of regions. In the regions of A, mesh primitives are refined to full resolution in frame 1. In the regions of B, mesh primitives are refined to full resolution in frame 2. A two-phase (mesh split and mesh merge) operation is required to refine the mesh according to changing contact configuration. In phase one, the legal edge collapse operations clustered in the regions A/B are collected from the vertex hierarchy. Then operations are performed in order on the mesh. In phase two, the legal vertex split operations clustered in the region B/A are collected from the hierarchy. Then the operations are performed in order on the mesh.

5.8 Implementation and Performance Study

Several models have been tested on the implemented local refinement algorithm. The configuration of the models is given in Table 4.1. Some screenshots are given in Figure 5.14. The output refined meshes in full resolution show that the meshes can be reconstructed without changing the topology and the geometry of the original models, which is consistent with the analytic resolution discussed in section 5.3. In the experiment, several sequences of VSP transformations are performed based on different contact points. The pair wise adjacent faces of each vertex split operation are recorded. The comparison of the pair wise adjacent

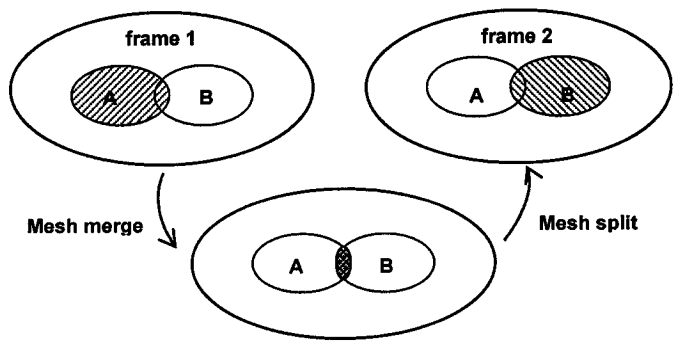


Figure 5.17 Two-phase Local Refinement

faces among difference sequences shows that theorem 3 and theorem 4 introduced in section 5.3 have held for the experiment.

Collision Detection for DVE

The performance study of collision detection in distributed environments focuses on data transmission, frame rate, accuracy, and local refinement cost at the server.

The work load at the server mainly includes:

- Traversing the list of VSP has a cost which is linearly proportional to the size of VSP sequence of the model;
- Selectively collecting some records whose cost is proportional to the number of regions to be refined and the volume of primitives inside the regions;
- Reformatting and sending those records for transmission, the cost of which is proportional to the number of VSP records collected.

The performance of data transmission is affected by the number of VSP records transmitted per frame and network bandwidth. At this point, the performance is affected by the same factors that affect globally refined meshes discussed in section 4.3.2. The accuracy and the frame rate of collision detection are comparable to that of the same models in static full resolution when collision regions on the large scale models are not shifting swiftly. In the worst case, when network transmission is down, the accuracy is not lower than the accuracy of collision detection on the coarsest meshes.

Collision Detection for Haptic Interaction

We have successfully applied our approach to interference detection between cylinder probe model which is represented by static LOD mesh and the manifold SPM benchmark models given in Table 4.1. Screen shots of collision detection on three of the models are shown in

Figure 5.18. The demonstrations have been run on dual Pentium4 2.8GHz processor PCs with 510MB of RAM and Windows XP OS to simulate a multi-machine haptic environment. The real-time graphic rendering is achieved with NVIDIA® GeForce™ FX5200 Graphics Cards. Our implementation uses C++ and an OpenGL library for physics simulation and graphics rendering.

In the HRTC, upon receiving mesh refinement data, time for collision detection can be expressed as

$$T = T_R + T_B + T_Q + T_L$$

where T_R represents the time for mesh refinement, T_B represents the time for BVH refitting, and T_Q represents the time for collision queries on the BVH. T_L represents the time for mesh loading. Assuming that collision query frame rate is fixed, T_L is proportional to the number of vertex split records loaded per frame. T_L increases when the cache space is decreased. The time for mesh refinement, T_R, is given in Figure 5.19. The time for collision query, T_Q, between the probe and the full mesh, locally refined SPM, and coarse mesh of the benchmark models are illustrated in Figure 5.20.. When contact location does not change drastically, the time cost for mesh refinement is near a small constant. With this assumption, T_B is observed to increase in the order of the logarithm of the size of the full meshes. This observation is consistent with the proved theorem 2 introduced in section 3.3.3. In terms of T_Q, we find out that the time for collision queries in a coarse mesh is 0.09% to 37% less than the time cost in a locally high resolution mesh. The variance of the ratio is caused by the difference of complexity of the coarse meshes relative to the refined meshes. Whereas, the cost for the finest resolution mesh is 1.2 to 2.5 times more than the cost for refined meshes. The variance of the ratio is caused by the difference of complexity on locally refined mesh

regions. In terms of memory usage, only the refined regional mesh data are kept in cache. Mesh reloading takes place once in a few seconds. This memory saving strategy makes complex models easy to handle in haptic applications.

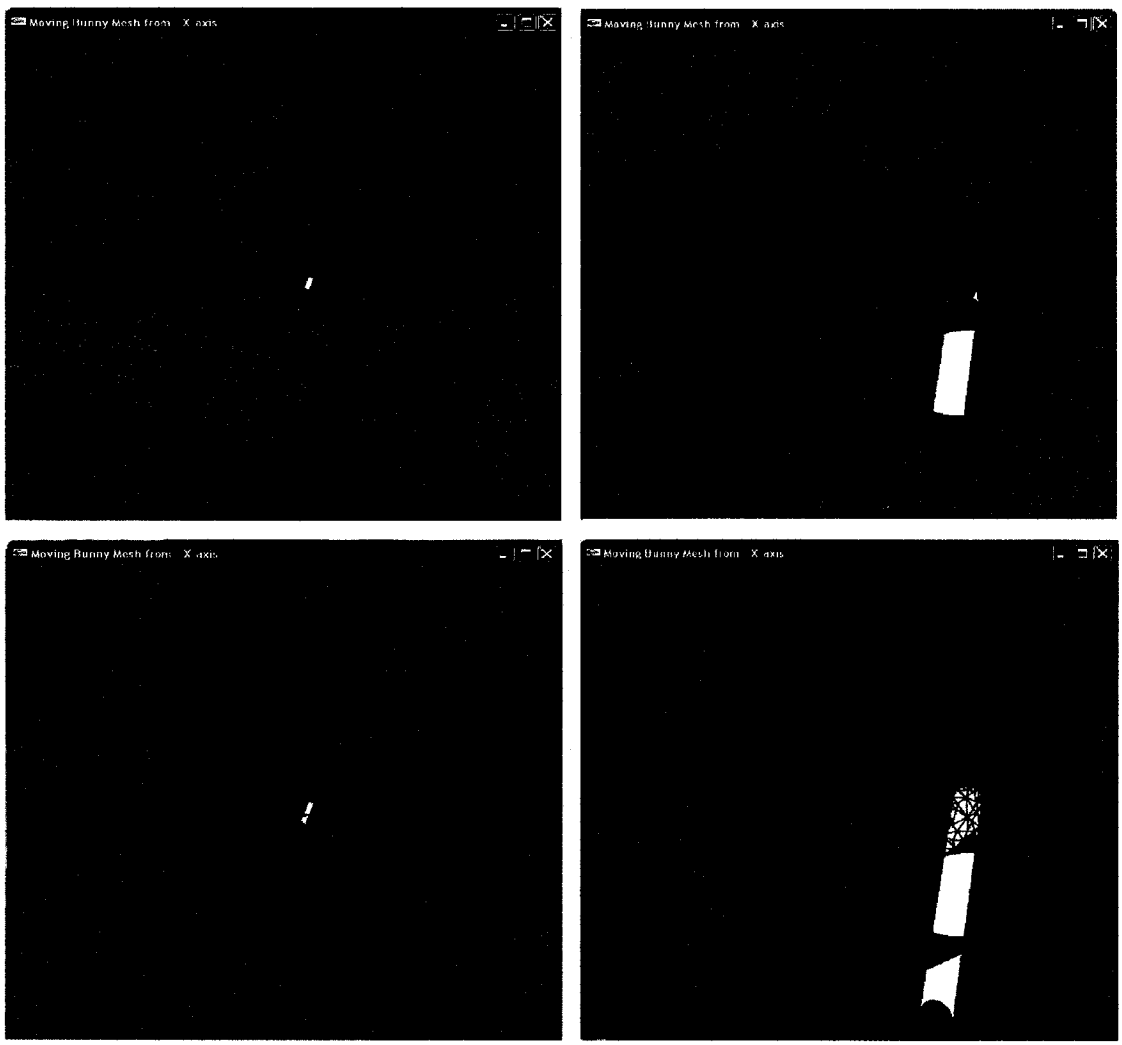


Figure 5.18a Far and Close Views of Locally Refined Meshes

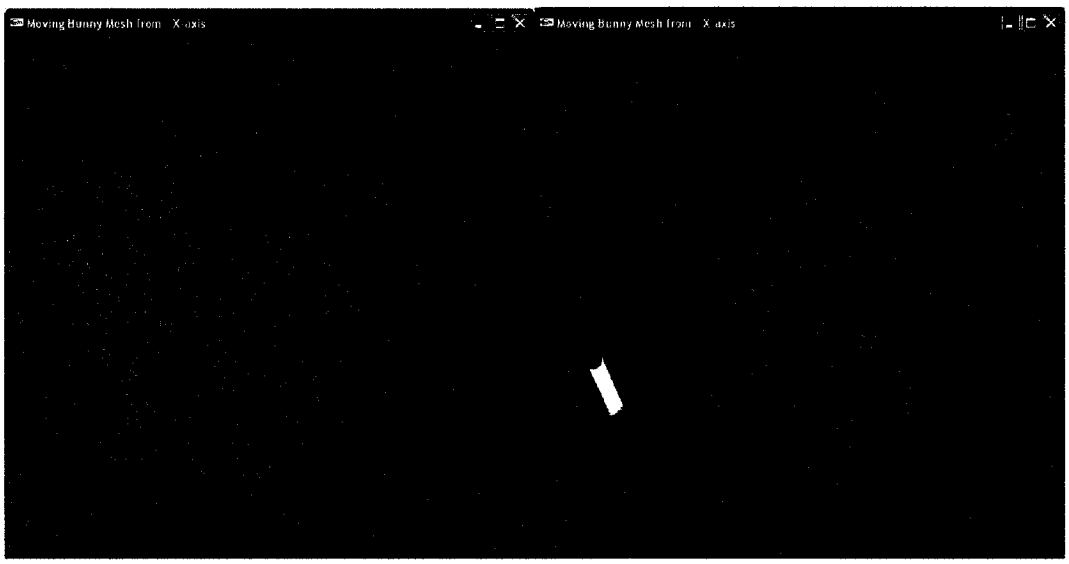


Figure 5.18b Locally Refined SPM. Left: contact region prediction; right: interference detection.

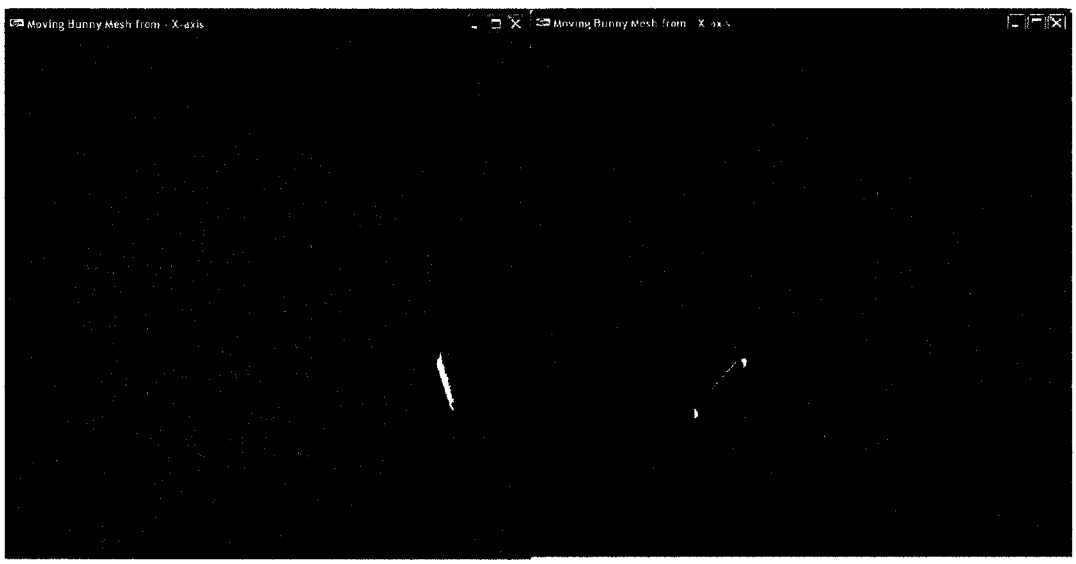


Figure 5.18c Frame-to-frame Refinement

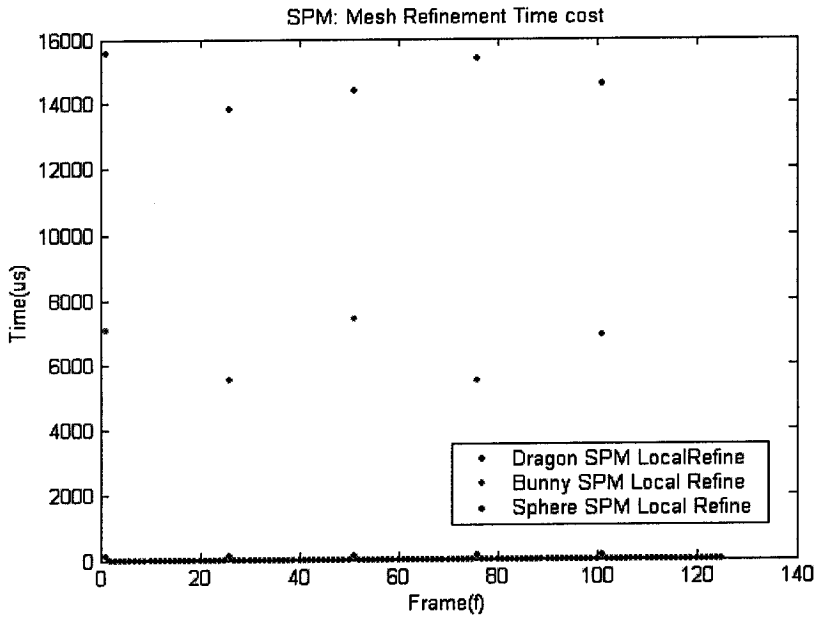
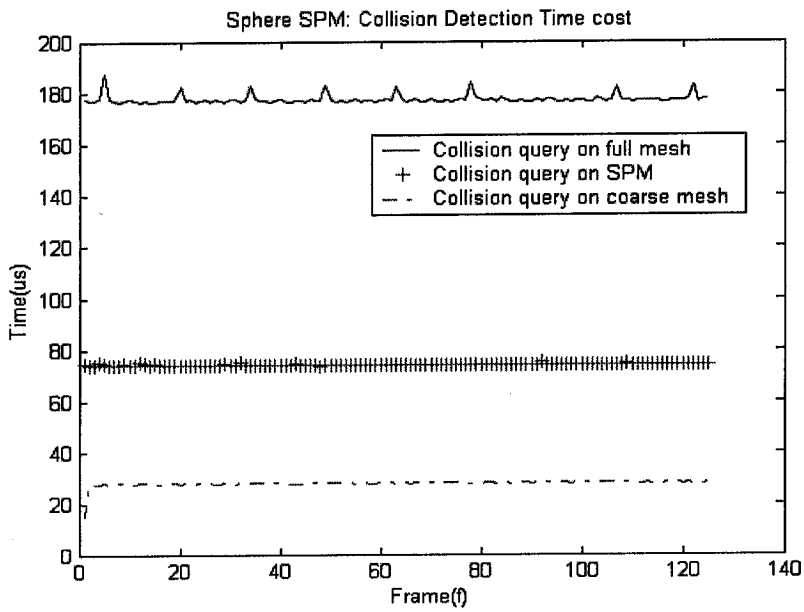
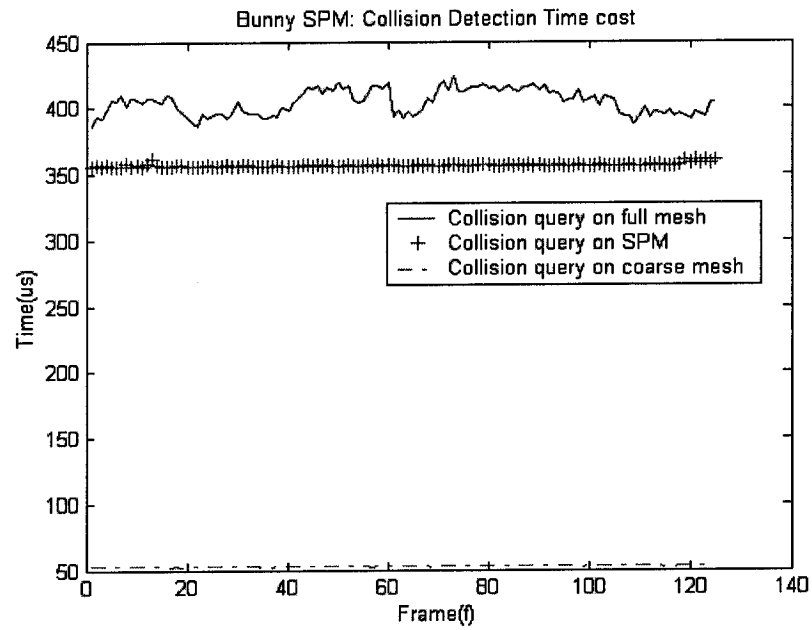


Figure 5.19 Time for mesh refinement on SPM models

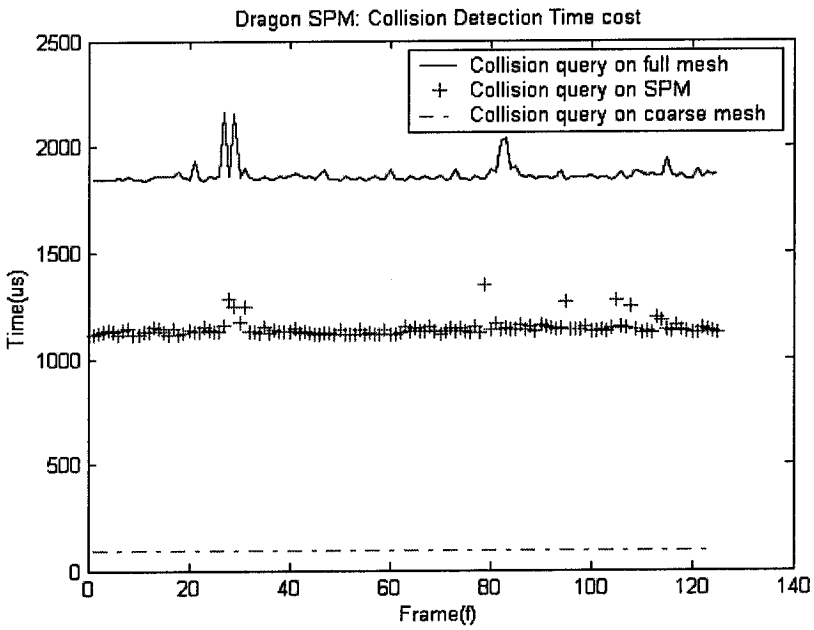


(a) Sphere model

Figure 5.20 Time for Collision Query between a Probe and a SPM Model



(b) Bunny model



(c) Dragon model

Figure 5.20 Time for Collision Query between a Probe and a SPM Model

Chapter 6 Conclusions

This chapter summarizes the main contributions of this thesis and describes avenues for future work in the area of real-time collision detection in virtual environments.

6.1 Summary

Since 1995, only a few VR systems have been commercially available and a few more in the academic domain. This is because there are still some persistent problems in the areas of electro-magnetic tracking, high-level specification of virtual environments, efficient interaction metaphors and frameworks, and real-time physically-based simulations. This thesis has made contributions to the last area.

Multi-resolution collision detection

This thesis presents a new *multi-resolution collision detection* approach which is based on a novel BVH CD algorithm ("AB-tree"). It provides a real-time CD by measuring the available CPU time, free cacheable memory, and memory access pattern at run time. The time and space costs for the CD algorithm are adjusted dynamically accordingly. Thus, real-time performance is guaranteed with some loss of accuracy on selected visual regions of the less important objects in a large scale VE. The proposed solution targets time-critical applications where real time interaction is more important than accuracy in physically-based simulation.

Framework for collision detection in DVEs

Since the Internet boom at the end of 1990s', VR application users require more and more interactions with not just the computer generated intelligence but also real people at the other

end of the network. This growing requirement accelerates the creation of many new applications, such as multiplayer online games, collaborative design, and remote training etc. Many research works have been done to improve real-time rendering, interactions, and simulations for applications running on a local host. However, how to guarantee the real-time performance in a networked environment with unreliable connection and non-uniform bandwidth is still an open problem. The framework presented in this thesis targets the network related issues.

In order to minimize the rate of suspension at the clients due to unpredictable network connection failure, a multi-resolution mesh representation CDPM is introduced. It solves the problem of traditional static LOD collision detection in distributed environments by streaming geometric data over the network from the lowest LOD to the highest LOD, and progressively increasing the accuracy of collision detection at runtime. Thus, the CD at the clients can start once the models in the lowest LOD are received instead of waiting until the entire scene is received. Afterwards, CD is performed at the clients on the data they received and the process never suspends due to network disconnection. Some clients in the system may have low bandwidth connection. In order to gain the same interactive rate as other clients, they can do CD on low LOD models. The accuracy of CD can be increased when more and more data are received. From the users' point of view the application is started instantly, however the realism of the simulation is enhanced progressively.

The multi-resolution collision detection, the CDPM multi-resolution mesh representation, and the framework for collision detection in DVEs presented in this thesis have been successfully integrated. A prototype application has been developed in a distributed setting to demonstrate the feasibility of the framework. The performance is studied both analytically

and experimentally. The proposed approach can significantly improve existing collision detection methods in distributed virtual environments especially in those that are connected by low bandwidth networks.

Locally refined collision detection for large scale models in DVEs

Furthermore, the multi-resolution CD approach is extended to also apply to applications involving large scale models in DVEs, where exact and real-time interference detection is required. Transmitting a large scale model with millions of polygons, such as the Grand Canyon model, is time consuming in comparison with transmitting simple models. Even if the model is transmitted in progressive manner, the earlier received model is in low LOD because the model is refined globally. Increasing the accuracy of CD at the client still takes time because increasing the LOD of the model is a slow process. A new approach, called *locally refined collision detection* is presented in this thesis to deal with this problem. The idea is to selectively refine the model stored in the server at certain areas that are predicted to be most likely to collide with other objects in the near future and then transmit the geometric data of the refined parts to the clients. In this way, the accuracy of CD is increased quickly at the predicted contact areas. The new approach uses the AB-tree CD algorithm and a new SPM representation which is derived from the globally refined CDPM.

Framework for collision detection in multi-machine haptic interaction

A framework for collision detection in multi-machine haptic interaction is proposed in this thesis. The multi-machine model is a centralized system. The VE machine and the haptic controller are connected by a reliable high-speed Ethernet. The major limitation of the physical environment is low CPU power and small cache memory on the haptic controller. Since CD task is typically resided in the haptic controller, the physical limitations keep the

performance from reaching the required high interactive rate. The proposed framework improves the performance by minimizing storage space of complex models used in CD and reducing the complexity of the CD algorithm. This solution tackles the problem by a divide-and-conquer approach based on locally refined collision detection. The idea is to partition large models into separate regions and selectively performs collision queries on them. In this way the required storage space is determined by the complexity of the predicted contact areas. The heavy computing task of CD is divided into many subtasks. Mesh refinement, data re-fetching, and collision prediction tasks reside in the VE machine. The BVH update and collision query tasks reside in the haptic controller. From the end users' point of view, they can start running a haptic application without knowing the entire geometric environment. A realistic interactive force display is achieved smoothly and instantly. The performance of the proposed CD algorithm degrades slightly when the object moves rapidly. We believe that using a locally refined CD is a fresh starting point for future work on multi-machine haptic interaction. However, our current implementation is limited to complex polygonal models without rapid motion. Using intelligent motion prediction to further reduce the re-fetching rate and the volume of data for transmission is a promising topic to be further studied.

6.2 Future Directions

Distributed virtual reality is one of the most promising areas in computing industry. Although it has matured to the point that it can be used commercially, there are still a lot of things that need improvement; such as graphics scene representation, data transmission protocol, and synchronization mechanism of physically-based behavior among end systems. Another important but not yet mature feature is force-feedback. In some special simulations

such as virtual surgery, virtual painting, or virtual sculpture, acoustic and visual feedback turned out to be not sufficient to meet the demands of the users. Force feedback would add significantly the degree of immersion and usability, and it would give a natural and expected cue how to resolve collisions. However, the requirements on collision detection in a force feedback system are very demanding: under all circumstances, query times must be less than 2 milliseconds for checking at least one object against the entire environment [Zac00]. Another important feature that still requires considerable research is interaction with articulated and deformable objects, and the physically-based modeling for these objects.

Since collision detection is an enabling technology not only for virtual reality application, but also for CAD designs, robotics, and tele-operations etc., this area will be scientifically active for many years to come. There are algorithms and object representations that allow faster collision detection queries [JTT01, LG98] than the algorithms presented in this thesis. However, they have other drawbacks, such as limited accuracy, limited model representations, and critical requirements for stable and predictable computing environments. Therefore, there is still a need for more robust algorithms in practice. In particular, three main directions need greater attention: collision detection of deformable objects, collision detection for polygonal model in arbitrary topology, and application-specific algorithms, such as collision detection for haptics. I believe that collision detection algorithms are suitable for implementation in graphics hardware which include GPU and frame buffer. Some recent works can be found in [GLM05, GRL+03]. Guick-CULLIDE, for example, is a fast algorithm for collision culling and detection among polygon-soup models using graphics hardware [GLM05]. The algorithm is efficient for detecting collision among large number of simple object and deformable models. However, the computation of all contacts is limited to

image-space precision. The GPUs used by graphics devices on the market are less powerful than the CPUs of the regular PCs. In virtual reality applications, the work load for graphics rendering is very burdensome in that the resources provided by the dedicated graphics hardware are exhausted. Collision detection by itself is very costly. Using GPU time and limited memory on graphics device will slowdown both graphics rendering and physics simulation.

References

[Ang97] E. Angel. *Interactive Computer Graphics: a top-down approach with OpenGL 3rd ed.* Reading, Mass.: Addison-Wesley, 1997. ISBN: 0201773430.

[AS94] S.S. Abi-Ezzi, and S. Subramaniam. *Fast Dynamic Tessellation of Trimmed NURBS Surfaces.* Computer Graphics Forum, 13, 3, pp. 107–126, 1994.

[Bar92] D. Baraff. *Dynamic Simulation of Non-penetrating Rigid Body Simulation.* Ph.D. thesis, Cornell Univ., 1992.

[BCP03] R. Brown, L. Cooper, and B. Phiam. *Visual Attention-based Polygon Level of Detail Management.* In Proc. the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia, Melbourne, Australia, 2003, pp. 55-62.

[BD92] C. Bajaj and T. Dey. *Convex Decomposition of Poyhedra and Robustness.* Siam J. Comput. 21, 2, pp.339-364, Apr., 1992.

[BEG+04] J. Basch, J. Erickson, L. Guibas, J. Hershberger, L. Zhang. *Kinetic Collision Detection Between Two Simple Polygons.* Computational Geometry: Theory and Applications, vol. 27, no. 3. 2004, pp. 211-235.

[Ber97] G. van den Bergen. *Efficient Collision Detection of Complex Deformable Models using AABB Trees.* Journal of Graphics Tools, 2(4), 1997, pp.1-13.

[BFA02] R. Bridson, R.P. Fedkiw, and J. Anderson. *Robust Treatment of Collisions, Contact, and Friction for Cloth Animation.* ACM Transactions on Graphics, Vol. 21. No.3, pp. 594-603, 2002.

[BHT97] R. Boulic, Z. Huang, D. Thalmann. *A Comparison of Design Strategies for 3D Human Motions.* Advanced Interface for the Information Society, (K. Varghese and S. Pfleger, Eds), Springer Verlag Heidelberg, 1997, pp.306-319.

[Bir85] K. Birman. *Replication and Fault-tolerance in the isis System.* In Proc. the 12th ACM Symposium on Operating Systems, 1985, pp. 79–86.

[BO04] G. Bradshaw and C. O'Sullivan. *Adaptive Medial-Axis Approximation for Sphere-Tree Construction.* ACM Transactions on Graphics, Vol 23, No. 1, pp 1-26, 2004.

[Bro99] F. P. Brooks, Jr. *What's Real About Virtual Reality?* IEEE Computer Graphics & Applications, 19(6):16–27, 1999.

[Cam90] S.Cameron. *Collision Detection by Four-dimensional Intersection Testing.* In Proc. Internat. Conf. Robot.Autom., pp. 291-302, 1990.

[Cam97] S.A. Cameron. *Enhancing GJK: Computing Minimum and Penetration Distance Between Convex Polyhedra.* In Proc. Internat. Conf. Robot. Autom., pp. 3112-3117, 1997.

[Cam86] S.A. Cameron and R.K. Cully. *Determining the Minimum and Penetration distances Between Convex Poyhedra.* In Proc. IEEE International Conference on Robotics and Automation. San Francisco (CA), no.1, pp.591-596, 1986.

[Can86] J. F. Canny. *Collision Detection for Moving Polyhedra.* IEEE Trans. Patt. Anal. Mach. Intell. 8, 2, pp. 200-209. Mar. 1986.

[CD97] M. Cani-Gascuel and M. Desbrun. *Animation of Deformable Models Using Implicit Surfaces.* IEEE Transaction on Visualization and Computer Graphics, vol. 3, no. 1, March 1997, pp. 39-50.

[CH93] C. Carlsson O. Hagsand. *DIVE: a Multi User Virtual Reality System.* In Proc. IEEE Virtual Reality Annual International Symposium, VRAIS '93, pp. 394- 400. Piscataway, NJ: IEEE Service Center, 1993.

[CLL+03]J. Chim, R.W.H. Lau, H.V. Leong, and A. Si. *Cyber Walk: A Web-based Distributed Virtual Walkthrough Environment*, IEEE Transactions on Multimedia, Vol.5, No.4, pp.503-515, Dec. 2003.

[CLM+95] J.Cohen, M.Lin, D.Manocha, and M. Ponamgi. *I-Collide: An Interactive and Exact Collision Detection System for Large-scale Environments.* In Proc. ACM Interactive 3D Graphics Conf., pp. 189-196, 1995.

[CLR99] D. Cohen-Or, D. Levin and O. Remez, *Progressive Compression of Arbitrary Triangular Meshes.* In Proc. Visualization'99. pp. 67-72.October 1999, San Francisco, California.

[CLR+01] T.H.Cormen, C.E.Leiserson, R.L.Rivest, and C.Stein, *Introduction to Algorithms.* MIT Press, 2nd edition, Sept., 2001.

[CPS95] P. Cignoni, E. Puppo, and R. Scopigno. *Representation and Visualization of Terrain Surfaces at Variable Resolution.* In Proc. Scientific Visualization'95, R. Scateni, Ed.,World Scientific, pp. 50–68, 1995.

[CW96] K. Chung and W. Wang. *Quick Collision Detection of Polytopes in Virtual Environments.* In Proc. ACM Symposium on Virtual Reality Software Technology, 1996.

[Dee95] M. Deering. *Geometry Compression.* In Proc. SIGGRAPH'95, pp. 13-22. 1995.

[DMP96] L. De Floriani, P. Marzano, and E. Puppo. *Multiresolution Models for Topographic Surface Description.* The Visual Computer, 12, 7, pp. 317–345, 1996.

[Dom90] D. Dobkin and D. Kirkpatrick. *Determining the Separation of Preprocessed Polyhedra – A Unified Approach.* In Lecture Notes in Computer Science (1990), vol. 443 (ICALP-90), pp. 400-413.

[Don87] B.R. Donald. *A Search Algorithm for Motion Planning with Six Degrees of Freedom.* Artificial Intelligence 31, 3, pp.295-353. 1987.

[EDD+95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. *Multiresolution Analysis of Arbitrary Meshes.* In Proc. SIGGRAPH'95, Auguest 1995.

[EDd+96] J.L. Encarnação, F. Dai, A. del Pino, H. Haase, U. Jakob, M. Unbescheiden, and G. Zachmann. Grenzen der Virtualisierung. Talk at Münchner-Kreis KongreßM"unchen, November 1996.

[EL00] S.Ehmann and M.C. Lin. *Accelerated Proximity Queries Between Convex Polyhedra Using Multi-level Voronoi Marching.* In Proc. IEEE/RSJ Internat. Conf. Intell. Robots Sys., pp. 2101-2106, 2000.

[EL01] S. Ehmann and M.C. Lin. *Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition.* Computer Graphics Forum, 20(3), pp. 500-510. 2001.

[ESG04] N. R. El-Far, X. Shen, and N.D. Georganas, *Applying Unison, a Generic Framework for Hapto-Visual Application Development, to an E-Commerce Application,* In Proc. IEEE Workshop on Haptic Audio Visual Environments and their Applications, Ottawa, Canada, October 2004.

[FS93] T. A. Funkhouser, C. H. Séquin, *Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments.* In Proc. ACM SIGGRAPH'93, 1993. pp. 247-253.

[GBT+99] A. Gueziec, F. Bossen, G. Taubin, and C. Silva. *Efficient Compression of Non-Manifold Polygonal Meshes.* IEEE Visualization '99, pp. 73-80, 1999.

[GGT99] A. Gregory, M. Lin, S. Gottschalk and R. Taylor. *H-Collide: A Framework for Fast and Accurate Collision Detection for Haptic Interaction.* In Proc. IEEE Virtual Reality Conference 1999.

[GHA94] A. Garcia-alonso, N. Serrano, and J. Flaquer. *Solving the Collision Detection Problem.* IEEE Computer Graphics and Applications, 14, 3, pp. 36-43, May 1994.

[GH97] M. Garland and Paul S. Heckbert, *Surface Simplification using Quadric Error Metrics.* In Proc. SIGGRAPH '97, 1997, pp. 209-216.

[GLM05] N. Govindaraju, M. Lin, D. Manocha. *Quick-CULLIDE: Efficient Inter- and Intra-Object Collision Culling using Graphics Hardware.* To Appear in IEEE VR 2005.

[GLM96] S. Gottschalk, M. Lin and D. Manocha, *OBB-Tree: A Hierarchical Structure for Rapid Interference Detection.* In Proc. ACM SIGGRAPH'96, pp. 171-180, 1996.

[GJK88] E.G. Gilbert, D.W. Johnson, and S. Keerthi. *A Fast Procedure for Computing the Distance between Complex Objects in Three Dimensional Space.* IEEE J. Robotics Automat. 4(2), pp. 193-203, April 1998.

[GNR+02] L. Guibas, A. Nguyen, D. Russel. *Collision Detection for Deforming Necklaces.* In Proc. ACM SoCG'02, June 2002, pp. 33-42

[GRL+03] N. Govindraju, S. Redon, M. Lin, and D. Manocha. *CULLIDE: Interactive Collision Detection between Complex Models in Large Environments using*

Graphics Hardware. In Proc. ACM SIGGRAPH/Eurographics Workshop Graphics Hardware, pp. 25-32, 2003.

[GS98] S. Gumhold, W. Straßer. *Real Time Compression of Triangle Mesh Connectivity.* SIGGRAPH'98 pp. 133-140. 1998.

[GSG96] M. Gross, O. Staadt, and R. Gatti, *Efficient Triangular Surface Approximations using Wavelets and Quadtree Structures.* IEEE Trans. on Visual and Computer Graphics, 2(2), 1996, pp.130-144.

[HBS99]C. Ho, C. Basdogan, and M.A. Srinivasan, *An Efficient Haptic Rendering Technique for Displaying 3D Polyhedral Objects and Their Surface Details in Virtual Environments (PDF)*, October'99 Vol. 8, No. 5, pp. 477-491, Presence: Teleoperators and Virtual Environments, MIT Press.

[HDD+93] H. Hoppe, T. Derose, T. Duchamp, J. Macdonals, and W. Stuetzle. *Mesh Optimization.* In Proc. SIGGRAPH'93, pp. 19-26, 1993.

[HDL+96]M. Hughes, C. Dimattia, M.C. Lin, and D. Manocha. *Efficient and Accurate Interference Detection for Polynomial Deformation.* In Proc. Computer Animation Conference, 1996.

[Hec00] C. Hecker, *Physics in Computer Games*. Communications of the ACM vol. 43, no. 7, 2000, pp. 34-39.

[HH96] K. Hamada and Y. Hori. *Octree-based Approach to Real-time Collision-free Path Planning for Robot Manipulator*. In ACM96-MIE, pp. 705-710, 1996.

[HLS97] O. Hagsand, R. Lea, and M. Stenius. *Using Spatial Techniques to Decrease Message Passing in A Distributed VR System.* In VRML 97: Second Symposium on the Virtual Reality Modeling Language, February 1997.

[Hub96] P.M. Hubbard, *Approximating Polyhedra with Spheres for Time-critical Collision Detection.* ACM Transactions on Graphics Vol. 15, Issue 3, 1996, pp.179-210.

[Hop93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. *Mesh Optimization.* In Proc. SIGGRAPH'93, 1993, pp. 19-26.

[Hop96] H. Hoppe, *Progressive Meshes.* In Proc. SIGGRAPH'96, pp. 99-108. 1996.

[Hop97] H. Hoppe. *View-dependent Refinement of Progressive Meshes.* In Proc. *ACM* SIGGRAPH' 97, pp.189-198. August 1997.

[JP04] D. James, D. K. Pai. *BD-tree: Output-sensitive Collision Detection for Reduced Deformable Models.* ACM Transactions on Graphics (TOG), vol. 23, no. 3, August 2004, pp. 393-398.

[JTT01] P. Jiménez, F. Thomas, and C. Torras, *Collision Detection: A Survey.* Computers and Graphics, Vol. 25, No. 2, 2001, pp.269-285.

[KAA+03] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann. *Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping.* Computer Graphics Forum, vol. 22, no. 3, 2003.

[KHM+98] J. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. *Efficient Collision Detection using Bounding Volume Hierarchies of k-DOPs.* IEEE Trans. Visualization Comput. Graph., vol. 4, no. 1, pp. 21-37, 1998.

[KML95] S. Kumar, D. Manocha, and A. Lastra. *Interactive Display of Large-scale NURBS Models.* In Proc. Symposium on Interactive 3D Graphics, ACM SIGGRAPH, pp. 51–58, 1995.

[KPL+98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. *Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries.* In Proc. 3rd Internat. Workshop Algorithmic Found. Robot., pp. 122-136, 1998.

[KS04] A. Kirkpatrick and J. Sze, *Operating-System Induced Jitter in Force Display Computations*, 12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS 2004), 27-28 March 2004, Chicago, IL, USA

[LC02] R.W.H. Lau and O. Chan. *A Collision Detection Framework for Deformable Objects*. In Proc. ACM VRST'02, November 2002, pp. 113-120.

[LC91] M.C. Lin and J.F. Canny. *A Fast Algorithm for Incremental Distance Calculation.* In Proc. IEEE Int. Conf. on Robotic and Automation, vol.2, pp. 1008-1014. Sacramental (CA), 1991.

[LE97] D. Luebke and C. Erikson, *View-Dependent Simplification of Arbitrary Polygonal Environments*, In Proc. SIGGRAPH' 97, August 1997, pp. 199-208.

[LDW97] M. Lounsbery, T. DeRose, and J. Warren. *Multiresolution Surfaces of Arbitrary Topological Type*. ACM Transactions on Graphics 16, 1, pp. 34–73, 1997.

[LE97] D. Luebke and C. Erikson, *View-Dependent Simplification of Arbitrary Polygonal Environments*, In Proc. SIGGRAPH' 97, August 1997, pp. 199-208.

[LG98] M. Lin and S. Gottschalk. *Collision Detection between Geometric Models: A Survey.* In Proc. IMA Conference on Mathematics of Surfaces, 1998, pp. 37-56.

[LGL+99] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. *Fast Proximity Queries with Swept Sphere Volumes*. Tech. Rep. TR99-018, Dept. of Comput. Sci., Univ. North Carolina, 1999.

[LGL+00] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, *Distance Queries with Rectangular Swept Sphere Volumes*. In Proc. IEEE Int. Conference on Robotics and Automation, 2000.

[LHN00] D. Luebke, B. Hallen, D. Newfield, and B. Watson. *Perceptually Driven Simplification Using Gaze-directed Rendering*. Tech. Rep. CS-2000-4, University of Virginia, 2000.

[LKR+96] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. *Real-Time Continuous Level of Detail Rendering of Height Fields.* In Proc. SIGGRAPH '96, pp. 109-118, 1996.

[LM01] T. Larrson and T. Möller. *Collision detection for continuously deforming bodies*. In Proc. EACG Conference on Eurographics, 2001.

[LRC+02] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan-Kaufmann Publishers, San Francisco, July 2002.

[LSG+05] P. Liu, X. Shen, N.D. Georganas and G. Roth. *Multi-resolution Modeling and Locally Refined Collision Detection for Haptic Interaction*, In Proc. 3-D Digital Imaging and Modeling, Jun. 2005.

[LSH+02] I. Lotan, F. Schwarzer, D. Halperin, and J. Latombe. *Efficient Maintenance and Self-Collision Testing for Kinematic Chains*. In Proc. ACM SoCG'02, June, 2002, pp.43-52.

[MBF04] N. Molino, Z. Bao, and R. Fedkiw. *A Virtual Node Algorithm for Changing Mesh Topology During Simulation*. ACM Transactions on Graphics (TOG), vol. 23, no.3, August, 2004, pp. 385-392.

[MPT99] W. A. McNeely, K. D. Puterbaugh, J. J. Troy. *Six Degree-of-freedom Haptic Rendering using Voxel Sampling.* In Proc. ACM Conf. SIGGRAPH90, pp. 401-408, 1999.

[NAT90] B. Naylor, J. Amanatides, and W. Thilhault. *Merfing BSP Trees Yield Polyhedral Modeling Results.* In Proc. ACM Conf. SIGGRAPH90, pp. 115-124, 1990.

[OD01] C. O'Sulivan and J.Dingliana, *Collisions and Perception.* ACM Trans. on Graphics, Vol.20, No.3, 2001, pp.151-168.

[OL03] M. A. Otaduy and M. C. Lin, *CLODs: Dual Hierarchies for Multiresolution Collision Detection.* In Proc. Eurographics Symposium on Geometry Processing, Aachen, Germany, 2003, pp. 94-101.

[Ove92] M. Overmars. *Point Location in Fat Subdivisions.* Information Processing Letters 44, pp.261-265. 1992.

[PR00] R. Pajarola and J. Rossignac. *Compressed Progressive Meshes.* Technical Report: GIT-GVU-99-05, GVU Center, Georgia Institute of Technology, January 1999.

[PR97] D.K. Pai and L.M. Reissel. *Haptic Interaction with Multiresolution Image Curves.* Computer and Graphics, 21, pp. 405-411, 1997.

[RB93] J. Rossignac and P. Borrel, *Multi-resolution 3D Approximation for Rendering Complex Scenes.* In Geometric Modeling in Computer Graphics. Springer Verlag, 1993, pp.455-465.

[Red01] M. Reddy. *Perceptually Optimized 3D Graphics.* IEEE Computer Graphics and Applications, vol. 21, no. 5, pp. 68-75, 2001.

[RHD89] A. Rockwood, K. Heaton, and T. Davis. *Real-time Rendering of Trimmed Surfaces.* Computer Graphics, vol. 23, pp. 107–116, 1989.

[Ric02] P. Rick, *Computer Animation; Algorithms and Techniques.* San Francisco, Calif., Morgan Kaufmann Publishers., 2002. ISBN: 1558605797.

[RKL+04] S. Redon, Y. J. Kim, M. C. Lin, and D. Manocha. *Fast Continuous Collision Detection for Articulated Models.* Journal of Computing and Information Science Engineering, 5(2), 2005.

[Ros98] J. Rossignac. *Edgebreaker: Connectivity Compression for Triangle Meshes.* IEEE Transactions on Visualization and Computer Graphics, 5(1), pp. 47-61, 1999.

[Sam89] H.Samet. *Spatical Data Structures: Quadtree, Octrees and Other Hierarchical Methods.* Addision-Wesley, 1989.

[SBN+03] X. Shen, F. Bogsanyi, L. Ni, and N. D. Georganas. *A Heterogeneous Scalable Architecture for Collaborative Haptics Environments.* The 2nd IEEE Workshop on Haptic, Audio and Visual Environments and their Applications -HAVE, September 2003

[Sca90] L.L. Scarlatos. *A Refined Triangulation Hierarchy for Multiple Levels of Terrain Detail.* In Proc. IMAGE V Conference, pp. 115–122, 1990.

[Shi02] P. Shirley, *Fundamentals of Computer Graphics.* Natick, Massachusetts: A K Peters, c2002. ISBN: 1568811241.

[SHO91] Van der Stappen, F. D. Halperin, and M.H.Overmars, *The Complexity of the Free Space for A Robot Moving Amidst Fat Obstacle.* CGTA: Computational Geometry: Theory and Applications, 1993.

[SK98] A. Schilling and R. Klein. *Texture-dependent Refinement for Multi-resolution Models.* In Computer Graphics International, June 1998.

[SSP+94] G. Singh, L. Serra, W. Png, and H. Ng. Bricknet: *A Software Toolkit for Network-based Virtual Worlds*. Presence: Teleoperators and Virtual Environments, vol. 3, no. 1, pp. 19-34, 1994.

[SV00] J. El-Sana and A. Varshiney. *Continuously-adaptive Haptic Rendering*. In Proc. Virtual Environments, pp. 135-144, 2000.

[SZ99] S. Singhal, M. Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley Professional, 1st edition, 1999.

[SZL92] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. *Decimation of Triangle Meshes*. In Proc. of SIGGRAPH'92. pp. 65-70. 1992.

[SZS+04] X. Shen, J. Zhou, A. El Saddik, and N. D. Georganas, *Architecture and Evaluation of Tele-Haptic Environments*, In Proc. 8th IEEE International Symposium on Distributed Simulation and Real Time Applications (IEEE DS-RT 2004), October 2004, Budapest, Hungary

[TB94] D.C. Taylor and W.A. Narrett. *An Algorithm for Continuous Resolution Polygonalizations of A Discrete Surface*. In Proc. Graphics Interface '94, pp. 33–42, 1994.

[TG98] C. Touma and C. Gotsman. *Triangle Mesh Compression*. In Proc. Graphics Interface '98, pp. 26-34, June, 1998.

[TGH+98] G. Taubin, A. Gueziec, W. Horn, and F. Lazarus, *Progressive Forest Split Compression*, In Proc. SIGGRAPH'98, 1998.

[TKH+05] 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, vol. 24, no. 1, pp. 61–81, 2005.

[TR98] G. Taubin and J. Rossignac. *Geometric Compression Through Topological Surgery*. ACM Transactions on Graphics (TOG), vol.17 no.2, pp. 84-115, Apr., 1998.

[Tur90] G. Turk. *Interactive Collision Detection for Molecular Graphics*, University of North Carolina at Chapel Hill, Chapel Hill, NC, 1990.

[VT94] P. Volino and N.M. Thalmann. *Efficient Self-collision Detection on Smoothly Discretized Surface Animations using Geometrical Shape Regularity*. In Eurographics'94 Computer Graphics Forum (Oslo, Norway), vol.13, pp. 155-166, 1994.

[WFL02] K.Ward, S. Fisher, and M.C. Lin. *Simplified Representations for Modeling Hair*. Technical Report #TR02-020, University of North Carolina at Chapel Hill. March, 2002.

[WM00] W. Wong and R. Muntz. *Providing Guaranteed Quality of Service for Interactive Visualization Applications*. In Proc. of ACM SIGMETRICS, June 2000.

[XEV97] J. Xia, J.El-Sana, and A. Varshney. *Adaptive Real-time Level-of-Detail Based Rendering for Polygonal Models*. IEEE Transactions on Visualization and Computer Graphics, vol.3, no.2, pp. 171-183, Jun. 1997.

[XV96] J. Xia and A. Varshney. *Dynamic view-dependent simplification for polygonal models*. In Proc. IEEE Visualization, pp. 327-334, 1996.

[YSL+04] S-E Yoon, B. Salomon, M. Lin, and D. Manocha. *Fast Collision Detection between Massive Models Using Dynamic Simplification*. In Proc. Symposium on Geometry Processing, pp. 136-146, 2004.

[Zac00] G. Zachmann. *Virtual Reality in Assembly Simulation —Collision Detection, Simulation Algorithms, and Interaction Techniques*. Ph.D thesis, Darmstadt University 2000.

[ZOM+93] M.J. Zyda, W.D. Osborne, N. J.G. Mon, and D.R. Pratt. *NPSNET: Real-time Vehicle Collisions, Explosions and Terrain Modifications*. J. Visual. Comput. Animation 4, 1, pp. 13-24. 1993.

[ZSG04] J. Zhou, X. Shen and N. D. Georganas. *Haptic Tele-Surgery Simulation*. In Proc. IEEE Workshop on Haptic Audio Visual Environments and their Applications, Ottawa, Canada, October 2004.

Appendix A: CDPM Mesh Example

```
# 3D model: 5804 faces 2904 vertices 2896 VSPs
# Base Mesh: 10 faces 8 vertices
v 0.0250617 0.0799181 -0.159895
v 0.38019 0.230182 -0.0215354
v 0.108728 -0.155195 0.0126244
v -0.433037 0.231398 0.0454415
v -0.357951 -0.143089 -0.116849
v 0.257309 0.187015 0.0507284
v -0.347619 -0.129812 0.124568
v 0.151632 -0.043319 -0.08824

f 3 2 6
f 5 4 1
f 7 5 3
f 2 1 6
f 2 3 1
f 1 3 5
f 6 1 4
f 3 6 7
f 7 6 4
f 4 5 7

v* 1  -0.123045 -0.00540496 -0.0360497  0.349609 -0.103558 -0.0485731  6 -5 & 3  5  4  6  2  7
v* 3  -0.00977649 -0.00219043 0.0608097  0.0216919 -0.0273744 -0.0655547  5 -6 & 3  6  1  5  3  8
v* 7  0.0545305 0.0568205 0.0304156  -0.138547 -0.0509124 -0.0253313  -5 4 & 1  10  3  8  9
v* 5  -0.165846 -0.0300601 0.0278729  0.0466149 0.0238474 -0.0253722  9 -7 & 5  6  12  13  3  15
2  10
v* 2  -0.0801721 0.0802583 -0.00379633  0.0293389 -0.0584474 -0.0168839  -6 6 & 3  1  5  4
.
.
.
v* 2655  -8.70228e-006 3.06964e-006 2.43261e-006  0.00278232 -0.000983924 -0.000777565
2854 -2850 & 3  2550  5305  5306  4280  2549
```

The QSlim 2.0 mesh simplification package is used to create CDPM representations. Some of the original models used in the experiments (bunny, cow, bones, etc.) are collected from the following web sites.

http://graphics.stanford.edu/data/3Dscanrep/

http://graphics.cs.uiuc.edu/~garland/home.html

Appendix B: SPM Mesh Example

```
# 3D model: 4096 faces 2050 vertices 2023 VSPs
# Base Mesh: 50 faces 27 vertices
b 50 27
$ 4096 2050 2023
l 0 0 0 0 0 0 0 0 0 0 1392 1304 1426 1273 0 0 0 1497 1301 1319 1410 1068 1090 0 0 1416 1414
1078 814 655 533 0 0 1263 1261 803 372 508 705 0 0 1083 632 494 378 378 0 0 0 0 834 505 424
417 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1412 1076 811 980 0 0 0 1418 1313 1181 980 654 667 0 1144 1074 1583
0 0 0 841 1295 808 1416 0 0 0 0 657 1272 529 1065 0 0 0 0 549 983 827 630 0 0 0 0 375 0 0 1278
1051 539 539 417 782 0 0 0 1787 821 368 0 0 0 0 1985 1275 552 369 510 777 0 984 1265 0 0 0 0
526 990 541 0 0 0 0 0 0 1253 365 0 0 0 0 0 0 1408 502 0 0 0 0 0 0 1072 664 0 0 0 0 0 822 1151 994
1067 0 0 0 0 536 987 0 1563 1420 1080 816 642 837 0 0 839 635 644 434 415 544 0 645 634 0 0 0
0 544 1208 763 0 0 0 0 0 0 1347 427 0 0 0 0 0 0 1328 379 0 0 0 0 0 0 1309 548 0 0 0 0 0 0 1354
1196 1334 0 0 0 0 1059 1260 0 1334 1303 1312 1406 1049 665 0 0 1089 833 504 376 376 543 0
531 826 0 0 0 0 367 1145 501 0 0 0 0 0 0 1071 379 0 0 0 0 0 0 1165 414 0 0 0 0 0 0 1308 366 0 0 0
0 0 0 1353 1154 1070 0 0 0 0 1058 1259 0 1185 1321 1311 1407 1048 629 0 0 1982 1276 551 371
509 778 0 985 1266 0 0 0 0 525 991 542 0 0 0 0 0 0 1254 364 0 0 0 0 0 0 1409 503 0 0 0 0 0 0 1073
663 0 0 0 0 0 823 1210 995 1066 0 0 0 0 535 986 0 1562 1421 1081 817 641 836 0 0 0 1413 1077
810 981 0 0 0 1419 1314 1182 981 653 666 0 1143 1075 1582 0 0 0 840 1296 809 1417 0 0 0 0 658
1271 530 1064 0 0 0 0 550 982 828 631 0 0 0 0 374 1150 0 1277 1050 540 540 416 781 0 0 0 1786
820 370 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1391 1305 1427 1274 0 0 0 1496 1302 1320 1411 1069 1091 0 0
1417 1415 1079 815 656 534 0 0 1264 1262 802 373 507 706 0 0 1082 633 493 377 377 0 0 0 0 835
506 425 416 0 0 0 0 0 0 0 0 0 0

v 1 -0.553269 0.232019 0.837
v 2 -0.898841 -0.0697452 0.512596
.
.
.
v 27 -0.366018 -0.8808 0.436344

f 1 1 15 16 1894
f 2 1 2 21 885
.
.
.
f 50 26 20 27 2578

v* 1 22 28 29 22 28 0.113711 0.145084 -0.126079 -0.0768683 -0.370532 0.0709667 0 0 0 0 0 & 51
2252 40 2 0 -14 3 0 & -52 506 19 2 0 -24 2 0
v* 2 11 30 31 11 29 0.00846654 0.115882 0.249605 0.0548369 0.0103818 -0.267578 0 0 0 0 0 & -53
3544 16 2 0 -29 1 0 & 54 3671 36 2 0 -37 3 0
.
.
.
v* 2023 1188 4072 4073 213 2050 0.0178834 0.0103266 0.0291798 -0.0132126 -0.00762641 -
0.0185131 87 0 87 0 581 & 4095 806 1211 2 581 -1212 2 581 & -4096 810 281 1 116 -655 1 303
```