This document is downloaded from DR-NTU (https://dr.ntu.edu.sg)
Nanyang Technological University, Singapore.

Interactive view-dependent rendering over networks

Zheng, Zhi

2007

Zheng, Z. (2007). Interactive view-dependent rendering over networks. Doctoral thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/35746

https://doi.org/10.32657/10356/35746

Downloaded on 05 Mar 2022 11:25:16 SGT

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

a529980

Interactive View-Dependent Rendering over Networks

Zheng Zhi

NANYANG TECHNOLOGICAL UNIVERSITY
08 APR 2008
LIBRARY

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2007

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Statement of Originality

I hereby certify that the content of this thesis is the result of work done by myself and has not been submitted for a higher degree to any other University or Institution.

....

Date Signature

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Abstract

Despite the rapid improvements in graphics hardware and network technology, interactive rendering of large online triangle models remains a big challenge. Rendering acceleration techniques such as Level-of-Detail (LOD) and visibility culling provide promising solutions for reducing the rendering load on the client as well as optimizing model transmission from the server to the client.

In this thesis, we investigate client-server based rendering of two distinct types of 3D scenes. The first type is composed of a single highly detailed object, and the best rendering acceleration and scene transmission strategy is view-dependent LOD. The second type is a large collection of densely distributed objects of small to medium sizes, for which visibility culling technique is the best choice. A common problem faced by the client-server based rendering of both types of scenes is the delay in rendering caused by the network latency. Therefore, we focus our research work on three aspects: client-server based view-dependent LOD, visibility-based scene transmission and overcoming network latency for client-server based rendering.

For online browsing of a single large model, we propose a new view-dependent LOD scheme that is favorable to the client-server architecture with low-capacity clients. The multiresolution hierarchy and the selective mesh are decoupled into two relatively independent units to fit the server-client architecture, and the selective mesh and mesh-updates are managed in an efficient way for low-capacity clients. The algorithm for

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

building the multiresolution hierarchy and the algorithm for performing run-time mesh selection are presented. We also introduce a general traversal management strategy that is beneficial for frame-rate regulation and triangle-budget control. Three different traversal algorithms are compared under this strategy.

For client-server based walkthroughs in scenes with densely distributed objects, we explore visibility-based scene transmission. Data prefetching algorithm based on precomputed from-region visibility information is proposed. This algorithm predicts and prefetches the precomputed Potentially Visible Set (PVS) of neighboring viewcells that will be needed in the near future according to the current view position of the client. We also provide a delta-transmission algorithm to optimize the prefetching transmission procedure.

In order to achieve interactive frame rate on the clients, we have to overcome network latency. We propose a novel solution to this problem in the context of client-server based view-dependent LOD rendering. Our solution is to make the client process and the server process run in parallel within a frame, and use the rendering time on the client to cover the network latency. A view-parameters prediction mechanism is developed to make the parallelism of the client and the server feasible. Our approach dramatically reduces the frame time compared to the previously used sequential method, and can achieve interactive frame rate in the system environment where the sequential method cannot. In our LAN experiment environment, the view-dependent rendering results of our parallel algorithm with predicted view-parameters are very close to those of the sequential algorithm that uses actual view-parameters. We also extend it to WAN environments with relatively long round-trip times. The issue of multi-frame parallelism is addressed and the prediction span can be adjusted dynamically according to the current network condition to improve the rendering quality.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Chan Kai Yun, who has been creating a great work environment at our lab, putting in a lot of time and effort in guiding my work, giving me valuable advice and supporting me in many different ways. I am also very grateful to Professor Edmond Prakash for giving me insightful comments and inspiration on my work, and for his willingness to be my second supervisor after Professor Chan retired.

I would like to express my gratitude to the staff of CAMTech, in particular, Dr. Wolfgang Müller-Wittig and Mr. Marcus Poicke who supported me in an amazing opportunity of overseas research attachment. I want to thank all members of CAMTech, current and former, Ms. Zhu Chao, Mr. Jochen Quick, Mr. Gerrit Voss, Mr. Thomas Elias, and Dr. Wu Zhongke, for the insightful discussions with them. I also express my thanks to all the staff of GameLAB. They have provided me a comfortable and well-equipped work place during the second half of my Ph.D candidature.

I spent three informative and enjoyable months at Fraunhofer Institute for Computer Graphics (IGD), Darmstadt, Germany. I sincerely appreciate the help from Professor Volker Luckas and Dr. Jörg Sahm, who introduced me to a much broader horizon of computer graphics and have been a true source of good advice.

Finally, my deepest thanks to my husband for his unconditional love and support, and my parents and sisters for their endless encouragement.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Contents

Abstract i

Acknowledgements iii

Contents iv

List of Figures ix

List of Tables xiii

1 Introduction 1

1.1 Background 1

1.2 Motivations 4

1.2.1 Client-server based view-dependent LOD 5

1.2.2 Visibility-based scene transmission 7

1.2.3 Overcoming network latency 8

1.3 Contributions 9

1.4 Thesis organization 12

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

2 Literature Review **15**

2.1 Mesh simplification 15

2.1.1 Mesh simplification operators 15

2.1.2 Simplification error metrics 18

2.2 View-dependent LOD 20

2.2.1 Continuous LOD 20

2.2.2 View-dependent error criteria 21

2.2.3 View-dependent LOD schemes 22

2.3 Visibility culling 27

2.3.1 From-point visibility 28

2.3.2 From-region visibility 29

2.4 Rendering over network 30

2.4.1 General client-server rendering systems 31

2.4.2 Client-server rendering with view-dependent LOD 32

2.4.3 Client-server rendering with occlusion culling 35

2.4.4 View-parameters prediction 35

2.5 Summary 37

3 A DAG Hierarchy for View-dependent Multiresolution **39**

3.1 Introduction 39

3.2 Legality condition 41

3.3 Construction of the DAG multiresolution hierarchy 46

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

3.3.1 Non-redundant DAG 47

3.3.2 Mini-redundant DAG 50

3.4 Construction of the visibility attributes 52

3.4.1 Out-of-view 53

3.4.2 Back-facing 54

3.4.3 Screen-space error 57

3.5 Results 58

3.6 Summary 61

4 DAG Hierarchy Traversal and View-dependent Rendering 63

4.1 Introduction 63

4.2 Maintenance of the fronts 65

4.2.1 Representations of the fronts 65

4.2.2 Maintaining the fronts in the DAG hierarchy 67

4.3 Traversal constraints and management 69

4.4 Comparison of three traversal algorithms 72

4.4.1 Priority-oriented traversal 73

4.4.2 Linear traversal 74

4.4.3 Constant-time traversal 75

4.5 Efficient representation of selective mesh 76

4.6 Results 79

4.6.1 Implementation details 79

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

4.6.2 Experimental results 79

4.7 Summary 84

5 Predictive Parallelism for View-dependent Multiresolution 87

5.1 Introduction 87

5.2 Parallelism for client-server view-dependent rendering 89

5.2.1 Overview 89

5.2.2 Synchronization 91

5.2.3 Algorithm details 94

5.3 View-parameters prediction 95

5.4 Results 97

5.4.1 Implementation details 97

5.4.2 Experimental results 98

5.5 Summary 102

6 Predictive Parallelism on WAN 105

6.1 Introduction 105

6.2 Multi-frame parallelism 106

6.3 Dynamic change of the prediction span 111

6.4 Results 112

6.5 Summary 118

7 Visibility-based Scene Transmission 121

7.1 Introduction 121

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

7.2 PVS prefetch 123

7.2.1 Problems with the SNP algorithm 123

7.2.2 The position-based neighbor prefetch algorithm 125

7.3 Delta-transmission 128

7.4 Client-side cache management 131

7.5 Results 133

7.6 Summary 138

8 Conclusion **139**

8.1 Summary 139

8.1.1 View-dependent LOD for client-server systems 140

8.1.2 Predictive parallelism to overcome network latency 141

8.1.3 Prefetching precomputed PVS 142

8.2 Discussion 143

8.2.1 View-dependent rendering for multiple clients 143

8.2.2 Out-of-core simplification and rendering 144

8.2.3 Integrating view-dependent LOD and visibility culling 146

8.2.4 View-dependent LOD with textures 147

Acronyms **149**

Author's Publications **151**

Bibliography **153**

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

List of Figures

1.1 Multiresolution hierarchy and selective mesh on a client-server system. 6

2.1 An example of half-edge-collapse. 17

2.2 An example of mesh fold-over caused by a half-edge-collapse. 18

3.1 Legality definition for vertex-split and edge-collapse in [137, 136, 37]. . 42

3.2 Legality definition for vertex-split and edge-collapse in [63]. 43

3.3 Different surrounding possibilities by Hoppe's legality definition. 44

3.4 Our legality definition for vertex-split and edge-collapse. 45

3.5 The content in a vertex-split / edge-collapse operator. 46

3.6 An example of a list of vertex-splits. 48

3.7 Building the DAG from the list of vertex-splits in Figure 3.6. 49

3.8 Non-redundant edges and redundant edges. 50

3.9 The non-redundant algorithm. 51

3.10 The mini-redundant algorithm. 52

3.11 Out-of-view test. 53

3.12 Merging bounding spheres. 54

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

3.13 An example of the out-of-view simplification. 55

3.14 Back-face test. 56

3.15 Merging cones of normals. 57

3.16 An example of back-face simplification. 57

3.17 An example of screen-space error control. 59

3.18 The data structure for a DAG node. 61

4.1 Incremental traversal on three different hierarchies. 66

4.2 The procedure of view-dependent error evaluation for a front node. . . 67

4.3 The maintenance procedure when a node t splits. 68

4.4 The maintenance procedure when a node t collapses. 68

4.5 An example of maintenance of the front F_c and the front F_s. 69

4.6 Management of the DAG hierarchy traversal. 71

4.7 Priority-oriented traversal algorithm. 74

4.8 Linear traversal algorithm. 75

4.9 Constant-time traversal algorithm. 76

4.10 The format of the selective mesh on the client. 78

4.11 The traversal times of the three traversal algorithms on the Happy model. 83

4.12 The accuracies of the linear traversal and the constant-time traversal on the Happy model. 83

4.13 Rendering results of the 135th frame in the first interaction path. 84

4.14 Rendering results of the 70th frame in the second interaction path. . . . 85

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

5.1 The sequential method for client-server view-dependent rendering. . . . 90

5.2 The one-frame predictive parallel strategy for client-server view-dependent rendering. 92

5.3 An example of a late batch of mesh-updates in one-frame predictive parallelism. 93

5.4 The server process. 94

5.5 The client process with one-frame predictive parallelism. 95

5.6 One-frame view-parameters prediction. 97

5.7 The frame time for the sequential algorithm and the one-frame predictive parallel algorithm on the Dragon model. 100

5.8 The accuracy of the one-frame predictive parallel algorithm on the Dragon model. 100

5.9 Rendering results for the sequential algorithm and the one-frame predictive parallel algorithm. 104

6.1 The multi-frame predictive parallel method in a client-server based view-dependent rendering system with a long round-trip time. 107

6.2 Rendering procedure with multi-frame predictive parallelism. 108

6.3 An example of the rendering procedure when the prediction span is 3 frames. 109

6.4 An example of the synchronization for multi-frame predictive parallelism. 110

6.5 Dynamic change of the prediction span. 113

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

6.6 The accuracies for different RTTs and their appropriate prediction spans for the Happy model. 115

6.7 Rendering results of the 120th frame in the first interaction path for the Happy model. 117

6.8 The accuracies for different prediction spans when RTT is $170ms$ for the Happy model. 118

7.1 SNP algorithm. 124

7.2 PBNP algorithm. 126

7.3 PVS of the viewcells share a lot of local similarity. 130

7.4 Delta-transmission routine for the client and the server. 131

7.5 Cache entry replacement strategy. 132

7.6 The top view of the simulated city model and the two walking routes. . 135

7.7 The number of transmitted objects at each step on the first route. . . . 137

7.8 The number of transmitted objects at each step on the second route. . 137

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

List of Tables

3.1 The models used throughout the thesis. 59

3.2 The time costs for DAG construction. 60

3.3 Comparison of the total numbers of incoming edges in the DAG hierarchies. 60

4.1 Average run-time data with three different traversal algorithms. 81

4.2 The memory footprint of the selective mesh. 81

4.3 The number of triangles and the number of front nodes. 86

5.1 Comparison of the sequential algorithm and the one-frame predictive parallel algorithm on various models. 99

6.1 The frame times and accuracies of the multi-frame predictive parallel algorithm for different RTTs. 114

7.1 Regions and their corresponding prefetching areas. 127

7.2 The number of prefetching transactions and the number of cache swaps for the two algorithms on the two walking routes. 136

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Chapter 1

Introduction

1.1 Background

Computer graphics and 3D geometries have widespread uses in diverse areas such as science, engineering, education, medicine, industry, business, art and entertainment. The main function of a 3D graphics system is to generate a two-dimensional image, given a virtual camera, three-dimensional models, light sources, lighting models, textures and more [92]. The process of creating 2D images from 3D models is often called *rendering* [41]. In an interactive 3D graphics system, the images must not only appear correct, but also be generated on the computer fast enough to respond to the viewer's action, so that the viewer would feel a dynamic process rather than individual images. As reported by Möller and Haines [92], at one frame per second (fps), there is little sense of interactivity; at around $6fps$, a sense of interactivity starts to grow; at $15fps$, the user can focus on action and reaction. A speed of $30fps$ is generally considered interactive, while a speed of $60fps$ is usually expected in action games.

Models may exist in many different representations, such as polygonal meshes, splines, voxels, implicit surfaces, point-based and image-based representations, etc. Each of these representations is of special interest to a certain application or problem

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

domain. Despite the diversity of model representations, polygonal meshes currently dominate interactive 3D graphics systems such as Virtual Reality (VR) environments, industry and military simulations, and video games. This is mainly due to the mathematical simplicity of the polygonal meshes, which has in turn led to widely available polygon rendering accelerators. In fact, most other model representations can be converted with arbitrary accuracy to a polygonal mesh. In particular, triangle meshes are most preferred for their constant memory requirements and guaranteed planarity of triangles [88]. The mesh composed of non-triangle polygons can be triangulated by existing triangulation algorithms [115, 95]. In this thesis, only the representation of triangle mesh is discussed.

A polygonal mesh is only an approximate representation of a smooth surface. In general, the more accurate the approximation, the more polygons are needed to define the model. Consequently the more polygons of the model, the more time and memory will be required to render an image of the model. The advances in 3D laser scanning and geometric modeling technologies have enabled the generation of very large and refined polygonal models. For examples, the UNC Powerplant model [39] is about 13 million polygons and the scanned model of David from the Digital Michelangelo project [79] is about 8 million polygons at a resolution of $2mm$. Although graphics hardware has rapidly evolved over the past decade, rendering such huge models at interactive frame rates is still a great challenge. To satisfy the requirement of higher frame rate and the increasing desire for more realistic and complex 3D scenes, rendering acceleration algorithms are always very important research topics in computer graphics.

Image-Based Rendering (*IBR*), *visibility culling* and *Level-of-Detail* (*LOD*) are the most important rendering acceleration techniques so far. IBR represents one or more objects with an image to provide more efficient rendering of geometrical models. Commonly used IBR techniques include sprites [78], billboards [11], imposters [89] and

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

particle systems [102].

Visibility culling reduces the number of polygons to be processed by removing portions of the scene that are not visible and therefore do not contribute to the final image. Generally speaking, there are three types of visibility culling techniques, among which *back-face culling* and *view-frustum culling* are two simpler culling techniques that eliminate polygons facing away from the viewer and polygons outside the view-frustum. *Occlusion culling* is a more complex culling technique that aims at quickly eliminating objects hidden by groups of other objects. Visibility computation can be performed during rendering with respect to the current viewpoint, and this is called *from-point visibility* [23]. Visibility information can also be computed with respect to a region and this is called *from-region visibility* [23]. From-region visibility information is often computed in a preprocessing stage, where the viewing space is partitioned into viewcells and for each viewcell a *Potentially Visible Set* (*PVS*) is computed and stored, so that the run-time visibility computation is simplified into PVS querying.

The basic idea of LOD is to reduce the polygon load by using simpler representation for small, distant or unimportant portions of the scene. According to Luebke *et al.* [88], the LOD frameworks in the literature can be classified into the following categories. *Discrete LOD* is the traditional approach, where multiple versions of each object are pre-generated, each at a different level of detail. At run-time, one of the versions is chosen to represent the object. *Continuous LOD* creates a data structure encoding a continuous spectrum of details for an object. The desired level of detail is calculated on-the-fly before the object is rendered. Continuous LOD can offer better granularity, smooth LOD transition, and even progressive streaming. *View-dependent LOD*, also called *multiresolution mesh* [31], extends continuous LOD by selecting the most appropriate level of detail for current view according to certain view-dependent criteria. Optimizing triangle distribution for the current view, view-dependent LOD leads to more efficient

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

use of system resources and better fidelity for a given triangle-budget. Another type of LOD scheme is *Hierarchical Level-of-Detail* (HLOD) [90, 39], where multiple smaller objects in a scene can be recursively replaced with a larger single representation.

Networked Virtual Environment (net-VE) has been a very active research area for many years. A large number of net-VE systems exist as described in the book by Singhal and Zyda [118]. Since they have different goals and motivations, they differ largely in network architecture, content transmission and interaction capability. In many application areas in industry, education and entertainment, the client-server paradigm is desirable, where highly detailed models with large number of polygons from industry products, scientific data, city models, artworks or cultural heritage objects can be viewed on multiple clients from a central server. The server has to be powerful enough to support multiple clients and the clients may be low-end computers, or even handheld devices. The client cannot keep the complete original model locally due to the restriction of its resources and capability, or due to copyright reasons, which is especially the case for digitalized artworks and heritage objects such as the Digital Michelangelo project [79, 74] and the virtual Peranakans [119]. The "download-and-play" pattern mostly used today is not practical for these models [122]. To remotely and interactively display such models on relatively low-capacity clients remains a big challenge and is receiving more and more research interests. Rendering acceleration techniques provide promising solutions for both reducing the rendering load on the client and optimizing model transmission from the server to the client.

1.2 Motivations

Complex 3D scenes of a large number of polygons exist with different characteristics. In this thesis, we investigate client-server based rendering of two distinct types of scenes.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

The first type of scene is composed of a single highly detailed object and the scene comes in "polygon soup" [87]. The representatives of this type include some CAD models, scientific or medical data and laser scans of artworks such as the Digital Michelangelo Project [79]. The second type of scene is a large collection of objects of small to medium sizes in terms of polygon numbers, and a typical scene of this type is a city model with densely distributed buildings. For viewing a single large object, the best rendering acceleration and scene transmission strategy is view-dependent LOD. However, for urban walkthrough or architectural walkthrough, where the scene comprises a large number of densely distributed objects, visibility culling technique is the best choice. A common problem faced by the client-server based rendering of both types of scenes is the lags in rendering caused by the network latency. Therefore, we focus our research work on three aspects: client-server based view-dependent LOD, visibility-based scene transmission and overcoming network latency for client-server based rendering.

1.2.1 Client-server based view-dependent LOD

Continuous LOD, especially the concept of the Progressive Mesh (PM) proposed by Hoppe [62], fits very well in the scenario of online browsing of a single large model. The model reaches the client progressively and the client can start viewing the model without waiting for the entire model to be downloaded. View-dependent LOD can offer more benefits for online browsing such as better fidelity and more efficient use of the network bandwidth and the graphics-card bandwidth. However, there are still several obstacles in developing client-server based view-dependent LOD systems.

Generally speaking, view-dependent LOD is realized by a *multiresolution mesh hierarchy*, on which the current view-parameters are used to select the best representation of the model for the current view. Before each frame is rendered, the hierarchy is traversed. The nodes that satisfy certain criteria are split and new vertices and triangles

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

are added to the selective mesh. This is called *refinement*. On the other hand, the nodes that do not satisfy the criteria are collapsed and some vertices and triangles are deleted from the selective mesh. This is called *simplification*.

In a client-server based view-dependent LOD system, the multiresolution hierarchy and the selective mesh should be maintained on the server and the client separately. The view-parameters and the mesh-updates for refinement and simplification are packed in the communication messages between the client and the server (refer to Figure 1.1). This architecture requires the multiresolution hierarchy and the selective mesh to be independent of each other. The traversal of the multiresolution hierarchy should not rely on the current condition of the selective mesh and the rendering of the selective mesh should be performed without accessing the multiresolution hierarchy. However, most existing view-dependent multiresolution schemes have closely coupled structures, which are not naturally adaptable to the client-server environment. In some works by Hoppe [63], Xia *et al.* [136], and El-Sana and Varshney [37], the traversal of the hierarchy is heavily dependent on the adjacency information of the selective mesh because of the necessary run-time precondition checks. In other works by DeFloriani *et al.* [31], To *et al.* [125], and Pajarola and DeCoro [98], the selective mesh is encoded by a subset of the multiresolution hierarchy and the rendering is performed by scanning the hierarchy.

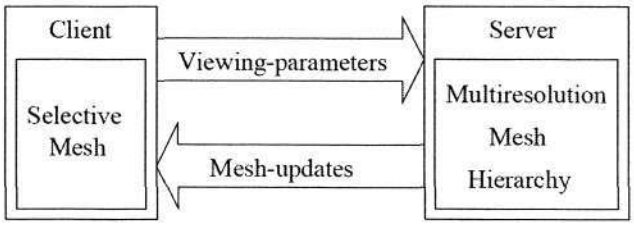


Figure 1.1: The multiresolution hierarchy and the selective mesh on a client-server based system.

For low-capacity clients in the client-server based rendering environments, it is very

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

important that the representation of the selective mesh is memory-efficient and optimal for rendering. From this consideration, the complex adjacency information is better avoided in the selective mesh that aims at interactive rendering. However, in most existing view-dependent multiresolution schemes [63, 31, 125, 71, 98], the modifications made by a mesh-update operator rely on the adjacency information of the current selective mesh, which requires the representation of the selective mesh to accommodate the adjacency information to support mesh modification, resulting in extra memory cost and inefficient rendering on the clients.

Although several client-server based view-dependent LOD frameworks have been proposed in the literature by To *et al.* [125], DeFloriani *et al.* [30], El-Sana *et al.* [35], and Kim *et al.* [72], the problems mentioned above are not well solved. Due to the closely coupled structure of their view-dependent LOD schemes, all these frameworks have to store more or less duplicate data either on the server or on the client, which not only requires extra memory, but also imposes the maintenance work for the duplicate data. The first motivation of this thesis is therefore to present a new client-server based view-dependent LOD scheme with decoupled structure of multiresolution hierarchy and selective mesh.

1.2.2 Visibility-based scene transmission

Although view-dependent LOD techniques are desirable for interactive rendering of large individual models, they are not efficient for urban scenes that include a large collection of small to medium-sized objects. For urban walkthroughs, visibility culling is the key to accelerate the rendering speed, as only a small fraction of the scene is visible from a given viewpoint [94].

From-region visibility algorithms and precomputed PVS have been used for visibility-based selective scene transmission [25, 76, 75]. They are favorable to client-server based

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

urban walkthroughs for two reasons. Firstly, the PVS information for one viewcell is valid for more than one frame and hence alleviates the network communication lag [25]. Secondly, from-region visibility has the predicting capability for prefetching strategies by using the visibility information from adjacent viewcells [23].

Many from-region visibility algorithms have been proposed [23] and any one of them can be used for visibility-based selective transmission. Our concern is to provide a prediction and prefetching strategy for the precomputed PVS, so that the visible objects for rendering can be delivered to the client in advance. Koltun *et al.* [76] have proposed a simple strategy that prefetches the PVS of all the adjacent viewcells of the current viewcell that includes the viewpoint. However, this prediction and prefetching strategy is too conservative and may cause oscillation effect when the client-side memory is restricted. This inspired us to propose an improved prediction and prefetching strategy for PVS transmission.

1.2.3 Overcoming network latency

For any network-based rendering system, network latency is a big obstacle to achieve interactivity. In this thesis, we investigate the problem of overcoming network latency in the context of client-server based view-dependent LOD rendering.

An obvious problem faced by any view-dependent LOD scheme is the relatively expensive run-time process of multiresolution hierarchy traversal before each frame is rendered. This adds a considerable amount of frame time. When it comes to client-server based view-dependent LOD, the network latency introduces an even larger time cost. The network latency consists of transmission time and Round-Trip Time (RTT). The round-trip time is the time elapsed to send a packet to a remote host and receive the response. Depending on the network environment, it can be less than one millisecond in a high-speed Local Area Network (LAN), or can be as long as hundreds

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

of milliseconds in a Wide Area Network (WAN). Overcoming the extra cost caused by server-side multiresolution hierarchy traversal and the network latency is critical to achieve interactive frame rate on the client side.

To reduce the network communication time, client-side cache and compression techniques have been used [30, 35], but these two techniques are not suitable for small-memory clients or networks with long round-trip times. Client-side cache adds heavy memory overhead for clients with scarce memory resource. Compression can reduce the packet transmission time, but cannot solve the type of network latency problem caused mainly by round-trip time. In this thesis, we attempt to solve the problem of network latency in a novel way that is favorable to low-capacity clients and scalable to networks with long round-trip times. Although this part of work is carried out in the context of client-server based view-dependent LOD rendering, it can be generalized to other client-server based rendering systems, such as the ones using visibility-based scene transmission.

1.3 Contributions

The main contributions of this thesis are:

1. We present *a new view-dependent LOD scheme with clearly decoupled structure* of the multiresolution hierarchy and the selective mesh, which is especially favorable to the client-server based system (see Chapter 3). A new legality definition is proposed for edge-based refinement/simplification operators, where the simplification operators are *edge-collapses* (*ecols*), and the refinement operators are *vertex-splits* (*vsplits*) [136, 63]. The legality definition is flexible enough for selective refinement and can also guarantee the independence of the vsplit/ecol operators from the adjacency information of the selective mesh. The multiresolution hierarchy

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

is a Directed Acyclic Graph (DAG) of vsplit/ecol operators, which can encode all the dependencies among the operators. Thus the traversal of the hierarchy does not rely on the selective mesh for precondition checks. Independent from the multiresolution hierarchy, the selective mesh can be represented in a format that is memory-efficient and optimal for rendering.

2. We present *an algorithm to build the DAG multiresolution hierarchy* from a Progressive Mesh (PM) [62] representation (see Chapter 3). In a simple DAG, an edge from one node to another is *redundant* if there already exists a path from the former to the latter. Redundant edges in a DAG multiresolution hierarchy can cost considerable extra memory and impose unnecessary operations on the edges during the run-time traversal. However, to our knowledge, none of the previous DAG-based view-dependent LOD schemes [31, 57, 121] has addressed this problem. We attempt to *reduce the redundant edges of the DAG hierarchy* as we construct it, while maintaining the time complexity of the construction practical. Our algorithm can build a DAG hierarchy from a PM in $O(n)$ time and with more than 80% redundant edges eliminated, where n is the number of vertices of the model.

3. We conduct studies on several aspects of client-server based view-dependent rendering of our DAG multiresolution hierarchy (see Chapter 4). For the server side, we propose *a numbering mechanism* for efficient incremental traversal of the DAG hierarchy, and *a general traversal management strategy* that is beneficial for interactive frame rate and triangle-budget control. Three different multiresolution hierarchy traversal algorithms are analyzed and compared under this traversal management strategy. The comparison results can be used as a guide to choose the most appropriate traversal algorithm for different system requirements. For the client side, *an efficient representation for selective mesh* is presented for op-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

timal OpenGL rendering and fast mesh modification.

4. We propose *a predictive parallel strategy* to overcome the extra time cost caused by server-side multiresolution hierarchy traversal and the network latency (see Chapter 5). Our solution is to make the client process and the server process run in parallel within a frame, and use the rendering time to cover the network latency. A view-parameters prediction mechanism is developed to make the parallelism of the client and the server feasible. In the LAN experimental environment, our predictive parallel approach dramatically reduces the frame time compared to the previously used sequential method, and can achieve interactive frame rate in the system environment where the sequential method cannot. The view-dependent rendering result of our parallel algorithm with predicted view-parameters is very close to that of the sequential algorithm which uses actual view-parameters.

5. We also *extend the predictive parallel strategy to WAN* environments with longer round-trip times (see Chapter 6). The procedures of getting mesh-updates for multiple frames are parallelized and the long round-trip time is covered by rendering multiple frames. In order to deal with the changing condition of the network, mainly the variable round-trip time, we propose a method to compute the appropriate prediction span and adjust the prediction span dynamically according to the current network condition.

6. For client-server based walkthroughs in scenes with densely distributed objects, we explore *visibility-based scene transmission*. A data prefetching algorithm based on precomputed from-region visibility information is proposed (see Chapter 7). This algorithm predicts and prefetches the precomputed potentially visible set (PVS) of neighboring viewcells that will be needed in the near future according to the current view position of the client. We also provide a delta-transmission algorithm

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

to optimize the prefetching transmission procedure by avoiding transmitting those geometries that are already in the client-side cache.

1.4 Thesis organization

The rest of the thesis is organized as follows:

- **Chapter 2** reviews the research works that are relevant to our research, including mesh simplification, view-dependent LOD, visibility culling, client-server based rendering and motion prediction approaches.

- In **Chapter 3**, we present a new DAG based view-dependent LOD scheme that is favorable to the client-server environments, and we provide an efficient algorithm to construct the DAG hierarchy from the PM presentation with most of the redundant edges eliminated.

- **Chapter 4** focuses on the run-time DAG multiresolution hierarchy traversal and rendering. The numbering mechanism for incremental traversal and the traversal management strategy are presented. Three different traversal algorithms are described and compared through experiments. Also, an efficient representation for client-side selective mesh is proposed.

- In **Chapter 5** we propose the predictive parallel strategy to overcome the cost of multiresolution hierarchy traversal and the network latency for client-server based view-dependent LOD rendering. The one-frame predictive parallel strategy described and implemented in this chapter is very effective in the LAN environment with very short round-trip time.

- **Chapter 6** extends the predictive parallel strategy in Chapter 5 to the WAN

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

environments with relatively long round-trip times. The issues of multi-frame parallelism and dynamically adjusting the prediction span are addressed.

- **Chapter 7** attempts to utilize another rendering acceleration technique, visibility culling, for client-server based walkthroughs in densely occluded scenes. A new position-based neighbor prefetch algorithm is presented for PVS prediction and prefetching. Also a delta-transmission algorithm is proposed to optimize the prefetching transmission procedure.

- **Chapter 8** concludes the thesis and discusses the possible improvements for our works and future research directions.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Chapter 2

Literature Review

This chapter gives an overview of the previous works related to client-server based rendering using LOD or visibility culling techniques. We first go through the literature on mesh simplification in Section 2.1 and view-dependent LOD in Section 2.2. Then we give a brief review of previous works on visibility culling in Section 2.3. Section 2.4 surveys the existing systems of client-server based rendering and the related technique of view-parameters prediction. Section 2.5 summarizes the chapter.

2.1 Mesh simplification

In all types of LOD techniques, the LOD representations are created by certain mesh simplification algorithms. In this section, we review the previous works on simplification of general triangle meshes.

2.1.1 Mesh simplification operators

Mesh simplification can be achieved by local simplification operators and global simplification operators. We will only review the algorithms with local simplification operators, because they are more commonly used in the view-dependent LOD schemes due to their

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

simplicity for implementation and the locality of their modifications on the mesh. A survey of global simplification operators, and a more detailed introduction to mesh simplification can be found in [88].

Many different types of local simplification operators have been proposed in the literatures. The *edge-collapse* operator [67] collapses an edge to a single vertex and removes the triangles adjacent to the edge. The *vertex-removal* operator [114] removes a vertex, along with its adjacent edges and triangles, and re-triangulates the resulting hole. The *triangle-collapse* [60, 48] operator collapses a triangle to a single vertex, and the triangles adjacent to the edges of the collapsed triangle are degenerated into edges. The *cell-collapse* [104, 87] operator collapses the vertices within a certain volume, or a cell, into a single vertex, and all degenerated triangles and edges are removed.

The edge-collapse operator supports fine control of the granularity and smooth transition. Also the mesh modified by an edge-collapse operator can be easily reverted by its inverse operator, a *vertex-split* (see Figure 2.1). Thus the edge-collapse operator has been widely used in continuous LOD [62], view-dependent LOD [136, 63, 37, 98, 71], progressive compression [6] and progressive transmission [57]. The edge-collapse operator again has three variations. The *full-edge-collapse* operator [62, 63] collapses an edge into a new vertex, of which the coordinates and the attribute values can be optimally determined. The *half-edge-collapse* [137, 136, 97, 98] operator (see Figure 2.1) collapses one vertex of the edge to the other, adding no new vertex into the mesh. Therefore, half-edge-collapse requires less number of triangles to be modified than full-edge-collapse and also simplifies the bookkeeping work associated with keeping track of the vertices [88]. The *virtual-edge-collapse* operator [46, 113, 100, 37] collapse two unconnected vertices into a single vertex. No triangles will be removed, but the triangles surrounding the two vertices will be modified.

Both full-edge-collapse and half-edge-collapse are topology-preserving, while the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

virtual-edge-collapse can connect unconnected components and close holes and tunnels, which allows topological simplification and more aggressive geometric simplification. The full-edge-collapse and half-edge-collapse only work on manifold mesh [88], where each edge belongs to exactly two triangles (except for the boundary edges, each of which only belongs to one triangle) and each vertex is adjacent to a single connected ring (or half ring) of triangles. However, virtual-edge-collapse can work on non-manifold meshes.

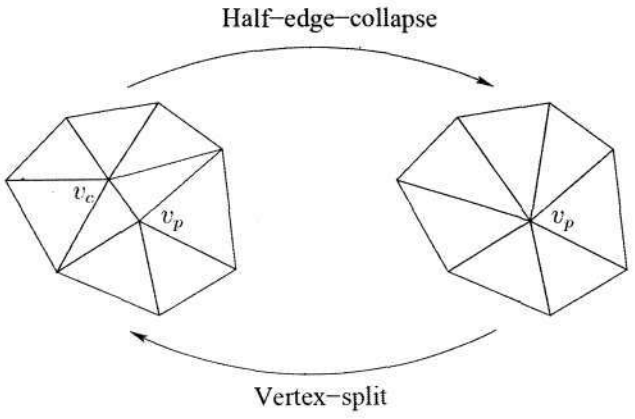


Figure 2.1: A half-edge-collapse, collapsing vertex v_c to v_p, and its inverse vertex-split.

In our research work, we use the half-edge-collapse operators and only consider the manifold operators, where the neighborhoods of the half-edge-collapses are manifold. Although the full-edge-collapse operators can produce higher-fidelity simplifications, we choose half-edge-collapses for their following advantages.

- The vertices of the selective mesh are a subset of the original mesh and no intermediate vertices are generated, so the selective mesh is easier to maintain.
- Unlike the full-edge-collapse operator that must record the offset data for two vertices, the half-edge-collapse only needs to record the offset data for one vertex. This feature not only leads to more efficient mesh updating process but also

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

reduces the size of the mesh-update packet for network transmission.

- The half-edge-collapse has a smaller number of triangles to modify than the full-edge-collapse, since it has a smaller affected neighborhood. This again leads to more efficient mesh updating process and smaller size of the mesh-update packet.

Although we use half-edge-collapse as simplification operator in our work, in the rest of this thesis we will generally refer to it as *edge-collapse* or *ecol* in short and refer to the inverse refinement operator as *vertex-split* or *vsplit* in short. We also refer to both simplification operator and refinement operator as *mesh-update* operator.

One problem the edge-collapse operator should deal with is mesh fold-over [137]. A mesh fold-over happens when the normal of a triangle flips after an edge-collapse (see Figure 2.2). Mesh fold-overs can cause disturbing artifacts such as illumination and texture discontinuity, and should be avoided during the simplification process. They can be detected by measuring the changes in the normals of the corresponding triangles before and after an edge-collapse. A large change in the angle of the normals (usually greater than 90 °) indicates a mesh fold-over [88].

Figure 2.2: A half-edge-collapse, collapsing vertex v_c to v_p, causes an mesh fold-over.

2.1.2 Simplification error metrics

Another important aspect of triangle mesh simplification is the metrics and measurement of the simplification errors. Error metrics can guide and optimize the mesh simpli-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

fication process, and are also the key for run-time LOD selection. The geometric error of a simplified mesh compared to the original mesh can be measured as vertex-vertex distance [104, 87], vertex-plane distance [103, 46], vertex-surface distance [67, 62] and surface-surface distance [127, 22]. Among many proposed error metrics, the quadric error metric by Garland and Heckbert [46] is the most popular and important one, as it can provide efficiency, quality and generality at the same time.

The quadric error metric measures the error of simplification as the sum of squared vertex-plane distances. Each vertex in the original mesh has a set of supporting planes, namely the planes of the triangles that meet at the vertex. When an edge is collapsed, the set of supporting planes for the resulting vertex is the union of the two sets of supporting planes for the two edge vertices. At anytime in the simplification process, the error E_v of a vertex v is the sum of squared distances to its supporting planes:

$$E_v = \sum_{p \in plane(v)} (p^\mathsf{T} v)^2 = \sum_p (v^\mathsf{T} p)(p^\mathsf{T} v) = v^\mathsf{T} \left[\sum_p (pp^\mathsf{T}) \right] v = v^\mathsf{T} \sum_p (Q_p) v = v^\mathsf{T} Q_v v \quad ,$$

where Q_p is the fundamental error quadric represented by a 4×4 matrix containing 10 floating-point numbers for one supporting plane, and the Q_v is the sum quadric associated with each vertex. In this way, the set of supporting planes of a vertex can be implicitly tracked using a single matrix. Updating the set of supporting planes for the resulting vertex after an edge-collapse is simplified as an addition operation of two 4×4 matrices.

Garland and Heckbert [47] later proposed to use higher-dimensional quadrics to track and minimize the errors of simplifications of models with attributes. Other variations of quadric error metric include the works of Lindstrom *et al.* [84] and Hoppe [66]. In [84], the position of the new vertex is selected based on the volume enclosed by the surface and the local area surrounding the edge being collapsed. In [66], the quadrics are constructed based on separating the geometric error from the attribute errors. The

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

attribute quadrics are sparse matrices and the space requirement is linear instead of quadric as in [47]. In both [84] and [66], the quadrics are in a "memoryless" fashion. Instead of maintaining the quadrics from the original mesh and summing them through each edge-collapse as in [47], the quadrics are computed based on the geometry and attributes of the mesh simplified so far.

In our research work, we use the basic quadric error metric in the mesh simplification process to generate the progressive mesh (PM) [62] representations.

2.2 View-dependent LOD

2.2.1 Continuous LOD

View-dependent LOD is an extension of continuous LOD. The first continuous LOD scheme is the Multiresolution Analysis (MRA) [85, 86] proposed by Lounsbery *et al.*, based on subdivision surfaces and wavelets. MRA stores a simple base mesh together with a stream of refinement details (wavelets) to produce a continuous-resolution representation. However, MRA requires the original fine-level mesh to be the result of subdividing a simple mesh. In other words, the mesh must have recursive 1-to-4 subdivision connectivity. As a result, an arbitrary initial mesh can only be converted into MRA representation with certain error tolerance [33].

Hoppe's progressive mesh (PM) [62] is the most popular continuous LOD scheme. The PM representation is based on edge-collapse operators. It has a similar structure as the MRA representation, but it can provide lossless continuous representations for meshes with arbitrary connectivity. An original mesh M can be simplified into a base mesh M_0 by a series of edge-collapse operators:

$$(M = M_n) \xrightarrow{ecol_n} M_{n-1} \xrightarrow{ecol_{n-1}}, \ldots, \xrightarrow{ecol_2} M_1 \xrightarrow{ecol_1} M_0 \quad .$$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

The PM representation of the original mesh M is represented by the base mesh M_0 followed by a list of vertex-splits, in the form $(M_0, \{vsplit_1, vsplit_2, \ldots, vsplit_n\})$, where $vsplit_i$ is the inverse operator of $ecol_i$. The efficient implementation of progressive mesh is presented in [64].

Progressive mesh provides an elegant solution to storing, rendering and transmitting meshes in a progressive manner. This continuous-resolution representation supports progressive transmission, mesh compression and smooth morphing between different LOD approximations. Although a selective refinement method on the PM presentation is also outlined in [62], it is not efficient enough for real-time performance and no facility is provided for selective mesh adaption as the view-parameters change [137, 63]. Since then, many research works on view-dependent LOD schemes have been presented, focusing on real-time selective refinement according to current view-parameters.

2.2.2 View-dependent error criteria

In [137], Xia *et al.* proposed the idea of using view-dependent information, such as local illumination, screen-space projection, back-face culling and silhouette boundaries to guide view-dependent level-of-detail selection. The local illumination criterion allows variation of details according to the intensity gradient of the light source. The regions with low intensity gradients are drawn in lower details, while the regions with high intensity gradients are drawn in higher details. The screen-space projection criterion aims to simplify the regions where the projected areas are sufficiently small. The back-face criterion is used to reduce the details of the back-facing regions and the silhouette criterion seeks to identify and preserve the silhouettes since they are perceptually important. In the works of Xia *et al.* [137, 136], each node in the merge tree stores the Euclidean distance for splitting the vertex to its children as well as the distance at which it will merge to its parent. These two scalar values are used to quickly measure the screen-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

space projection. Each node also stores a normal cone [116], bounding all the subtree normal vectors. The normal cone is used for back-face check, silhouette check and local illumination check. The screen-space projection, back-face and silhouette criteria are also used in [87].

In [63], besides the screen-space projection and back-face criteria, Hoppe also proposed the out-of-view criterion that aims to coarsen the mesh outside of the view-frustum. Pajarola *et al.* [97, 98] presented optimized implementations of out-of-view, screen-space projection, back-face and silhouette criteria. They also proposed a shading criteria with the purpose of simplifying the regions where the difference in lighting and shading effects is not visually significant. The shading criterion is not view-dependent. The authors suggested considering both the shading criterion and the view-dependent local illumination criterion to achieve a better fidelity.

2.2.3 View-dependent LOD schemes

The core structure of a view-dependent LOD scheme is a multiresolution hierarchy. In most existing view-dependent LOD schemes a hierarchy of vertices is used, which is often called *vertex forest* or *vertex trees*.

Xia *et al.* [137] developed the merge tree, which is probably the first view-dependent LOD scheme for general triangle mesh based on half-edge-collapse operators. The merge tree is a binary forest of vertices built by determining parent-child relationships among as many vertices at each level as possible. All the vertices that are not children are promoted to a higher level in the hierarchy. The active vertices of the selective mesh are a subset of the vertex forest. However, not any subset can reconstruct a valid mesh. In order to prevent the mesh fold-over problem, the authors defined dependencies amongst the nodes of a merge tree to enforce a partial order of the vertices. A child vertex c can merge to its parent vertex p only when all the vertices defining the boundary of

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

the influenced region are present and the boundary vertices cannot merge with other vertices until c first merges with p. Similarly, p can split into c and p only when all the vertices defining the boundary of the influenced region are present and the boundary vertices cannot split until p first splits into p and c. The active vertex list and triangle list are determined by updating the corresponding list from the last frame, using the frame-to-frame coherence.

Hoppe [63] proposed a *view-dependent progressive mesh* (VDPM) framework based on the original PM representation [62]. A binary vertex forest is constructed top-down as a PM is loaded, using a simple traversal of the vertex-split list. Similarly to the merge tree [137], the vertices of the selective mesh form a *front* [63] through the vertex forest and the view-dependent refinement is achieved by traversing the vertex hierarchy up and down from the current position. A new definition of dependency or legality for the vertex-split operators is proposed [63]. The precondition for a vertex-split to be legal is the presence of the two pairs of neighboring faces adjacent to the vertex to be split. And the precondition for an edge-collapse is that the two pairs of faces are adjacent to the two faces to be collapsed. The problems of geomorph and frame-rate regulation are also addressed. Hoppe emphasized that the selective mesh should be consistent, which means given the view-parameters the selective mesh should be unique, regardless of the sequences of vertex-splits and edge-collapses that lead to it. The consistency property of VDPM is later proved formally by Grabner [52]. Hoppe also presented a specialized version of VDPM for interactive terrain rendering [65].

Luebke and Erikson [87] introduced a generalized view-dependent LOD framework called Hierarchical Dynamic Simplification (HDS). The entire model is constructed into a vertex tree by hierarchical vertex clustering and the vertex tree is queried dynamically to generate a simplified scene. The nodes will be collapsed to their parents or expanded to their children based on their projected sizes. Each parent vertex in the vertex tree

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

contains one or more child vertices. A node collapsing is operated by using the parent vertex to represent its child vertices and removing degenerated triangles from the active display list. A node expanding will split the representative vertex into its children and introduce new triangles into the display list. No precondition is required for a node to be collapsed or expanded. When a vertex, which is not available in the current selective mesh, is needed by the new triangles added to the display list, the first active ancestor of the vertex in the hierarchy is used. The problem of triangle-budget simplification is addressed. The boundary nodes between the split nodes and the collapsed nodes are managed in a priority queue, sorted by the screen-space error. The boundary nodes are expanded from the top of the queue until one more expansion would exceed the triangle-budget. The authors also outlined the asynchronous simplification, where the dynamic LOD selection stage and the rendering stage can be parallelized over multiple processors.

El-Sana *et al.* [37] introduced the view-dependence tree that is a generalization form of the merge tree [137, 136]. Since the construction of the view-dependence tree is based on virtual-edge-collapses, it is capable of handling non-manifold mesh and topology simplification. Another important new feature compared to the merge tree is that the dependencies lists [137, 136] are not explicitly stored. A concept of *implicit dependency* is proposed, which relies on the enumeration of the vertices generated by the edge-collapses. The n vertices of the original model are assigned the vertex ID from 0 to $n-1$. Each edge-collapse generates a new vertex with the vertex ID one more than the greatest vertex ID so far. According to this vertex enumeration scheme, the legal precondition for an edge-collapse is that the ID of the new vertex from the edge-collapse is less than the vertex IDs of the parents of the boundary vertices. The legal precondition for a vertex-split is that the ID of the vertex to be split must be greater than the vertex IDs of all its neighbors. The paper reported that the implicit

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

dependency can reduce the space requirement by a factor of two.

To *et al.* [125] attempted to reduce the neighboring dependencies to a minimum. They also constructed a binary vertex forest by half-edge-collapse operators. The unique feature is that with each vertex in the vertex forest, the fan of adjacent triangles is stored. Any active vertex is legal to split or collapse as long as the simple parent-child relationship defined by the vertex forest is satisfied. The active triangle list is the union of the fans of triangles stored with all the active vertices. If two fans have overlapped triangles, the triangles at a higher resolution level are selected. Similar to the strategy in [58], if any triangle requires vertices that are not in the current selective mesh, the nearest ancestors in the vertex hierarchy are used to represent the unavailable vertices. When rendering the selective mesh, the active nodes of vertices are traversed and their fans of triangles are retrieved to form a triangle list to display.

Kim and Lee [71] managed to achieve a truly selective refinement scheme by a different approach. They proposed the concept of *progressive transitive mesh space*, which is the set of all possible selective meshes for a PM representation. They observed that the dual pieces of the vertices can clearly enumerate all possible selective meshes. By examining the properties of the dual pieces, the authors concluded that any vertex-split or edge-collapse can be applied without incurring additional vertex-splits or edge-collapses. The key step is to locate the two *pivot vertices* [71] of the vertex-split or edge-collapse operator in the current 1-ring neighbor. This is achieved by defining the *fundamental cut vertices* [71] of vertex-split or edge-collapse operators and then finding the first active ancestors of the fundamental cut vertices up in the vertex hierarchy. However, as the authors pointed out, since this scheme allows abrupt resolution changes, the mesh fold-overs may happen.

Although in the above two schemes [125, 71], the mesh can be truly selectively refined without legality constraints, the penalty is the quality of the selective mesh. The

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

problems such as mesh fold-overs and sliver triangles may happen. In order to prevent such undesirable artifacts, run-time checks have to be performed, which are more expensive than the legality precondition checks [137]. Since the qualities of the triangles are usually kept under control in the simplification procedure, using precondition checks to ensure the partial order of the refinement operators and hence the quality of the triangles in the selective refinement procedure is more computationally efficient.

Instead of the hierarchy of vertices, the hierarchy of mesh-update operators is used as the multiresolution hierarchy in the following view-dependent LOD algorithms.

Pajarola and DeCoro introduced FastMesh [97, 98], which uses a binary hierarchy of half-edge-collapses to define the multiresolution mesh. The binary hierarchy of half-edge-collapses is a direct translation of the binary vertex forest. Each node in the hierarchy is a half-edge data structure [130], a simplified version of winged-edge data structure [7]. The authors utilized the correspondence relationship between the half-edge-collapse operator and the half-edge data structure for fast mesh updating. The selective mesh is represented by all the nodes (half-edges) in *split* status, and the rendering is done by recursively traversing the nodes and constructing the active triangles. A topological constraint for valid half-edge-collapse to prevent topology changes and non-manifold connectivity is checked during run-time. The authors also defined some soft constraints that can be checked optionally online to improve the quality of the triangles generated.

Gueziec *et al.* [58] proposed a surface partition representation for encoding multiple continuous LODs of a model. Half-edge-collapses are used to simplify the model. The parent vertex of the half-edge-collapse operator becomes the representative of the child vertices. The vertices and the triangles of the original model are assigned levels according to the relative dependencies among the half-edge-collapses. The surface is then partitioned by the levels of the vertices and the triangles. In a surface partition

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

representation, the vertices and triangles of the mesh are stored in the order of their levels together with the vertex representative arrays. Similar to the progressive mesh representation, this format supports progressive transmission, but the refinement details are recorded in batches according to their levels. The authors also described a locally selective LOD scheme by building a Directed Acyclic Graph (DAG) of the half-edge-collapses to encode the partial order. In their later work [57], a framework for streaming geometry in VRML [9] is proposed based on the surface partition representation.

The Multi-Triangulation (MT) demonstrated by DeFloriani *et al.* [31] is proposed as a framework for generic mesh-update operators, but all the works on MT are based on vertex insertion/removal and re-triangulation. Two forms of MT representations have been presented. The *explicit MT* has a more complex structure. It is a labelled DAG where the nodes represent mesh-updates and the arcs, labelled with removed/added triangles, represent the dependencies between mesh-updates. The *implicit MT* does not store triangles with the arcs and the storage cost is reduced. However, the mesh-updating procedures are more computationally expensive because of the triangulation operations. The selective mesh can be extracted from the MT hierarchy according to the current view.

2.3 Visibility culling

Visibility culling aims at quickly rejecting invisible geometries before actual hidden-surface removal is performed by the graphics hardware, so that only subset of primitives that may contribute to at least one pixel of the screen will be sent to the rendering pipeline [23]. Visibility culling includes view-frustum culling, back-face culling and occlusion culling [23]. View-frustum culling and back-face culling have become common practices in computer graphics, while occlusion culling is still an active research field.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Occlusion culling is especially effective for walkthroughs of urban area, since most objects are hidden behind the occluders with respect to current viewing position [68]. In [23], a comprehensive survey of available occlusion culling algorithms is presented. In this section, we will only give a brief review of some important occlusion culling algorithms. We follow the classification of *from-point visibility* vs. *from-region visibility* in [23].

2.3.1 From-point visibility

In from-point visibility algorithms, the visible set is computed with respect to the current viewpoint and has to be computed each time the viewpoint changes. Hierarchical Z-Buffer (HZ) [54] and Hierarchical Occlusion Map (HOB) [141] are two *image-space* from-point occlusion culling algorithms, as the occlusion culling is performed in image space. In HZ [54], a Z-pyramid structure is proposed, which is a multi-layered z-buffer with the resolution of each level growing coarser from bottom to top. Octree is used to organize the scene and test for occlusion hierarchically. Like HZ, HOB [141] also has a multi-layered structure with the resolutions growing from finer to coarser. However, HOB only stores opacity information and hierarchical bounding-box is used for occlusion test.

There are also *object-space* from-point occlusion culling algorithms. Coorg and Teller [26] used separating and supporting planes to characterize the occlusion caused by a single large convex occluder. The space is partition into three regions: fully visible region, partially occluded region and fully occluded region. Hudson *et al.* [69] exploited the fact that a viewer cannot see the occludee if it is inside the shadow of the occluder. They proposed an algorithm using the shadow frusta to cull out the objects hierarchically. Wonka and Schmalstieg [133] also took advantage of occluders' shadows. They render the occluders' shadows by orthogonally projecting the shadows

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

into a bitmap coincident with the ground plane. Then the Z values of the shadows form the cull map.

2.3.2 From-region visibility

From-region occlusion culling algorithms compute the visible set for a region, also called viewcell [76]. The visible set is valid as long as the viewpoint is within the viewcell, thus the computation cost for the viewcell is amortized over all the frames generated from the given viewcell. The basic idea behind from-region visibility is to partition the space into a number of viewcells and compute the *Potentially Visible Set* (*PVS*) [23] for each viewcell. For most from-region occlusion culling algorithms (with the exception of [75] and [76]), PVS computation is performed during the preprocessing stage and the PVS information is stored for run-time use. In rendering stage, the culling task is simplified into PVS querying and is only needed when the viewpoint is crossing the borders of the viewcells, thus the rendering for each frame is speeded up.

In [108], Saona *et al.* proposed a method to compute the visibility for an octree node by first computing the from-point visibility for the eight vertices of the node. The invisible portion caused by a single convex occluder for the node is the intersection of the invisible portions caused by the occluder for the eight vertices. Schaufler *et al.* [109] presented an algorithm to exploit occluder fusion by extending the occluder into adjacent opaque regions of the space. A conservative scene discretization is used to decouple the effectiveness of the occlusion from how the scene is modeled. Durand *et al.* [32] proposed the concept of *extended projection* for a viewcell. The extended projection of an occluder is defined as the intersection of its projections from all the points in the viewcell, and the extended projection of an occludee is defined as the union of its projections. Wonka *et al.* [134] presented a from-region occlusion culling algorithm for 2.5D urban walkthrough, based on their former algorithm of from-point occlusion

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

culling [133]. The main idea is that conservative visibility for a viewcell can be calculated by shrinking occluding objects and sampling visibility from a set of discrete points on the boundary of the viewcell. Koltun *et al.* [75] introduced the notion of virtual occluder for a viewcell, which is a view-dependent simple convex object guaranteed to be fully occluded from any given point within the viewcell and serves as an effective occluder for the given viewcell. As an intermediate PVS representation, virtual occluders compactly represent the aggregate occlusion for a given viewcell. Another algorithm introduced by Koltun *et al.* [76] utilizes the dual ray space to represent visibility in a two-dimensional space. Standard rendering hardware is used for rapid visibility computation, so that the PVS for a viewcell can be computed online.

Another type of from-region visibility algorithms are called *cell-and-portal* [124, 123]. They are based on the characteristics of architectural scenes, which can be naturally subdivided into *cells* (rooms) by major occluders (walls). Visibility occurs through openings on the walls (windows and doors), which are called *portals*.

2.4 Rendering over network

Many research works have been carried out on the area of rendering large complex scenes in the networked virtual environments. A lot of distributed systems for shared virtual environments can be found in the literature, such as SIMNET [91], NPSNET [13], DIVE [42], SPLINE [129], MASSIVE [55], SmartCU3D [128] and Virtual Campus [120]. These works are more focused on interactions among multiple users. In this section, we will only review the research works that use LOD or visibility culling techniques to optimize the geometry data transmission and rendering in client-server based virtual environments. More specifically, we will focus on the use of view-dependent multiresolution meshes and from-region visibility in client-server based rendering. The issue of

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

view-parameters prediction, which is closely related to data prefetching on client-server based rendering systems, is also reviewed.

2.4.1 General client-server rendering systems

The LOD and visibility culling techniques have been used in many client-server rendering systems. Funkhouser and Sequin's work [45] provides a base to many other subsequent research works on networked rendering. A set of benefit factors are proposed, including the visibility of the object and the accuracy of the representation of the object, to select the most beneficial LOD representations of the scene for the estimated rendering budget. In the algorithms of both Teler *et al.* [122] and Zach *et al.* [140], the scene stored on the server is regarded as a collection of 3D objects, each of which may have one or more discrete LOD representations. The retrieval decision is made by a certain benefit/cost equation based on the benefit factors in [45]. For a given set of view-parameters, which consist of position and viewing direction, the set of representations that can provide the best benefit are selected. Schneider and Martin introduced [112] a network graphics framework that allows the client to select the best method of transmitting the model represented as various LODs, according to its numerous system performance parameters.

In [110], Schmalstieg and Gervautz introduced the demand-driven geometry transmission protocol. The authors attempted to optimize the geometry data transmission with the concept of *area of interest* (AOI), which is a circular area around the viewpoint. AOI can be viewed as a simple form of visibility culling, as only the objects within the client's AOI are sent from the server and stored in the main memory of the client. The AOI in [110] is divided into circular zones, where objects in different zones are represented by different LODs. Later in [61], the *smooth levels of detail* [111] is incorporated into the system to eliminate the popping effects resulted from switching

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

between discrete LODs. A recent client-server based visualization framework presented by Sahm *et al.* [105] also prefetches the relevant parts of the scene according the client's AOI. The multi-layered AOI can prioritize the data inside the AOI, so that the objects nearer to the viewpoint have a higher priority for transmission.

Popescu *et al.* [99] presented a transmission scheduling policy for a scene of a set of objects represented by progressive meshes. The successive progressive records in a PM are aggregated into blocks and these blocks are the basic units for the transmission scheduler. Visibility tests are performed and the blocks of the visible objects are measured according to a weighing formula and added into the queue of the transmission scheduler.

Other research works on scene transmission in networked environments can be found in [18, 17, 40, 53].

2.4.2 Client-server rendering with view-dependent LOD

There are some research works especially focused on view-dependent multiresolution mesh for client-server based rendering. To *et al.* [125] noticed that previous view-dependent LOD schemes are not flexible enough for selective transmission due to the neighboring dependency constraints. They presented a progressive and selective transmission scheme by reducing the neighboring dependency constraints to a minimum. Their framework allows visually important parts of the model to be transmitted to the client at higher priorities. At the client, the vertex forest is progressively reconstructed as the node records are received, so that the view-dependent rendering can be performed locally on the partially reconstructed vertex forest. A similar progressive streaming framework is developed by Kim *et al.* [72] for their truly selective refinement scheme [71]. The vertex front on the server keeps track of the vertices that have already been transmitted to the client, and is only updated downward in the vertex hierarchy

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

as more vertex-splits are transmitted according to the requirements of the client. The client maintains a partially constructed vertex hierarchy as the vertex-splits are progressively received. The vertex front at the client moves freely within the partial vertex hierarchy for view-dependent rendering.

The paradigm used in the above two schemes can be considered as interruptible downloading guided by client-side view-parameters. This paradigm can minimize the network communication cost since each node in the multiresolution hierarchy will only be transmitted once. However, it is not suitable for low-capacity clients, as reconstructing the multiresolution hierarchy on the client imposes high memory requirement. Performing view-dependent multiresolution hierarchy traversal locally is also too expensive for low-capacity clients.

Southern *et al.* [121] proposed a view-dependent transmission scheme for *stateless* clients. The client is completely dependent on the server to maintain the state of the refinement. It never receives the entire model, but only the base representation and a sequence of necessary vertex-splits to increase the details of selected regions. Instead of using vertex forest as the multiresolution hierarchy, a DAG of vertex-splits is constructed according to the legality precondition defined by Hoppe [63]. However, only top-down traversal on the DAG hierarchy based on breadth-first search is developed. Also no incremental mesh-updating mechanism is provided for the selective mesh. Instead, each time the view-parameters change, a new mesh has to be created from the base mesh and the sequence of vertex-splits.

DeFloriani *et al.* [30] introduced a client-server architecture based on their earlier Multi-Triangulation (MT) [31] representation. Incremental coarsening and refining traversals are performed through the DAG hierarchy of mesh-updates on the server. The client only stores the current selective mesh that will not exceed the given memory budget. In order to reduce the network traffic, a compressed representation of the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

mesh-updates is proposed. The mesh-updates are encoded into a binary stream on the server and decoded on client to modify the selective mesh. A simple FIFO client-side cache is used to further reduce the bandwidth occupancy. As the mesh-updates are vertex-insertion/vertex-removal operators, the mesh-updating procedure involving re-triangulation is quite complex and requires the format of the selective mesh to support adjacency queries for vertices and edges. The authors implemented the selective mesh as the *doubly connected edge list* (DCEL) data structure [93]. This format, although is versatile for mesh adjacency queries, is not optimal for rendering and less space-efficient for low-capacity clients. Only a simulation of this client-server architecture on a stand-alone computer is presented in [30].

El-Sana [34] improved the earlier view-dependence tree [34] for a multi-user system by separating the active-nodes list from the view-dependence tree. The server loads the view-dependence tree of the model into the shared memory and sets multiple *interfaces* associated with multiple clients. Each client only maintains the current selective mesh in the form of a vertex list and a triangle list. The implicit dependency [37] mechanism is used in the server-side traversal to prevent mesh fold-overs. Each vertex in the view-dependence tree needs to keep two integers. One is the maximum ID among its adjacent vertices and the other is the minimum ID among the parents of the collapse boundary vertices. The two IDs of relevant nodes are updated after each vertex-split or edge-collapse operator is performed. Updating these two IDs requires the adjacency information of the current selective mesh. In order to solve this problem, the server has to keep a copy of the current selective mesh for each client. This framework is later extended [35] to wide area network (WAN) environments. The view-dependence tree is divided into blocks so that the mesh-update operators are transmitted on block basis. In order to avoid network latency, the client caches a fraction of the view-dependence tree in the form of blocks and the server predicts the blocks that may be needed in the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

future and sends them to the client.

As we mentioned in Chapter 1, the compression technique used in [30] cannot solve the type of network latency problem caused mainly by round-trip time, and the cache technique used in [35] is not favorable to the clients with small memories.

2.4.3 Client-server rendering with occlusion culling

Occlusion culling algorithms are originally proposed to avoid overdraw and to speed up rendering for densely occluded scenes, but they are also applied to selecting what the server should transmit to the client in the client-server walkthrough scenarios [43, 25, 75, 76] or similarly to disk-to-memory prefetch in out-of-core rendering [44, 3, 27].

In [25], Cohen-Or *et al.* proposed a visibility streaming framework for network-based urban walkthroughs. The server determines a conservative set of visible objects for a ϵ-neighborhood of a given viewpoint, so that the client can render the scene independently from the server as long as the viewpoint moves within that ϵ-neighborhood. In [43, 44], Funkhouser utilized the precomputed cell-to-cell visibility in the indoor architectural walkthrough to reduce the exchanges of update messages in the multi-client system and to prefetch the objects that may potentially be visible to the observer. Koltun *et al.* [75, 76] proposed two different from-region occlusion culling algorithms for client-server based urban walkthrough. Particularly in [76], a simple neighbor prefetch algorithm is presented, which takes advantage of the predicting capability of the from-region visibility algorithm and prefetches the PVS of neighboring viewcells to the client.

2.4.4 View-parameters prediction

Prefetching techniques are very important for client-server based rendering systems to hide the network latency. For view-dependent rendering, the key to prefetching is view-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

parameters prediction. Teler *et al.* [122] proposed a very simple prediction method, under the assumption that once the user has started a particular type of motion, this type of motion will continue in the near future. Three simple motion types are considered: standing still, turning about and moving in a straight line. In [99], Popescu *et al.* attempted to improve the efficiency of their scheduling policy by predicting the future motion of the viewpoint with a "random walk" model, which assumes that the viewpoint is moving at a constant speed, with the direction angle determined by a probability function.

A related research field is the motion prediction techniques used for tracking the head movement with Head-Mounted Displays (HMDs) in virtual realities and augmented realities [5, 73]. A typical prediction system in this field is a Kalman filter [70] based on a certain motion model, usually the Position-Velocity model or the Position-Velocity-Acceleration model [12].

The standard for Distributed Interactive Simulation (DIS) [1] provides a good reference for motion predictions. The concept of *dead-reckoning* is proposed to reduce the network communication and latency among the simulation environment. Instead of frequently sending the update information for each entity among the simulation environment, each host in the simulation extrapolates the position and other states of each entity according to a specified dead-reckoning algorithm. Only when the differences between the extrapolated states and the actual states exceed a predefined threshold, will the update information be sent among the simulation environment. The DIS standard defines nine dead-reckoning formulas for first-order motion (uniform motion in a straight line), second-order motion (motion with uniform acceleration) and rotation.

In [138], the dead-reckoning algorithm is used for predicting the viewpoint motion in a web-based shiphandling training system. Guthe *et al.* [59] approximated the movement of the viewpoint as a quadric function using the view positions and directions of

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

the last three frames. Their prediction model assumes that the viewpoint is moving at a constant speed within one frame and with a constant acceleration between successive frames.

Chim *et al.* [18, 17] and Chan *et al.* [14, 15] have proposed motion prediction schemes for mouse-based interaction. In [18, 17], an *exponential weighted moving average* (*EWMA*) scheme is proposed, in which the future mouse movements are predicted based on weighted past movement vectors, with each preceding vector having an exponentially decreasing weight. In [14, 15], Chan *et al.* presented a hybrid motion prediction method. When the mouse is moving in a low velocity, the first-order motion model is used for prediction, and when the mouse is moving in a high velocity, the prediction formula is switched to an elliptic model.

2.5 Summary

Client-server based rendering using LOD or visibility culling techniques is an interesting topic for computer graphics applications. In this chapter, the research works related to the background of view-dependent LOD and visibility culling techniques are reviewed. Several important view-dependent LOD schemes and occlusion culling algorithms in the literature are outlined. We have also surveyed the existing client-server based rendering systems. The existing architectures of view-dependent multiresolution systems are compared and the respective strengths and weaknesses are analyzed. We conclude that none of the previous view-dependent multiresolution systems is favorable to low-capacity clients on networks with relatively long round-trip times. Other works related to client-server based rendering systems, such as view-parameters predictions, are also reviewed.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Chapter 3

A DAG Hierarchy for View-dependent Multiresolution

3.1 Introduction

Hoppe's Progressive Mesh (PM) [62] is an effective representation for progressive transmission because of the progressive structure of the list of vertex-splits. However the PM representation itself is not efficient for view-dependent rendering. Hoppe later proposed a view-dependent progressive mesh scheme (VDPM) [63], which decomposes the progressively arranged vertex-split operators and reconstructs them into a binary vertex forest (vertex trees). As we discussed in Chapter 2, the traversal of VDPM is dependent on the current selective mesh for precondition checks during run-time, which is an unfavorable factor for client-server based rendering. In this chapter, we present a new multiresolution hierarchy more suitable for client-server based view-dependent rendering, which is a Directed Acyclic Graph (DAG) of vertex-split operators.

Compared to the vertex forest hierarchy, the DAG hierarchy of vertex-splits has several favorable characteristics. First, the vertex trees cannot encode sufficient dependency information and thus have to rely on the condition of the current selective mesh to check the dependencies of the vsplit/ecol operators [136, 63] during the traver-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

sal. The DAG hierarchy, on the contrary, can encode all the dependencies among the nodes [29], which makes the hierarchy traversal independent of the selective mesh. This feature is especially desirable for the client-server based view-dependent LOD system, where the multiresolution hierarchy and the selective mesh are stored separately on the server and the client.

Second, the DAG hierarchy has a much smaller number of front nodes than the vertex trees. The *front* [63] is the set of boundary nodes between the split nodes and the collapsed nodes. It corresponds to the current selective mesh and also is the set of the candidate nodes for further refinement/simplification operations. In the vertex trees, the front nodes include all the vertices of the current selective mesh, the number of which is about half the number of the triangles. While in the DAG hierarchy, the front nodes are composed of only the nodes that are legal to split or collapse, and the size is only about 1/4 to 1/3 the number of the triangles of the selective mesh (shown in Chapter 4).

The third, unlike the vertex trees that use one front for both simplification and refinement, the DAG hierarchy has two separate fronts (refer to Chapter 4), one for simplification traversal to generate collapse operators and the other for refinement traversal to generate split operators. This feature provides the flexibility to control the refinement traversal and the simplification traversal individually, as demonstrated in Chapter 4.

Although DAG as the multiresolution hierarchy has been seen in [31, 57, 121], to our knowledge, no previous work has dealt with the problem of redundant edges of DAG multiresolution hierarchies. In this chapter, we present the algorithm to build the DAG hierarchy of vertex-splits from a PM representation and manage to eliminate the redundant edges. The algorithm of eliminating all the redundant edges has a too high time-complexity for practical use. Therefore, we present a *mini-redundant* algorithm, which can eliminate most of the redundant edges and is time-efficient. After the DAG

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

hierarchy of vertex-splits is built, we assign each node a set of visibility attributes that are used for run-time view-dependent rendering. The detailed method of hierarchically computing the visibility attributes is presented.

The connections among the DAG nodes are built according to a certain definition of legal surrounding condition for a vsplit/ecol operator. For a given PM, different legality definitions will result in different DAGs with different levels of mesh-adapting flexibility and different levels of dependence on the selective mesh for the vsplit/ecol operators. Therefore, before we present the DAG construction algorithm, we first introduce a new legality definition that is flexible enough for selective refinement and can also guarantee the independence of the vsplit/ecol operators from the adjacency information of the selective mesh.

The rest of the chapter is organized as follows. Section 3.2 defines the legality condition and dependency among the vertex-splits. In Section 3.3, we describe the algorithm of constructing the DAG hierarchy of vertex-splits from the PM representation and the algorithm of eliminating the redundant edges. In Section 3.4, the method of constructing the hierarchy of visibility attributes is presented. Section 3.5 shows the experimental results of the DAG construction algorithm. Sections 3.6 summarizes the chapter.

3.2 Legality condition

As discussed in Chapter 2, in order to ensure that the selective mesh is well-defined and to prevent the mesh fold-overs, the legality condition has to be defined for the vsplit/ecol operators to be safely applied. In other words, a vsplit/ecol operator can be applied to the selective mesh only when certain vsplit/ecol operators have already been applied and the necessary triangles and vertices are present in the current selective

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

mesh.

Different definitions of the vsplit/ecol legality have been proposed [137, 136, 63, 37]. Xia *et al.* [137, 136] and El-Sana *et al.* [37] proposed that an edge-collapse or a vertex-split is legal only when its surrounding vertices are identical to those in the stage of mesh simplification. As Figure 3.1 shows, vertex v_c can collapse to vertex v_p only when v_0, v_1, ..., v_6 are present as neighbors of v_p and v_c. And vertex v_c can be split from v_p only when v_0, v_1, ..., v_6 are present as neighbors of v_p.

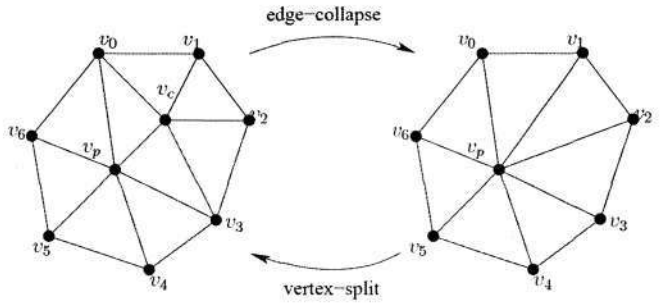


Figure 3.1: Legality definition for vertex-split and edge-collapse in [137, 136, 37].

Hoppe [63] found that the full set of the surrounding vertices is not necessary. He proposed that the existence of the two pairs of triangles adjacent to the two collapsed triangles (f_l and f_r in Figure 3.2) are sufficient legal precondition for a vertex-split or edge-collapse operator. Figure 3.2 illustrates this definition. Vertex v_c is legal to collapse to vertex v_p if f_{n0}, f_{n1}, f_{n2} and f_{n3} are present and adjacent to f_l and f_r. Vertex v_c is legal to be split from v_p if f_{n0}, f_{n1}, f_{n2} and f_{n3} are present and adjacent to v_p.

Hoppe's definition for legality is flexible for highly adaptable refinement, however this flexible definition leads to the uncertainty of the vsplit/ecol operators. The modifications that a vsplit/ecol operator makes to the selective mesh are dependent on the current formation of the neighborhood. For example, in Figure 3.3(a) and (b), both

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

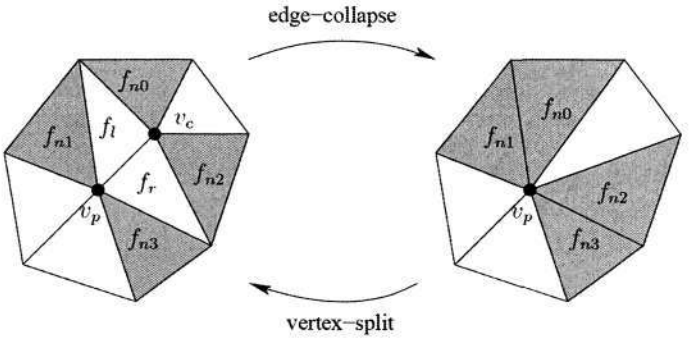


Figure 3.2: Legality definition for vertex-split and edge-collapse in [63].

neighborhood formations are legal for v_p to split, however their sets of adjusted triangles are different, so the set of adjusted triangles for a vsplit/ecol operator has to be determined online by exploiting the information of the surrounding neighborhood. Therefore this definition does not support the memory-efficient and rendering-optimal format for the selective mesh, which requires the modification procedure of a vsplit/ecol operator to be independent of the topology of current selective mesh.

We need a legality definition that can ensure a unique surrounding neighborhood, so that the modifications a vsplit/ecol operator makes to the vertices and triangles can be predetermined offline. We here propose a new legality definition (refer to Figure 3.4), which is more flexible than Xia and El-Sana's but more restricted than Hoppe's.

- A vertex v_c is legal to collapse to v_p, if the two pivot vertices v_r and v_l (see Figure 3.4) are present, and all the adjusted triangles as in the stage of mesh simplification (f_0, f_1 and f_2 in Figure 3.4) are present and adjacent to v_c.
- A vertex v_c is legal to be split from v_p, if v_r and v_l are present, and all the adjusted triangles as in the stage of mesh simplification are present and adjacent to v_p.

By our legality definition, a vertex-split operator makes the following modifications to the selective mesh:

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

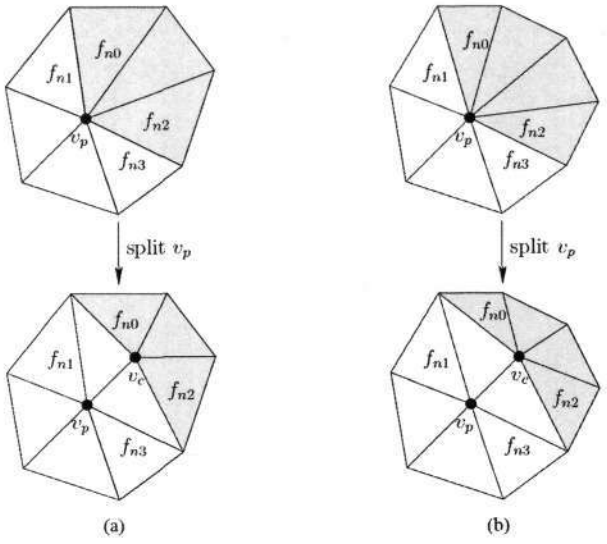


Figure 3.3: Different possibilities of the surrounding topology by Hoppe's legality definition. The adjusted triangles are shaded. Both (a) and (b) are legal condition to split vertex v_p, however there are three adjusted triangles in (a) and four in (b).

1. Add the vertex v_c to the selective mesh, including its coordinates, normal or other attributes.

2. Add the two collapsed triangles f_l and f_r, constructed by (v_p, v_c, v_l) and (v_c, v_p, v_r), to the selective mesh.

3. Replace v_p by v_c in all adjusted triangles.

An edge-collapse operator makes the reverse modifications to the selective mesh:

1. Delete the vertex v_c from the selective mesh.

2. Delete the two collapsed triangles f_l and f_r from the selective mesh.

3. Replace v_c by v_p in all adjusted triangles.

Notice that by our legality definition, the adjusted triangles for a vsplit/ecol operator are a fixed set. Since the modifications that a vsplit/ecol operator makes are

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

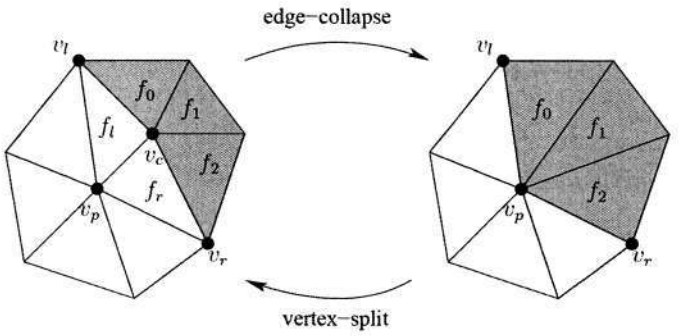


Figure 3.4: Our legality definition for vertex-split and edge-collapse.

predetermined, the content of the operator can be stored explicitly, so that no neighborhood queries are needed when updating the selective mesh. Figure 3.5 is the content of a vsplit/ecol operator. The index of the vertex v_c, which is split form v_p, is not stored, as it can be easily computed by:

$$v_c = i + base_ver_num \quad ,$$

where i is the index of the vertex-split in the vertex-split list of the PM and $base_ver_num$ is the number of the vertices in the base mesh of the PM. Also, the indices for the two collapsed triangles f_l and f_r are not stored. They can be computed by:

$$f_l = 2 \cdot i + base_tri_num \quad ,$$

$$f_r = 2 \cdot i + 1 + base_tri_num \quad ,$$

where $base_tri_num$ is the number of triangles in the base mesh of the PM. If either f_l or f_r does not exist due to the boundary condition, its corresponding pivot vertex v_l or v_r will be set as -1.

The legality definition creates the dependencies between the operators. If a vertex-split $vsplit_i$ is dependent on another vertex-split $vsplit_j$, $vsplit_j$ has to be performed before $vsplit_i$, in order to create certain vertices or triangles required by the legality of

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library


```
struct VSP {
    int v_p;                // index of the vertex to split v_p
    float v_c_coord[3];     // coordinates of the split vertex v_c
    float v_c_norm[3];      // normal of the split vertex v_c
    int v_l;                // left pivot vertex v_l
    int v_r;                // right pivot vertex v_r
    Array<int> adjusted_triangles;
}
```

Figure 3.5: The content in a vertex-split / edge-collapse operator.

$vsplit_i$. In the DAG hierarchy of vertex-splits, the dependency is represented by the node of $vsplit_j$ being the parent of the node of $vsplit_i$. On the other hand, if $vsplit_i$ is dependent on $vsplit_j$, then $ecol_j$ must be dependent on $ecol_i$, where $ecol_i$ and $ecol_j$ are the reverse edge-collapses corresponding to $vsplit_i$ and $vsplit_j$. The parent-child relationships for the edge-collapses are symmetric to that for the vertex-splits. So the DAG for the edge-collapses is the same as the DAG for the vertex-splits with all the directed edges reversed. Therefore, we need to consider only vertex-splits to build the DAG.

3.3 Construction of the DAG multiresolution hierarchy

In the DAG G of the vertex-splits we want to build, each node t contains a vertex-split operator ($t.vsplit$), a list of pointers to the parent nodes ($t.in_edges$) and a list of pointers to the child nodes ($t.out_edges$). Each node t is indexed by an ID ($t.id$), which is equal to the index of $t.vsplit$ in the vertex-split list of the PM. The DAG can be built incrementally by adding one node of vertex-split to it at a time in the order of the vertex-split list in the PM. This is because the final DAG should define a partial order of the vertex-split operators that is compatible with the total order of the vertex-split list in the PM. In other words, for a node t in G, all $a \in ancestors(t)$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

should have $a.id < t.id$, where $ancestors(t)$ is the set of nodes that are ancestors of t in G. Adding the vertex-split nodes to the DAG in the order of the vertex-split list in the PM guarantees that when t is added to G, all t's ancestors have already been built into G, so that t can be added to G by linking it to its parents.

Under our legality definition, each vertex-split operator has a dependency list as $(v_p, v_r, v_l, < adjusted_triangles >)$, which can be retrieved directly from the data structure for vertex-splits (refer to Figure 3.5). The most straightforward way of building the parent linkages for a node t is to find the creator of each element in the dependency list and link all these creators as its parents. This is trivial since for each vertex or each triangle we can record the vertex-split that has created it. Figure 3.6 shows an example of a series of vertex-split operators with their dependency lists and Figure 3.7 depicts the incremental building procedure of the DAG using this straightforward way.

We call this straightforward method *full-redundant* algorithm, since about 60% of the edges in the DAG built by this method are redundant, as observed from our experiments (refer to Section 3.5). In a simple DAG, an edge from one node to another is *redundant* if there already exits a path from the former to the latter. Notice that edge $t_1 \rightarrow t_4$, $t_1 \rightarrow t_5$ and $t_2 \rightarrow t_5$ are redundant in Figure 3.7. Redundant edges can cost considerable extra memory and impose unnecessary operations on the edges during the run-time traversal of the DAG. In order to optimize the DAG multiresolution hierarchy, we need to eliminate the redundant edges.

3.3.1 Non-redundant DAG

For each node t to be added to the DAG, the set of parents P determined by the full-redundant algorithm is actually the set of t's potential parents, including both its real parents and the redundant parents. To build the non-redundant DAG, all the redundant parents need to be eliminated from set P and only the real parents are linked to t.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

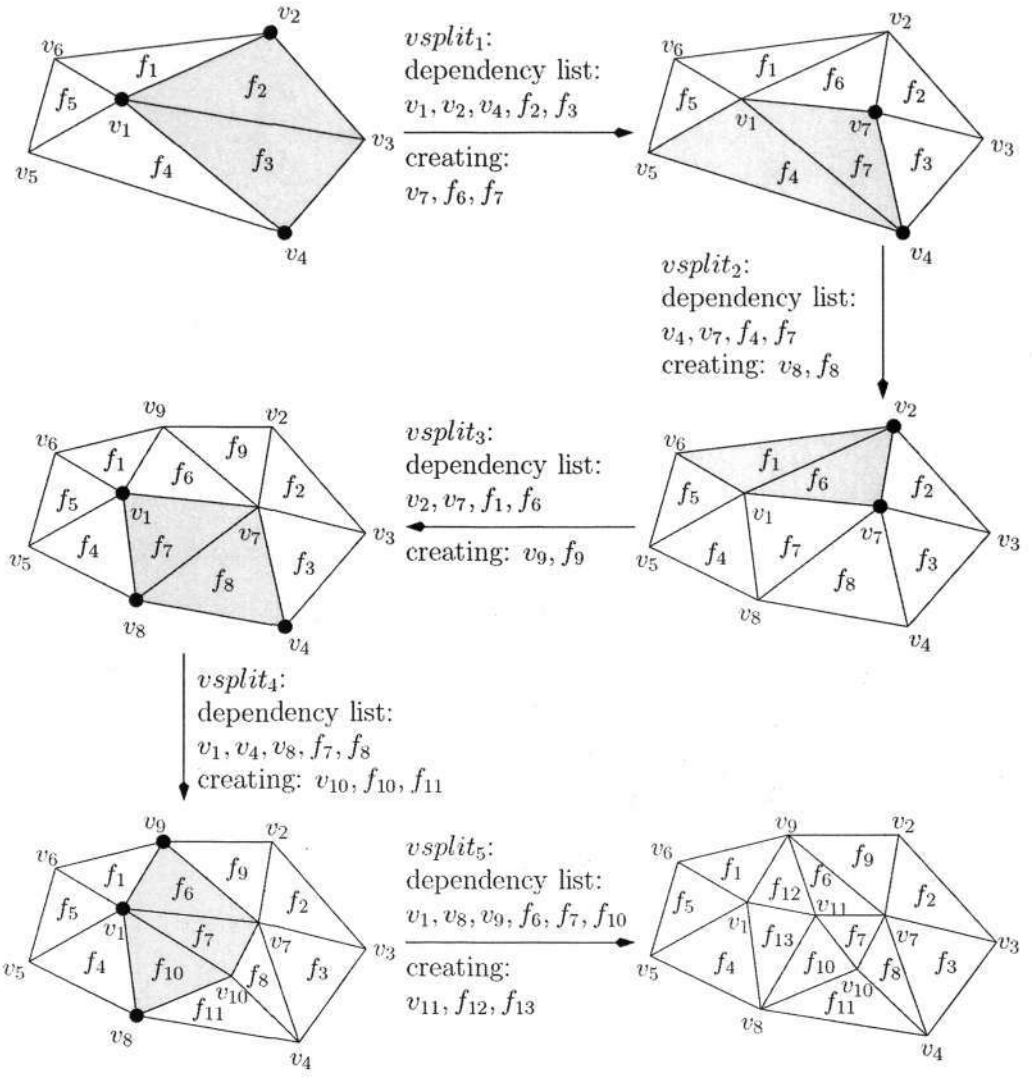


Figure 3.6: An example of a list of vertex-splits and their corresponding dependency lists.

Since all the potential parents in set P are already in the DAG, for any two nodes p_i and p_j, where $p_i \in P$, $p_j \in P$ and $p_i.id > p_j.id$, there are two possibilities:

1. Nodes p_i and p_j are independent of each other. In this case, both p_i and p_j are real parents of t (see Figure 3.8(a)).

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

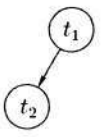
Adding $vsplit_1$:

Depend.	v_1	v_2	v_4	f_2	f_3
Creator	Null	Null	Null	Null	Null

(a)

Adding $vsplit_2$:

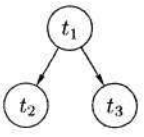
Depend.	v_4	v_7	f_4	f_7
Creator	Null	$vsplit_1$	Null	$vsplit_1$



(b)

Adding $vsplit_3$:

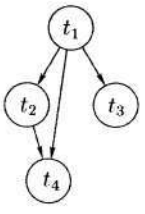
Depend.	v_2	v_7	f_1	f_6
Creator	Null	$vsplit_1$	Null	$vsplit_1$



(c)

Adding $vsplit_4$:

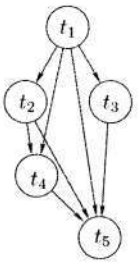
Depend.	v_1	v_4	v_8	f_7	f_8
Creator	Null	Null	$vsplit_2$	$vsplit_1$	$vsplit_2$



(d)

Adding $vsplit_5$:

Depend.	v_1	v_8	v_9	f_6	f_7	f_{10}
Creator	Null	$vsplit_2$	$vsplit_3$	$vsplit_1$	$vsplit_1$	$vsplit_4$



(e)

Figure 3.7: Building the DAG from the list of vertex-splits in Figure 3.6 incrementally, in the straightforward way. All the creators of the elements in a vertex-split's dependency list become that vertex-split's parents. For each node t_i, $t_i.vsplit = vsplit_i$.

2. Node p_j is an ancestor of node p_i. In this case, only p_i is a real parent of t, and p_j is a redundant parent (see Figure 3.8(b)).

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

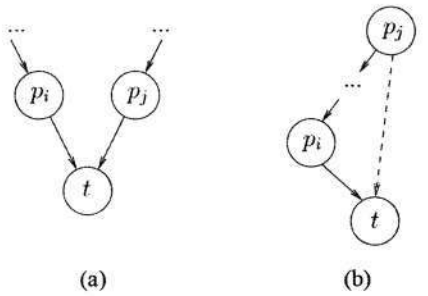


Figure 3.8: Non-redundant edges and redundant edges.

Suppose p_{max} is the potential parent with the maximum ID among P, i.e. $p_{max} = \text{argmax}_{p \in P}(p.id)$. Since p_{max} cannot be the ancestor of any other potential parent in P, it is always a real parent of t. Based on this idea, we can find all the real parents of t iteratively. In each iteration, we add the edge $p_{max} \rightarrow t$ to the DAG, and remove p_{max} and all the elements that are ancestors of p_{max} from set P. This procedure is performed iteratively until no element is left in set P. As this method can build the DAG hierarchy without any redundant edge, we call it *non-redundant* algorithm. Figure 3.9 describes adding a node t to the DAG G with this non-redundant algorithm, in which $potential(x)$ is the set of potential parents of node x and $ancestors(x)$ is the set of ancestors of node x in the DAG.

The worst-case time complexity of constructing the non-redundant DAG is $O(n^2)$, where n is the number of nodes, due to the $O(n)$ worst-case complexity of testing whether $p' \in ancestors(p_{max})$ is true in line 7 of Figure 3.9.

3.3.2 Mini-redundant DAG

The non-redundant algorithm with $O(n^2)$ complexity is not feasible for practical use. The feature of local influence of the vertex-split operators can be used to optimize the time complexity of the DAG construction algorithm.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library


```
NON-REDUNDANT(t, G)
P ← potentials(t)
for all p ∈ P do
    p_max ← argmax_{p∈P}(p.id)
    add edge p_max → t in G
    delete p_max from P
    for all p' ∈ P do
        if p' ∈ ancestors(p_max) then
            delete p' from P
        end if
    end for
end for
```

Figure 3.9: The non-redundant algorithm eliminates all redundant parents by searching in the ancestor sets.

The vertices and the triangles in the dependency list of a vertex-split are all direct neighbors to the vertex to be split. They are also direct neighbors to each other and hence their creators have high probabilities of being each other's potential parents. Therefore, most redundant parents of node t are its potential grandparents. So instead of finding out and eliminating all the redundant parents in the expensive way in Section 3.3.1, we can quickly eliminate the potential parents that are also the potential grandparents.

For two potential parent p_{max} and p' in P, and p_{max} is a real parent of t, we can detect whether p' is a potential grandparent by checking if $p' \in Q$, where Q is the potential parent set of p_{max}. We call this method *mini-redundant algorithm* (Figure 3.10), as it can eliminate most of the redundant parents but not all of them.

Finding the potential parent set of a node only takes $O(1)$ time, as it only involves collecting the creators of the elements in the dependency list. Thus, constructing the DAG hierarchy using the mini-redundant algorithm has a time complexity of $O(n)$, where n is the number of nodes.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library


```
MINI-REDUNDANT(t, G)
P ← potentials(t)
for all p ∈ P do
    p_max ← argmax_{p∈P}(p.id)
    add edge p_max → t in G
    delete p_max from P
    Q ← potentials(p_max)
    for all p' ∈ P do
        if p' ∈ Q then
            delete p' from P
        end if
    end for
end for
```

Figure 3.10: The mini-redundant algorithm eliminates most of the redundant parents by searching in the potential grandparent sets.

3.4 Construction of the visibility attributes

During the run-time, whether a node in the DAG hierarchy should split or collapse depends on certain view-dependent error criteria. Each node must store certain visibility attributes corresponding to the error criteria. These visibility attributes, together with the view-parameters will determine the run-time view-dependent error for a node.

In our work, we adopt three previously proposed view-dependent error criteria [63, 98]: *out-of-view*, *back-face* and *screen-space error*. With each node in the DAG, three scalar values, *rad* (Section 3.4.1), *cone_sin* (Section 3.4.2) and *ap_err* (Section 3.4.3) are stored for run-time evaluation of the three error criteria. The visibility attributes of the nodes are constructed hierarchically after the parent-child relationships among the vertex-splits are built in the DAG.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

3.4.1 Out-of-view

The purpose of this criterion is to simplify the parts of the model that are outside of the view-frustum in order to reduce graphics load, since these parts are actually invisible to the viewer. Bounding sphere is the most efficient way for conservative out-of-view checking. The method presented by Hoppe [63] is simple and accurate, however it requires to extract the 16 coefficients for the four bounding planes of the view-frustum from the rendering subsystem every time the view changes. Instead, we follow the method presented by Pajarola [98], which approximates the 4-sided view-frustum with a cone as $view_cone(e, n, \alpha)$, where e is the eye position, n is the viewing direction and α is the semi-angle of the cone. This representation is also more convenient for our view-parameters predication strategy described in Chapter 5. As shown in Figure 3.11, the out-of-view of a bounding sphere can be quickly determined by whether $(\gamma - \varphi) > \alpha$.

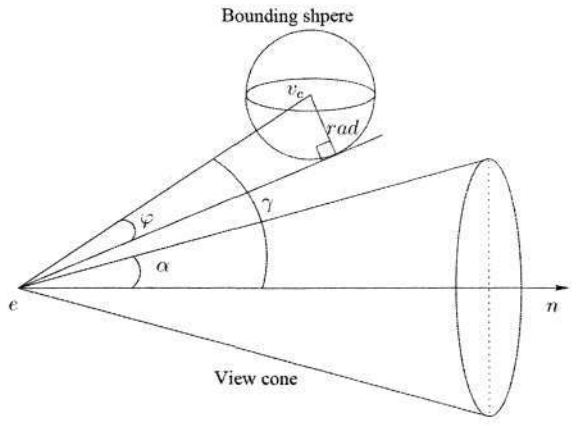


Figure 3.11: Out-of-view test.

More detailed mathematical transformations are described in [98].

To improve the space efficiency, we center the bounding sphere of each vertex-split VSP_i at $VSP_i.v_c$ (refer to Section 3.2 for the data structure of VSP for vertex-splits). Thus for each VSP_i, we only construct and store the radius rad of its bounding sphere

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

that bounds the region affected by VSP_i and all its descendants. First the bounding sphere radius for each vertex-split is initialized as:

$$rad = \max(|v_c - v_p|, |v_c - v_r|, |v_c - v_l|) \quad .$$

Then, in a bottom-up manner, for each vertex-split VSP_i, its bounding sphere radius $VSP_i.rad$ is constructed by sequentially combining each $VSP_d.rad$, where VSP_d is a child of VSP_i (refer to Figure 3.12),

$$VSP_i.rad = \max(VSP_i.rad, |VSP_i.v_c - VSP_d.v_c| + VSP_d.rad) \quad .$$

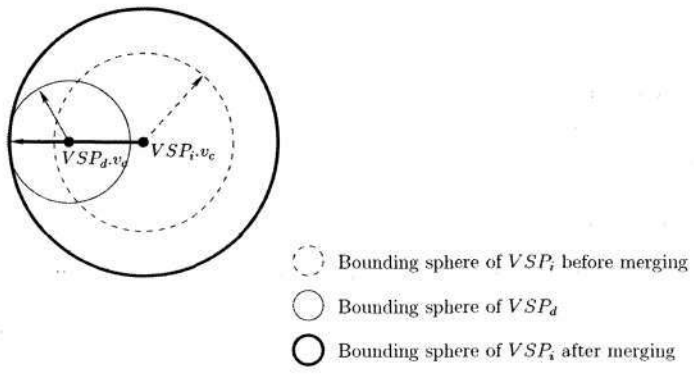


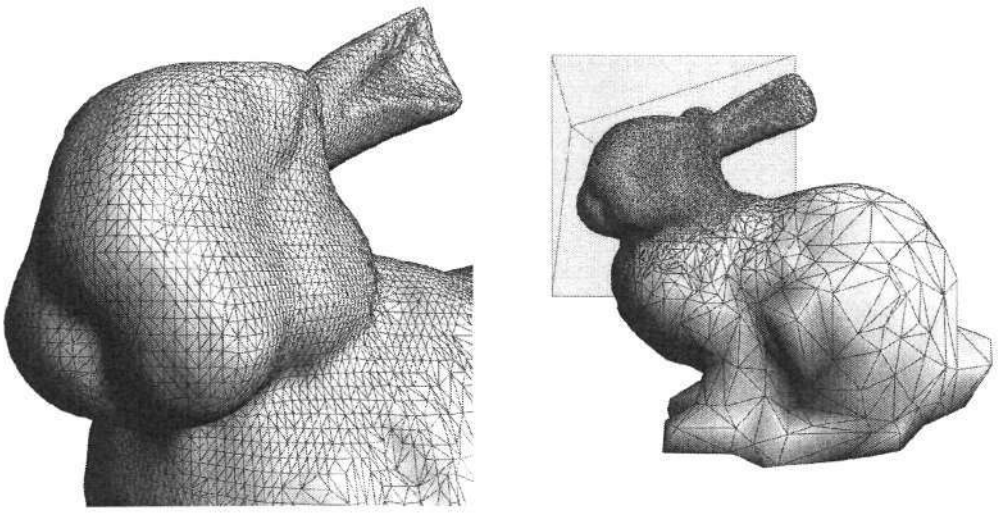
Figure 3.12: Merging the bounding sphere of child VSP_d into the bounding sphere of parent VSP_i.

The effect of out-of-view simplification is demonstrated in Figure 3.13 as an example.

3.4.2 Back-facing

This criterion aims to coarsen the parts of the model oriented away from the viewer. Like the out-of-view test, a bounding volume is necessary to fast detect whether the region affected by the vertex-split is back-facing. The cone of normals [116, 63, 87, 98] is used to serve this purpose. The cone of normals of a vertex-split is defined by a

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



(a) (b)

Figure 3.13: An example of the out-of-view simplification. (a)The Bunny viewed from the actual viewpoint. (b)The side view of the Bunny with the view-frustum.

cone axis a and a semi-angle θ, where $\theta \leq \pi/2$, about the axis. It bounds the space of normals associated with the region affected by the vertex-split and all its descendants. As shown in Figure 3.14, the cone of normals is oriented away from the viewer if the direction of sight lies in the back-facing region of the cone in the normal space, that is, if

$$\gamma < \pi/2 - \theta \;\Rightarrow \cos(\gamma) > \sin(\theta) \quad . \tag{3.4.1}$$

Again, to improve the space efficiency, for each vertex-split VSP_i, $VSP_i.v_c$ is used as the anchor point [116] of the cone and the normal of $VSP_i.v_c$ is used as the cone axis a. Therefore, Equation 3.4.1 can be rewritten to:

$$\frac{v_c - e}{|v_c - e|} \cdot a > \sin(\theta) \quad ,$$

where e is the eye position. So for each vertex-split, only the semi-angle θ is enough to store the information of the cone of normals.

The semi-angle θ of the cone of normals is computed in a similar way as computing

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

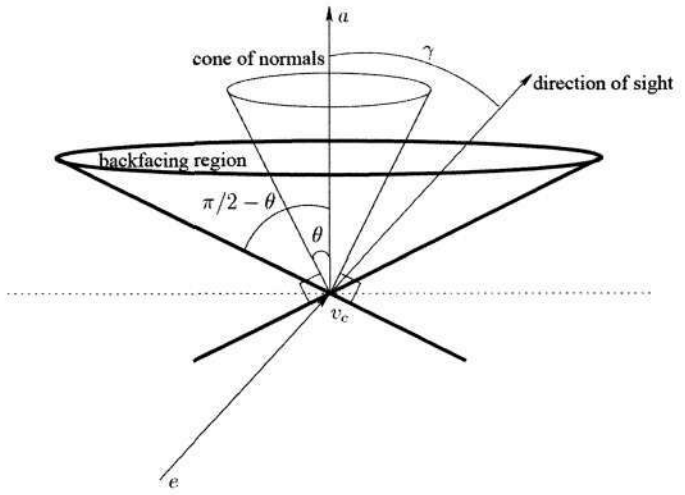


Figure 3.14: Back-face test.

the bounding sphere radius. First for each vertex-split, θ is initialized as

$$\theta = \min(\pi/2, \max(\delta_1, \delta_2)) \quad ,$$

where δ_1 is the angle between the normals of v_c and f_l, and δ_2 is the angle between the normals of v_c and f_r. Then also in a bottom-up manner, for each vertex-split VSP_i, the semi-angle $VSP_i.\theta$ is constructed by sequentially combining each $VSP_d.\theta$, where VSP_d is a child of VSP_i (refer to Figure 3.15):

$$VSP_i.\theta = \max(VSP_i.\theta, \min(\pi/2, \delta + VSP_d.\theta)) \quad ,$$

where δ is the angle between the two cone axes of VSP_i and VSP_d. Like in [98], in implementation we store the value $\sin(\theta)$ instead of θ with each node as $VSP_i.cone_sin$, since only $\sin(\theta)$ is needed for back-face checking.

The effect of back-face simplification is demonstrated in Figure 3.16 as an example.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

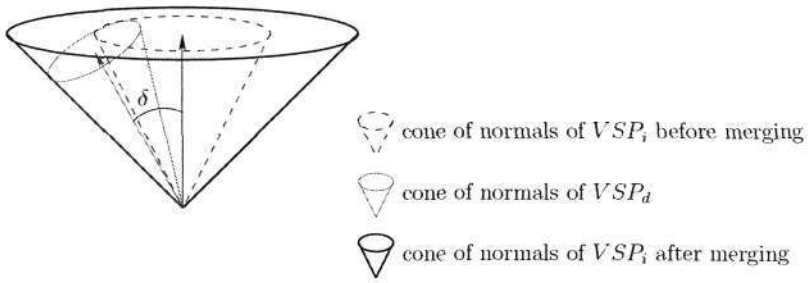
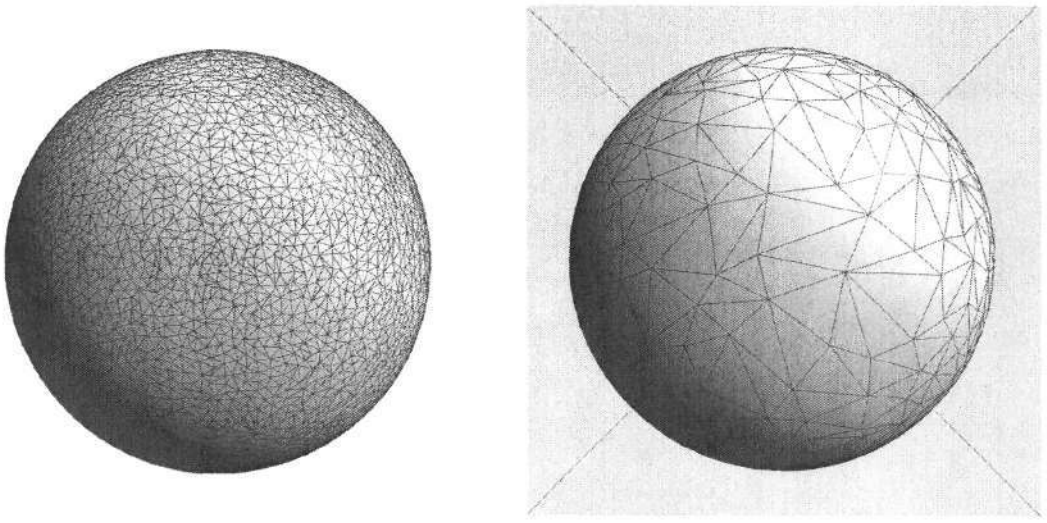


Figure 3.15: Merging the cone of normals of child VSP_d into the cone of normals of parent VSP_i.



(a) (b)

Figure 3.16: An example of back-face simplification. (a)The Globe viewed from the actual viewpoint. (b)The back view of the same Globe.

3.4.3 Screen-space error

This criterion tries to prevent unnecessary refinements on the parts of the model where the projected errors on the screen are already below certain specified threshold. This error criterion can be measured in many different ways [63, 87, 98]. In our work, the screen-space error of a vertex-split VSP_i is measured by the projected size of its approximation error $VSP_i.ap_err$, which represents the error from replacing $VSP_i.v_c$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

by $VSP_i.v_p$ on the model. Since we use the quadric error (QE) metric [46] in the mesh simplification stage to produce the PM of the model, we store the QE of each edge-collapse during the simplification, and use it to initialize the approximation error *ap_err* of the corresponding vertex-split. After the DAG of the vertex-splits is built, the *ap_err* of each vertex-split VSP_i is assigned, again in a bottom-up manner as follows,

$$VSP_i.ap_err = \max(VSP_i.ap_err, \max_{\forall VSP_d}(VSP_d.ap_err)) \quad ,$$

where VSP_d is a child of VSP_i.

The screen-space error threshold τ is defined as the fraction of the viewport size [63]. To test whether the screen-space error of a vertex-split is large enough for performing the split operation, we adopt the formula presented in [88]. The vertex-split VSP_i does not need to be split if

$$VSP_i.ap_err < 2\tau \times \tan(\alpha) \times d \quad ,$$

where α is the semi-angle of the view cone and d is the world-space distance from the eye position e to $VSP_i.v_c$ along the viewing normal n, i.e.,

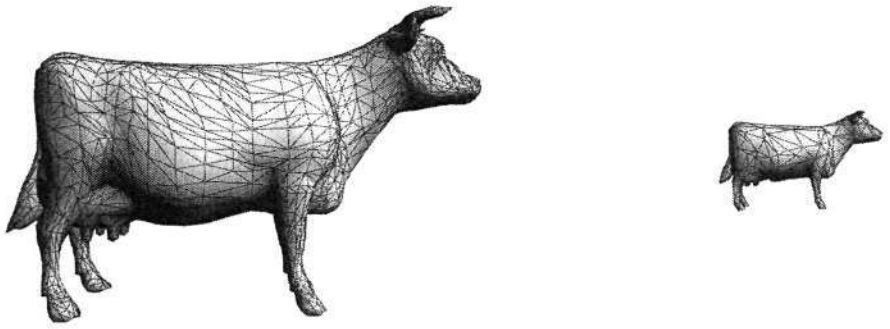
$$d = (VSP_i.v_c - e) \cdot n \quad .$$

The effect of screen-space error control is shown in Figure 3.17 as an example.

3.5 Results

We test our mini-redundant algorithm of constructing the DAG hierarchy, compared with the full-redundant algorithm and the non-redundant algorithm, on a variety of models with sizes ranging from thousands of triangles to more than one million triangles. The details of the models are listed in Table 3.1. These models are also used in

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



(a) (b)

Figure 3.17: An example of screen-space error control. The Cow model is rendered with $\tau = 0.0001$. (a)The model is refined to 3554 triangles when the viewpoint is near. (b)The model is simplified to 630 triangles when the viewpoint is far.

Table 3.1: The models used throughout the thesis.

	Cow	Bunny	Globe	Hand	Dragon	Happy
Vertices	2903	34834	36866	327323	435545	543524
Triangles	5804	69451	73728	654666	871306	1087474

Chapter 4, Chapter 5 and Chapter 6. The tests reported in this section are performed on a PC with a Pentium4 3.6GHz CPU and 512MB memory.

The time costs for DAG construction by the full-redundant algorithm, the non-redundant algorithm and the mini-redundant algorithm are listed in Table 3.2 for each model. As the results show, the full-redundant algorithm takes the least time and the mini-redundant algorithm takes about twice the time of the full-redundant algorithm. The results of both algorithms confirm the theoretically $O(n)$ time complexity. Although the non-redundant algorithm can eliminate all the redundant edges of the DAG, it is magnitudes slower than the other two algorithms.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Table 3.2: The construction time costs (in seconds) of the full-redundant DAG, the non-redundant DAG and the mini-redundant DAG, for each model.

	Cow	Bunny	Globe	Hand	Dragon	Happy
Full. (sec.)	0.009	0.224	0.186	2.430	3.474	4.264
Non. (sec.)	0.118	28.692	33.299	358.525	381.397	382.334
Mini. (sec.)	0.018	0.296	0.262	4.640	6.614	8.296

In Table 3.3, the total numbers of incoming edges in the DAGs built by the full-redundant algorithm, the non-redundant algorithm and the mini-redundant algorithm are compared. The efficiency of the mini-redundant algorithm is measured by the percentage of all redundant edges it can eliminate. This table demonstrates two important facts. First, the number of incoming edges in the DAG built by the full-redundant algorithm is about three times the number by the non-redundant algorithm. In other words, more than 60% of the edges in the DAG built by the full-redundant algorithm are redundant. Second, the mini-redundant algorithm can eliminate more than 80% of the redundant edges, while keeping the $O(n)$ time complexity.

Table 3.3: The total numbers of incoming edges in the full-redundant DAG, the non-redundant DAG and the mini-redundant DAG, for each model, and the efficiency and memory cost of the mini-redundant DAG.

	Cow	Bunny	Globe	Hand	Dragon	Happy
Full.	13421	171122	179559	1579610	2038130	2538069
Non.	5055	62891	68158	584449	759603	940533
Mini.	6437	80882	85956	750627	970208	1206504
Effi. of Mini.	83.5%	83.4%	84.0%	83.3%	83.5%	83.4%
Mem. of Mini.	242K	2976K	3092K	28251K	36557K	45236K

The unoptimized data structure for a DAG node is shown in Figure 3.18. The members `rad`, `cone_sin` and `ap_err` are the visibility attributes for view-dependent error evaluation and are computed for each node after the DAG hierarchy is constructed

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

(Section 3.4). The structure of VSP is defined in Section 3.2.

```
struct DAG_node {
    VSP vsplit;                 // vertex-split operator
    float rad;                  // radius of bounding sphere
    float cone_sin;             // sine of normal cone semi-angle
    float ap_err;               // error of the vertex-split
    Array<int> in_edges;        // incoming edges
    Array<int> out_edges;       // outcoming edges
}
```

Figure 3.18: The data structure for a DAG node.

In the full-redundant DAG, each node has 5 incoming edges and 5 outcoming edges on average. In the mini-redundant DAG, each node has 2.5 incoming edges and 2.5 outcoming edges on average. Thus, using the mini-redundant algorithm can save about 20 bytes per node, or about 20% memory in the finally DAG. As shown in Table 3.3, the memory cost of the mini-redundant DAG is about 84 bytes per vertex on average, comparable to the memory-efficient view-dependence tree in [37] and the FastMesh in [98]. As a result, the mini-redundant algorithm is the most practical algorithm to build the DAG hierarchy of vertex-splits from a PM representation.

3.6 Summary

A DAG multiresolution hierarchy can encode all the dependency relationships among the vertex-split operators, and thus makes the multiresolution hierarchy traversal independent of the selective mesh. A properly defined legality of dependencies for the vertex-split operators can guarantee unique modifications on the selective mesh. Most of the previously proposed legality definitions for the dependency relationships are too loose to provide predetermined modification procedures for the vsplit/ecol operators. Although the legality definition by Xia *et al.* [137] and El-Sana *et al.* [37] ensures a

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

unique surrounding neighborhood for a vsplit/ecol operator, it is overly restricted. We propose our own legality definition that is more flexible than the one in [137] and [37] and at the same time can provide predetermined modification procedures for the vsplit/ecol operators. The straightforward way of building the DAG according to the legality dependencies results in a large number of redundant edges. We manage to devise an algorithm that can eliminate all the redundant edges, however its high time complexity is unfavorable to practical use. We then present a practical algorithm that can reduce most of the redundant edges in linear time, by utilizing the locality of the vertex-split and edge-collapse operators. The time-efficiency and space-efficiency of the mini-redundant DAG construction algorithm are demonstrated through a series of experiments on various models. Visibility attributes for each node in the DAG for run-time view-dependent rendering are constructed hierarchically after the DAG is built.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Chapter 4

DAG Hierarchy Traversal and View-dependent Rendering

4.1 Introduction

As we mentioned in Chapter 3, the *front* nodes are the boundary nodes between the split nodes and the collapsed nodes in the multiresolution hierarchy. Multiresolution hierarchy traversal is the process of visiting the dynamic set of front nodes and selecting the appropriate refinement/simplification operators from them according to the current view and certain error evaluation criteria to update the selective mesh. The traversal algorithm is critical to the view-dependent rendering system, because it affects both the frame time and the view-dependent rendering result. Usually within a frame, the traversal takes a considerable portion of time and reducing this part of time is very helpful to increase the frame rate.

In this chapter, we study the multiresolution hierarchy traversal and the view-dependent rendering of the DAG hierarchy of vertex-splits built in Chapter 3. We especially focus on the traversal problems faced by the interactive client-server based view-dependent rendering, where the server stores the multiresolution hierarchy and performs the traversal, and the client maintains and renders the selective mesh. Not

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

only the hierarchy traversal time on the server should be considered, but also the client-side triangle budget. However, in such systems, the server cannot easily get the information of the selective mesh at each step during the hierarchy traversal. A traversal management strategy is proposed to address the problems. Instead of making the server consult the client for the information of the selective mesh during the traversal, we let the client decide certain control parameters according to its current condition and send them to the server to control the traversal before it starts.

We also present a comprehensive analysis and comparison of three different traversal algorithms that can be used under the traversal management strategy. Two of them, the *linear traversal* and the *priority-oriented traversal*, have been used in previous works, and we adapt them to our traversal management strategy. We also propose a third traversal algorithm, the *constant-time traversal* that can provide the minimum and almost constant traversal time. Extensive experiments are carried out to compare the performances of the three algorithms in terms of the traversal time and the accuracy of the view-dependent refinement/simplification.

Another important issue in run-time view-dependent LOD rendering is the representation of the selective mesh and its modification. We present a compact representation of the selective mesh based on OpenGL vertex arrays [135], which is memory efficient and optimal for rendering. This representation also supports fast mesh modification and is independent of the multiresolution hierarchy during mesh modification and rendering.

The rest of the chapter is organized as follows. Section 4.2 describes the maintenance of the dynamic set of the front nodes during the traversal. Section 4.3 presents the traversal constraints and introduces the traversal management strategy. In Section 4.4 we compare the three different traversal algorithms and Section 4.5 describes the efficient representation of selective mesh on the client side. Section 4.6 discusses the results of the comparative experiments for the three traversal algorithms and Section 4.7

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

summarizes the chapter.

4.2 Maintenance of the fronts

4.2.1 Representations of the fronts

The core structure of view-dependent LOD is the multiresolution hierarchy, where each node can be either in *split* status or in *collapsed* status. Splitting a node means performing a refinement operation on the selective mesh and collapsing a node means performing the reverse simplification operation on the selective mesh. At any moment of run-time, the multiresolution mesh hierarchy is partitioned by a *cut* [31]. A cut of a connected graph is a set of edges which, if removed, disconnect the graph. The cut partitions a multiresolution mesh hierarchy G into two contiguous parts, the sub-graph G_s above the cut and the sub-graph G_c below the cut (see Figure 4.1). G_s contains all the nodes that are in split status and G_c contains all the nodes that are in collapsed status. Usually, the cut is represented by the *front* nodes, which are the boundary nodes between G_s and G_c. Incremental view-dependent refinement is accomplished by moving the cut up and down on the multiresolution mesh hierarchy. Different types of multiresolution mesh hierarchies have different representations of front for the cut, and different methods to incrementally move the cut and maintain the front nodes. Generally speaking, the multiresolution mesh hierarchy can be a hierarchy of vertices [87, 137, 63], often called *vertex forest* or *vertex trees*, or a hierarchy of mesh-update operators [97, 58, 31].

In a hierarchy of vertices, the siblings always exhibit the same status; either all are split or all are collapsed (see Figure 4.1(a)). The *front* of a vertex forest [136, 63] is defined as the set of split nodes with no split children, and hence the front nodes are the active vertices on a particular LOD. Each of them has both the possibility to further split into its children and the possibility to collapse to its parent for the next frame.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

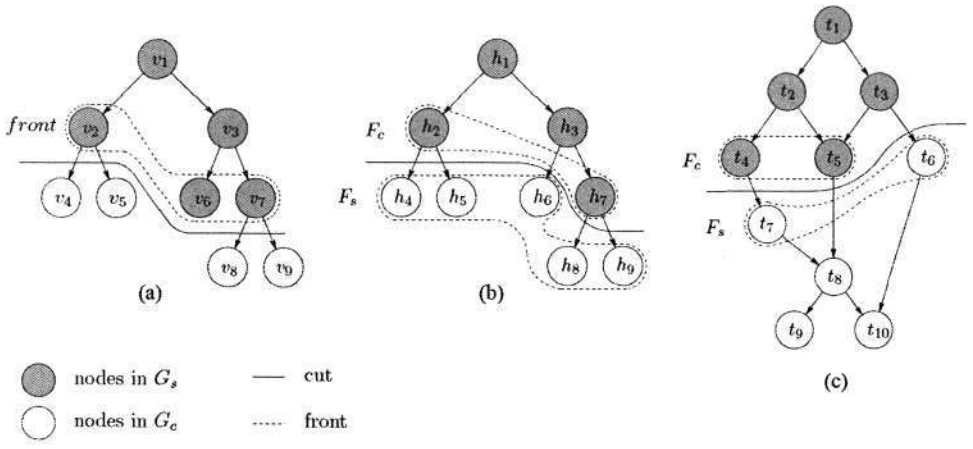


Figure 4.1: Incremental traversal on three different hierarchies. (a) On the binary vertex forest. (b) On the binary forest of vertex-splits. (c) On the DAG of vertex-splits.

In a hierarchy of mesh-update operators (Figure 4.1(b) and Figure 4.1(c)), the siblings may not show the same status. The nodes that are ready to split and the nodes that are ready to collapse are two separate sets, which are the leaf nodes of G_s, denoted by F_c as the front of nodes for collapse operations, and the root nodes of G_c, denoted by F_s as the front of nodes for split operations. The hierarchy G of mesh-update operators can be a binary forest as in [97, 98] (Figure 4.1(b)) or a DAG as in this thesis (Figure 4.1(c)). In the binary forest, only F_c is necessary to be maintained, since we can always get the set F_s from F_c by $F_s = CHILDREN(F_c) \cup C_SIBLING(F_c)$, where $CHILDREN(F_c)$ is the set of child nodes of the nodes in F_c and $C_SIBLING(F_c)$ is the set of sibling nodes with collapsed status of the nodes in F_c (refer to Figure 4.1(b)). However, this relationship between F_c and F_s is not true for a DAG hierarchy, since each node in a DAG may have multiple parents. Therefore, in our DAG-based multiresolution hierarchy of vertex-split operators, we explicitly maintain the two fronts F_c and F_s during the run-time incremental view-dependent rendering (see Figure 4.1(c)).

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

4.2.2 Maintaining the fronts in the DAG hierarchy

A node in the DAG is legal to split only when all its parents are already in split status; a node is legal to collapse only when all its children are already in collapsed status. Front F_s contains all the collapsed nodes of which all the parents are split. Front F_c contains all the split nodes of which all the children are collapsed. Only the front nodes in F_s are legal to split and only the front nodes in F_c are legal to collapse. The two fronts are dynamically maintained during the multiresolution hierarchy traversal. During run-time view-dependent rendering, front F_s is traversed for vertex-split operators and front F_c is traversed for edge-collapse operators. For each visited front node, view-dependent error evaluation is performed to decide whether the status of the node should be split or collapsed (see Figure 4.2). The view-dependent error criteria have been discussed in Chapter 3.

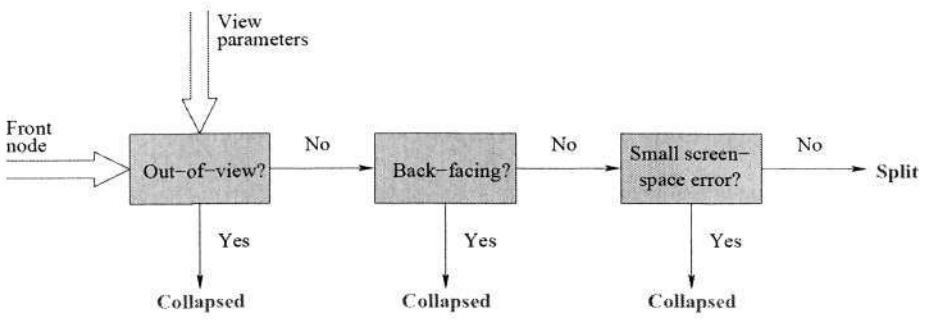


Figure 4.2: The procedure of view-dependent error evaluation for a front node.

In order to dynamically maintain the fronts of the DAG G during the traversal, each node t of G keeps a counter $t.counter$. The counter $t.counter$ indicates the number of dependencies that have to be released before t is legal to split or to collapse. If t is in collapsed status, $t.counter$ is the number of constraints for it to be legal to split, i.e. the number of t's collapsed parents. If t is in split status, $t.counter$ is the number of constraints for it to be legal to collapse, i.e. the number of t's split children. The front

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

nodes that are legal to split or collapse are identified by their counters being zero.

Initially, the list F_c is empty and the list F_s contains all the root nodes of G. The initial counter value of a node is the number of its incoming edges, that is, $t.counter = |t.in_edges|$. When a node splits or collapses, the relevant counters will be modified to reflect the new fronts. Figure 4.3 and Figure 4.4 show the maintenance procedures when a node t splits or collapses respectively.

```
SPLIT-MAINTENANCE(t)
delete t from F_s
add t to F_c
for all child c of t do
    c.counter − −
    if c.counter = 0 then
        add c to F_s
    end if
end for
for all parent p of t do
    p.counter + +
end for
```

Figure 4.3: The maintenance procedure when a node t splits.

```
COLLAPSE-MAINTENANCE(t)
delete t from F_c
add t to F_s
for all parent p of t do
    p.counter − −
    if p.counter = 0 then
        add p to F_c
    end if
end for
for all child c of t do
    c.counter + +
end for
```

Figure 4.4: The maintenance procedure when a node t collapses.

In the procedure of SPLIT-MAINTENANCE, when a front node t in F_s splits, it is

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

moved from the front F_s to the front F_c. The constraints of t's children are reduced by one and the constraints of t's parents are increased by one. If a child c of t has *c.counter* equal to 0, c becomes a new front node in F_s. If a parent p of t is in F_c before t is moved, p will no longer be a legal front node after t is added to F_c. The traversal algorithm can detect this because *p.counter* will not be 0. Figure 4.5 shows an example of node t_7 splitting. The procedure COLLAPSE-MAINTENANCE is similar and symmetric to the procedure SPLIT-MAINTENANCE.

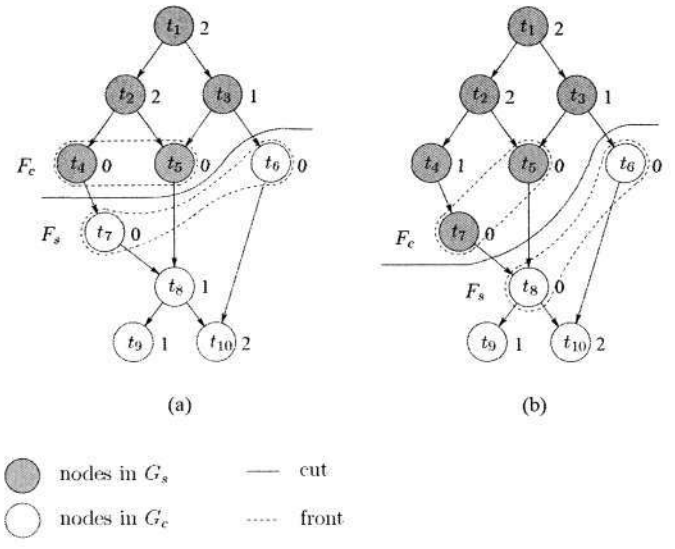


Figure 4.5: An example of maintenance of the front F_c and the front F_s. The counters are labelled beside the nodes. (a)Before node t_7 splits. (b)After node t_7 splits.

4.3 Traversal constraints and management

In the literature of view-dependent LOD, multiresolution hierarchy traversal is performed under either the fidelity constraint or the triangle-budget constraint [63, 87]. Under the fidelity constraint, all nodes with view-dependent fidelity errors larger than a predefined threshold should split [63, 136]. Under the triangle-budget constraint, the goal is to keep a constant number of triangles and the traversal stops when further

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

refinement operations would exceed the triangle budget [87, 132]. However, sometimes we need to consider both the fidelity constraint and the triangle-budget constraint. More specifically, if the selective mesh that already satisfies a given fidelity constraint is still smaller than the triangle-budget, we do not need to perform further unnecessary refinement operations to reach the triangle-budget. On the other hand, the selective mesh cannot exceed the triangle budget even if the given fidelity constraint cannot be satisfied within the triangle budget.

Besides the fidelity constraint and the triangle-budget constraint, we find that the constraint on the amount of mesh changes is also very important. The amount of mesh changes can vary greatly from frame to frame, depending on the change of the view. This leads to a high variability in frame time, since the traversal time, the mesh updating time and the network transmission time for a client-server system are closely correlated to the amount of changes per frame. So the large amount of mesh changes for one frame should be amortized over several consecutive frames for frame-rate regulation [63]. Also the predictable and controllable amount of mesh changes for each frame is a useful hint for triangle-budget control. An efficient way of controlling the amount of mesh changes per frame is to specify the maximum number of mesh-update operators and stop the traversal when the maximum number of vertex-splits or edge-collapses have been generated.

In Figure 4.6 we construct a generic traversal management strategy that considers the fidelity constraint defined by the view-parameters (vp), the triangle-budget constraint controlled by the traversal mode ($mode$) and the mesh-change constraint defined by the maximum number of mesh-updates (N) for both vertex-splits and edge-collapses. A presumption here is that all the mesh-update operators are manifold, which means that all vertex-splits add two new triangles to the mesh and all edge-collapses delete two triangles from the mesh.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

MANAGE-TRAVERSAL($vp, mode, N$)
1: $num_collapsed \leftarrow$ TRAVERSE-COLLAPSE(vp, N)
2: **if** $mode$ =BUDGET-MODE **then**
3: TRAVERSE-SPLIT($vp, num_collapsed$)
4: **else if** $mode$ = FREE-MODE **then**
5: TRAVERSE-SPLIT(vp, N)
6: **end if**

Figure 4.6: Management of the DAG hierarchy traversal.

First, function TRAVERSE-COLLAPSE traverses the front F_c to select edge-collapses. Then TRAVERSE-SPLIT traverses F_s for vertex-splits. Both functions are controlled by the fidelity constraint and the mesh-change constraint. When the traversal mode is BUDGET-MODE, the triangle-budget control is applied, by setting the mesh-change constraint for TRAVERSE-SPLIT to the number of actual edge-collapses (*num_collapsed* in line 1) generated by TRAVERSE-COLLAPSE, because the number of triangles deleted by the edge-collapses determines the amount of room to accommodate the new triangles introduced by the vertex-splits.

This strategy is especially useful for the client-server based system, where the hierarchy traversal on the server cannot access the information of the selective mesh on the client. It gives the client the capability to control the traversal on the server. The arguments N and *mode* for the function MANAGE-TRAVERSAL are set by the client according to the current condition of its selective mesh and sent to the server with the view-parameters vp to instruct the server to perform the controlled traversal. The FREE-MODE is used when there is no budget control or the size of the selective mesh is below the triangle-budget; the BUDGET-MODE is used if the size of the selective mesh is likely to exceed the triangle budget. The client knows when to set the traversal mode to BUDGET-MODE or FREE-MODE since the maximum number of possible new triangles is predicable to the client due to the mesh-change constraint. In this manner, the traversal on the server can comply with the triangle-budget of the client without

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

knowing the detailed information of the selective mesh.

Even if not all the vertex-splits or edge-collapses are manifold operators, which means that equal number of edge-collapses and vertex-splits do not necessarily delete and add equal number of triangles, we can still use a similar method as in Figure 4.6 for triangle-budget control. Just change the constraint on the number of mesh-updates to the constraint directly on the number of triangles to be deleted or added by the mesh-updates. Thus, the triangles that can be added by function TRAVERSE-SPLIT will be restricted by the number of triangles that are deleted in TRAVERSE-COLLAPSE.

Functions TRAVERSE-COLLAPSE and TRAVERSE-SPLIT can be implemented in different traversal algorithms, which are discussed in the next section.

4.4 Comparison of three traversal algorithms

Conventionally, the linear traversal is used in fidelity-based view-dependent LOD [136, 63, 37], while the budget-based view-dependent LOD depends on the priority-oriented traversal [87, 30, 132]. The traversal management strategy in Figure 4.6 provides an alternative way for triangle-budget control with the flexibility of using different traversal algorithms. In this section, we first adapt the linear traversal and the priority-oriented traversal to work under the traversal management strategy, and then we propose a third traversal algorithm with the motivation of providing small and constant traversal cost for the time-critical rendering. All three algorithms perform with the fidelity constraint and the mesh-change constraint, defined by the view-parameters vp and the maximum number of mesh-updates N.

The traversal algorithms detailed in the following subsections are applicable to both the simplification traversal and the refinement traversal, namely function TRAVERSE-COLLAPSE and TRAVERSE-SPLIT in Section 4.3. The same function notations are used

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

in the following subsections. Function ERROR-EVALUATE is a combined representation for fidelity evaluation of various error criteria, which can include out-of-view, back-face, screen-space error, etc (refer to Section 3.4). Function NEED-APPLY determines if the node needs to split or collapse by comparing the evaluated error of the node with the error threshold of the fidelity constraint. Function APPLY-OP performs the application-dependent action on the selected node. It might be applying the vertex-split or edge-collapse to the selective mesh, or packing the operator for network transmission. Function ADJUST-FRONTS maintains the two fronts on the DAG hierarchy after each node splits or collapses, as described in Section 4.2.

4.4.1 Priority-oriented traversal

In the priority-oriented traversal, all the front nodes have to be first evaluated by ERROR-EVALUATE and inserted into a priority queue sorted by their evaluated errors. The priorities of F_c are in increasing order so that the node with the smallest estimated error can collapse first. The priorities of F_s are in decreasing order so that the node with the largest estimated error can split first. The traversal of the priority queue stops when the specified number of mesh-updates are generated or the first node that does not need to split or collapse according to the fidelity constraint is met. The new front nodes introduced during the traversal also need to be evaluated and inserted into the priority queue. The priority-oriented traversal is summarized in Figure 4.7.

This algorithm can produce good view-dependent refinement/simplification results, because the vertex-splits and edge-collapses applied to the mesh are the most appropriate ones selected from all the front nodes according to the current fidelity errors. However, due to the cost for all the front nodes to go through ERROR-EVALUATE and the cost for rebuilding and maintaining the priority queue, the priority-oriented traversal is rather expensive. The time complexity for this algorithm is $O(n)$, where n is the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library


```
PRIORITY-ORIENTED-TRAVERSAL(vp, N)
for all front node node do
    ERROR-EVALUATE(node, vp)
    Q_PRIO.insert(node)
end for
num ← 0
while Q_PRIO is not empty and num < N do
    node ← Q_PRIO.pop()
    if NEED-APPLY(node) then
        num ← num + 1
        APPLY-OP(node)
        new_front_nodes ← ADJUST-FRONTS(node)
        for all new front node new_node do
            ERROR-EVALUATE(new_node, vp)
            Q_PRIO.insert(new_node)
        end for
    else
        break
    end if
end while
```

Figure 4.7: Priority-oriented traversal algorithm.

number of front nodes.

4.4.2 Linear traversal

The linear traversal algorithm keeps the fronts F_c and F_s as two FIFO queues. The front nodes are evaluated for collapse or split in the FIFO order. The traversal continues until the specified number of mesh-update operators have been generated or no further mesh-update operators are possible according to the current view-parameters and the fidelity constraint. The new front nodes introduced during the traversal are appended to the end of the queue. The statuses of the FIFO queues are retainable from one frame to the next, therefore the traversal for the next frame can continue traversing the front from where the last traversal left. In this way, all the front nodes have the opportunity to be visited. The linear traversal algorithm is detailed in Figure 4.8.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library


```
LINEAR-TRAVERSAL(vp, N)
 1: num ← 0
 2: while Q_FIFO is not empty and num < N do
 3:    node ← Q_FIFO.pop()
 4:    ERROR-EVALUATE(node, vp)
 5:    if NEED-APPLY(node) then
 6:       num ← num + 1
 7:       APPLY-OP(node)
 8:       new_front_nodes ← ADJUST-FRONTS(node)
 9:       Q_FIFO.push_back(new_front_nodes)
10:    end if
11:    if no split or collapse in current front then
12:       break
13:    end if
14: end while
```

Figure 4.8: Linear traversal algorithm.

If all the front nodes have been traversed and none of them can split or collapse according to the fidelity constraint, the traversal can stop since no further split or collapse operations are possible for the current view-parameters. This is checked in line 11 of Figure 4.8. The worst time complexity of the linear traversal is also $O(n)$, where n is the number of front nodes.

4.4.3 Constant-time traversal

Like the linear traversal, the constant-time traversal algorithm also keeps the fronts F_c and F_s as two FIFO queues, and the statuses of the queues are retainable from one traversal to the next. However, instead of measuring the number of mesh-updates selected, constant-time traversal is controlled by the number of nodes visited. The goal is to achieve an almost constant traversal time by controlling the times that the function ERROR-EVALUATE is performed. Figure 4.9 shows the constant-time traversal algorithm.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library


```
CONSTANT-TIME-TRAVERSAL(vp, N)
 1: num ← 0
 2: while Q_FIFO is not empty and num < N do
 3:    node ← Q_FIFO.pop()
 4:    num ← num + 1
 5:    ERROR-EVALUATE(node, vp)
 6:    if NEED-APPLY(node) then
 7:       APPLY-OP(node)
 8:       new_front_nodes ← ADJUST-FRONTS(node)
 9:       Q_FIFO.push_back(new_front_nodes)
10:    end if
11: end while
```

Figure 4.9: Constant-time traversal algorithm.

In this algorithm, the parameter N is defined differently from that of the other two algorithms. It restricts the number of front nodes to be traversed and evaluated in one traversal. Each node that goes through ERROR-EVALUATE is counted in line 4, no matter whether this node will be selected to apply or not. Hence the time complexity for constant-time traversal is $O(N)$.

4.5 Efficient representation of selective mesh

The result of the DAG hierarchy traversal is a set of vertex-split operators and a set of edge-collapse operators. These two sets of operators will be streamed to the client and applied to the selective mesh. It is very important for the client to have an efficient representation of the selective mesh for both mesh modification and rendering. Although some boundary representations like winged-edge structure [7] and half-edge structure [130] are flexible and efficient for dynamic mesh and adjacency queries, they are not directly suitable for rendering. And for low-capacity clients, the memory requirements of these representations are rather high. Since the vsplit/ecol operators in our view-dependent LOD scheme have predetermined modifications (Chapter 3), we

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

do not need the adjacency queries supported by these mesh representations. Inspired by [10], we use OpenGL vertex arrays [135] as the core format of the selective mesh. This representation stores the geometric data, the attribute data and the connectivity data in compact arrays, such as vertex coordinate array, normal array, indexed triangle array etc., without data redundancy. It also benefits the rendering speed by reducing the subroutine call overhead and per-vertex calculations. Compared with OpenGL display list [135], the representation of vertex arrays is more advantageous in this case, as it allows frame-to-frame mesh modification.

However, in order to apply the vsplit/ecol operators to the selective mesh represented by vertex arrays, some auxiliary data structures are needed. The arrays are allocated within the restriction of the triangle-budget of the client, and the size of the selective mesh is often much smaller than the full model on the server. Thus, the vertices and triangles of the selective mesh are indexed differently from the vertex indices and triangle indices in the vsplit/ecol operators. *ID-maps* are necessary on the client to translate the indices between the full-model space and the array space of the selective mesh (refer to Figure 4.10).

To apply each operator to the selective mesh, the indices of the vertices and triangles referred in the operator need to be translated to the array-space indices to refer to the correct vertices or triangles on the selective mesh. So for both vertices and triangles, an ID-map from the full-model space to the array space ($T_{F\rightarrow A}$ for the triangles and $V_{F\rightarrow A}$ for the vertices in Figure 4.10) is necessary. When a triangle is deleted from the triangle array, we need to rearrange the triangle array, as the triangle array is required to be a continuous block by OpenGL. We remove the last triangle in the triangle array and use it to fill the hole left by the deleted triangle. This movement involves the change of the ID-mapping for the moved triangle. In order to quickly locate the entry of the moved triangle in $T_{F\rightarrow A}$, we need another ID-map for the triangle array, which translates

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

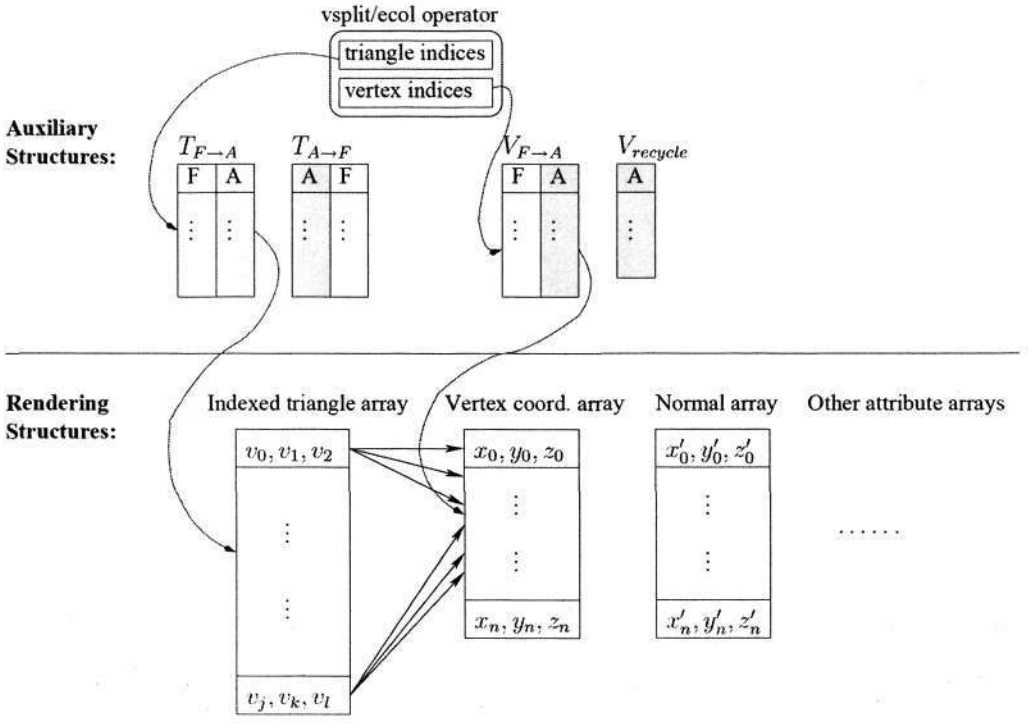


Figure 4.10: The format of the selective mesh on the client. An example of ID mapping is shown with the curve lines.

the indices from the array space to the full-model space ($T_{A\to F}$). For the vertices, since leaving the deleted vertices in the array will not affect the rendering result, we can avoid rearranging the vertex coordinate array and other attribute arrays after one vertex is deleted. However, another list (V_{recyle}) is needed to record the positions released by the deleted vertices so that new vertices added to the selective mesh can be inserted in these positions.

The mesh updating procedure can be optimized by streaming and applying the vertex-split operators and edge-collapse operators in pairs, so that the new vertex and new triangles of the vertex-split operator can reuse the memories of the vertex and the triangles deleted by the edge-collapse operator, and hence memory management for mesh updating can be minimized.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

4.6 Results

4.6.1 Implementation details

The run-time view-dependent LOD rendering of our DAG multiresolution hierarchy is implemented as a client-server based system. The server that performs the hierarchy traversal is a Pentium4 3.6GHz PC, and the client that maintains and renders the selective mesh is a Pentium4 1.6GHz PC with an ATI FireGL 8800 64MB graphics card. Three different traversal algorithms presented in Section 4.4 are implemented on this system. All three traversal algorithms are performed under the constraints and the management strategy discussed in Section 4.3. For the fidelity constraint, the out-of-view simplification and the back-face simplification are considered, and the screen-space error threshold is 0.1%. The mesh-change constraint is 500 for all three traversal algorithms. More specifically, for the priority-oriented traversal and the linear traversal, the maximum numbers of vertex-splits and edge-collapses per frame are both 500, and for the constant-time traversal the number of front nodes to be traversed is 500 for both simplification traversal and refinement traversal. Without any acceleration technique and under the OpenGL immediate mode [135], the client can render a model of 60000 triangles at an average of $34ms$, which is the upper limit for an interactive frame rate ($30fps$). So we assign a 60000 triangle-budget on the client.

4.6.2 Experimental results

We carry out extensive experiments on our run-time view-dependent LOD rendering system with the three traversal algorithms, on various models, including the Bunny (69451 triangles), the Hand (654666 triangles), the Dragon (871306 triangles) and the Happy (1087474 triangles). The performance data, including the traversal time (T_{trav}), mesh updating time (T_{upd}), rendering time (T_{rnd}), the number of triangles rendered per

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

frame and the number of mesh-updates per frame are shown in Table 4.1. The traversal time is measured on the server and all other data are measured on the client.

We analyze the efficiency of our DAG-based multiresolution hierarchy by comparing it with the FastMesh [98] scheme, which is a binary hierarchy of half-edge-collapse operators for stand-alone view-dependent rendering system. In the implementation of the FastMesh, the linear traversal is used, so we make the comparison with the data from the linear traversal in our experiments. The amortized traversal time for one mesh-update in our system with the linear traversal is about $7\mu s$ on the server PC with a Pentium4 3.6GHz CPU, comparable to that of the FastMesh, which is about $20\mu s$ to $40\mu s$ on a Sun Ultra60 workstation with a 450MHz UltraSPARC-II CPU [98]. The amortized time for applying one mesh-update to the selective mesh in our system is about $7\mu s$ on the client PC with a Pentium4 1.6GHz CPU, slightly slower than that of the FastMesh, which is about $3\mu s$ to $5\mu s$ [98], but faster than the data reported in [71], which are $22\mu s$ for vertex-split operators and $17\mu s$ for edge-collapse operators on an 866MHz Pentium III system. Although our selective mesh representation based on OpenGL vertex arrays (see Section 4.5) does not provide faster mesh updating than the FastMesh, it does offer much faster rendering speed. The rendering task of about 50000 triangles takes about $17ms$ on our client PC, while the rendering task of the same number of triangles in the FastMesh took about $60ms$ on the Sun Ultra60 workstation with an Expert3d graphics card [98].

We also show the memory efficiency of our selective mesh representation in Table 4.2. The rendering data structures, including the vertex coordinate array, the normal array and the indexed triangle array, are in compact format and only take about 1.4MB for the selective mesh of 60000 triangles. The auxiliary data structures cost about 66% of the memory of the rendering structures, which is a low overhead compared with the half-edge data structures used in [30, 98] where the extra memory cost is at least 150%

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Table 4.1: Average run-time data with three different traversal algorithms: priority-based traversal, linear traversal and constant-time traversal.

		Bunny	Hand	Dragon	Happy
Pri. Trav.	T_{trav} (ms)	4.2	10.8	11.1	10.9
	T_{upd} (ms)	2.6	4.8	5.5	5.4
	T_{rnd} (ms)	10.3	22.1	21.8	22.1
	Num. Triangles	36258	59136	59142	59084
	Num. Updates	335	691	789	766
Linear Trav.	T_{trav} (ms)	2.6	5.1	5.0	5.2
	T_{upd} (ms)	2.6	5.2	5.6	5.5
	T_{rnd} (ms)	10.2	23.6	21.6	22.1
	Num. Triangles	36526	59706	58654	59215
	Num. Updates	336	724	796	782
Cons. Trav.	T_{trav} (ms)	0.8	1.7	1.7	1.7
	T_{upd} (ms)	2.0	3.8	4.5	4.2
	T_{rnd} (ms)	9.9	23.2	22.0	22.8
	Num. Triangles	35604	59660	59178	59742
	Num. Updates	246	498	633	584

of memory of the data for rendering.

In order to compare the three different traversal algorithms, we test each traversal algorithm on the four models over two scripted interaction paths. The first path has 180 frames, in which the viewpoint is moving close to the model, and then scanning the model from the bottom to the top and from the right to the left. The second path has 194 frames, in which the viewpoint is moving close to the model and rotating around the model, first at increasing speed then at uniform speed. As an example, Figure 4.11(a)

Table 4.2: The memory footprint of the selective mesh on the client for both the auxiliary data structures and the rendering data structures.

	Bunny	Hand	Dragon	Happy
Mem. Aux. (MB)	0.595	0.956	0.954	0.957
Mem. Ren. (MB)	0.898	1.435	1.436	1.433

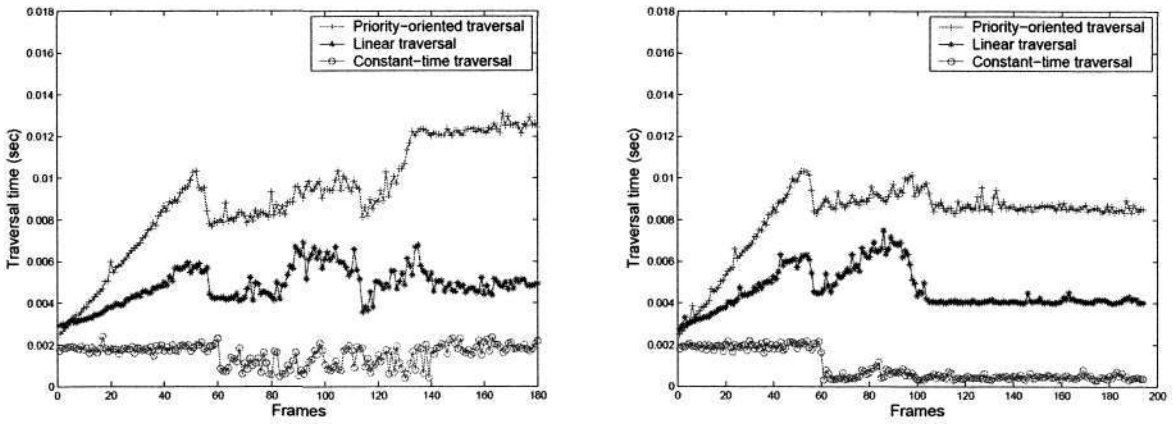
ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

and Figure 4.11(b) compare the traversal times of the three traversal algorithms on the Happy model over the two paths respectively.

Another important aspect of the performance of the traversal algorithms is how precisely the resultant mesh is refined/simplified to the current view. Since the priority-oriented traversal always selects the most appropriate mesh-updates, we assume that the selective mesh from the priority-oriented traversal has all and the only correct triangles according to current view-parameters and other traversal constraints. The accuracy of the selective mesh from the linear traversal or the constant-time traversal is measured by the percentage of the correct triangles that the linear traversal or the constant-time traversal can achieve for every 5 frames. The accuracies of the two algorithms on the Happy model over the two interaction paths are compared in Figure 4.12(a) and Figure 4.12(b). To visually show the differences of the selective meshes from the three traversal algorithms, Figure 4.13 and Figure 4.14 compare the rendering results of the three traversal algorithms on the Happy model over the first interaction path and the second interaction path respectively.

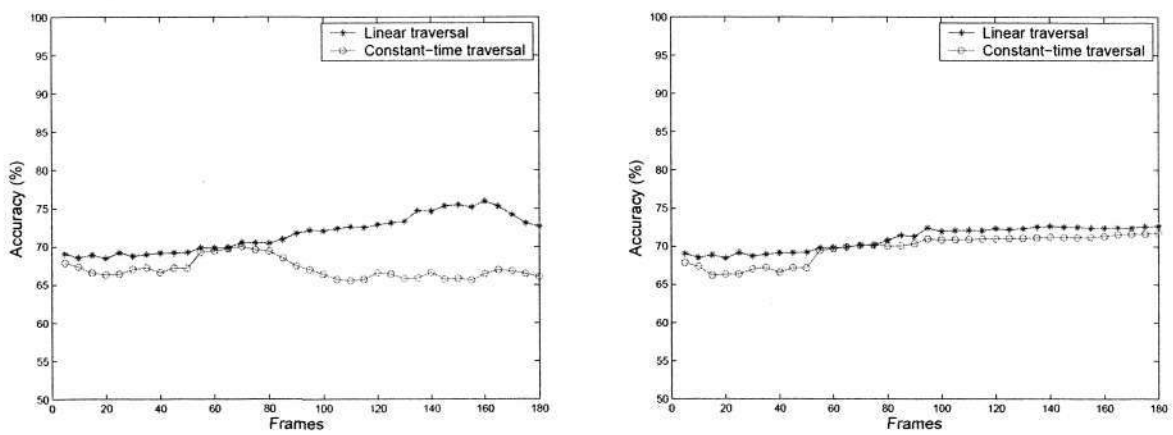
Our results show that the priority-oriented traversal can provide the best view-dependent rendering quality, but its traversal time is the longest among the three algorithms. In our experiment settings, it can easily be more than $10ms$, which is rather expensive for a rendering system targeting at interactive frame rate. Linear traversal has the next better view-dependent rendering quality. It can achieve about 70% accuracy of the selective mesh compared with the priority-oriented traversal, but takes much less traversal time. The constant-time traversal can provide very short traversal time, but produces less mesh-updates per frame and even lower accuracy of the selective mesh. The constant-time traversal is preferred when the interactivity has a higher priority. The not-so-precise rendering quality can be compensated by the high interaction speed. Also if there are several consecutive frames without much change of the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



(a) (b)

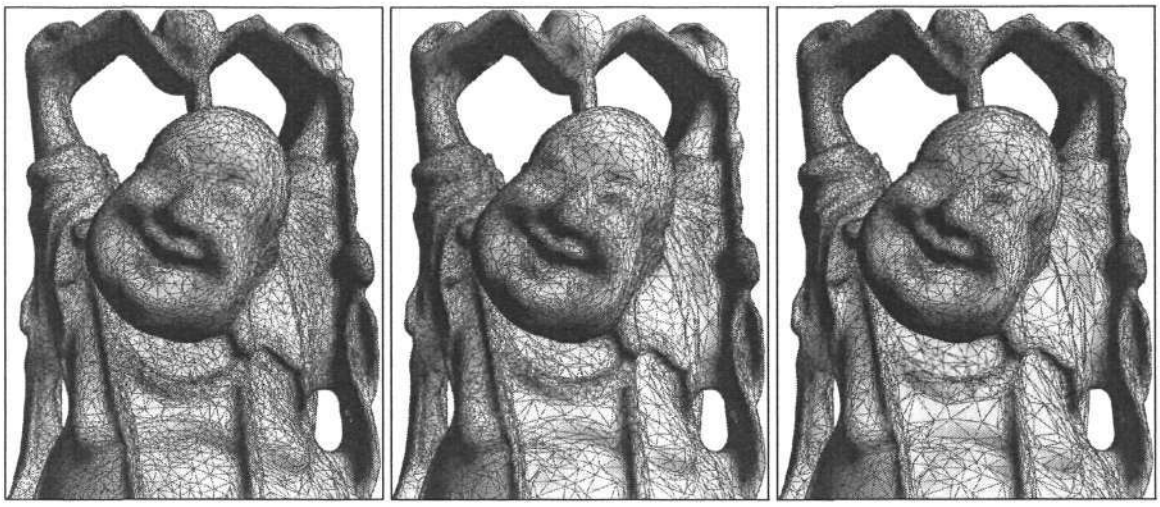
Figure 4.11: The traversal times of the three traversal algorithms on the Happy model. (a) Over the first interaction path. (b) Over the second interaction path.



(a) (b)

Figure 4.12: The accuracies of the linear traversal and the constant-time traversal on the Happy model. (a) Over the first interaction path. (b) Over the second interaction path.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



(a) (b) (c)

Figure 4.13: Rendering results of the 135th frame in the first interaction path. (a)Using the priority-oriented traversal. (b)Using the linear traversal. (c)Using the constant-time traversal.

view-parameters, the rendering quality will catch up. The priority-oriented traversal is best used in the systems where the rendering quality is more emphasized than the interactivity. And the linear traversal is a compromised solution.

Another interesting fact we find through our experiments is that for all three traversal algorithms, the size of the fronts in the DAG hierarchy of vertex-splits is always about 1/4 to 1/3 of the number of triangles of the selective mesh. Table 4.3 shows this relationship for the three traversal algorithms on four different models. This feature proves that the DAG hierarchy of vertex-splits has a smaller number of front nodes than the vertex trees (refer to Section 3.1).

4.7 Summary

In this chapter, we present the run-time view-dependent rendering of our DAG-based multiresolution hierarchy. The problem of multiresolution hierarchy traversal is focused.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

(a) (b) (c)

Figure 4.14: Rendering results of the 70th frame in the second interaction path. (a)Using the priority-oriented traversal. (b)Using the linear traversal. (c)Using the constant-time traversal.

We discuss the maintenance of the fronts of the DAG hierarchy during the run-time view-dependent rendering. A general traversal management strategy is proposed that is especially useful for client-server based view-dependent LOD systems. We compare three different traversal algorithms under the traversal management strategy. The advantages and disadvantages of the three algorithms are shown through experimental results. The comparison results can be used as a guide to choose the most appropriate

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Table 4.3: The number of triangles of the selective mesh and the number of front nodes at the 160th frame in the first interaction path.

		Bunny	Hand	dragon	Happy
Pri. trav.	Triangles	16647	57638	54750	59618
	Front nodes	2480	15747	16613	15542
Linear trav.	Triangles	16668	58445	53159	58709
	Front nodes	2483	16018	15860	15317
Cons. trav.	Triangles	16589	55403	57259	57813
	Front nodes	2477	15305	17284	15102

traversal algorithm for different system requirements. We also propose an efficient representation for the client-side selective mesh that is optimal for OpenGL rendering and with low memory overhead. We demonstrate the efficiency of our view-dependent LOD scheme through a series of experiments.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Chapter 5

Predictive Parallelism for View-dependent Multiresolution

5.1 Introduction

For a client-server based view-dependent LOD rendering system, the cost of multiresolution hierarchy traversal on the server and the network latency can be rather expensive for each frame rendered on the client, and can severely hamper the system's frame rate. To reduce the overheads from hierarchy traversal and network communication, client-side cache and compression techniques have been used in previous frameworks [30, 35], but these two techniques are not favorable to small-memory clients or networks with long round-trip times, as we explained in Chapter 1.

Luebke *et al.* [87] proposed the asynchronous view-dependent rendering on a shared-memory multiprocessor system, in which the view-dependent simplification task and the rendering task are run in parallel asynchronously. The rendering task can keep a consistent frame rate. If the simplification task falls behind, the scene rendered for the viewer will be in somewhat coarse quality until the simplification task catches up. A similar asynchronous view-dependent LOD rendering scheme on shared-memory multiprocessors is described by El-Sana *et al.* [38]. Inspired by their works, we propose a novel

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

predictive parallel approach to handle the per-frame overhead for client-server based view-dependent rendering. Our approach does not impose extra memory requirement on the client and is scalable to networks with relatively long round-trip times.

This chapter presents the basic predictive parallel approach and focuses on client-server systems in LAN environments with very short round-trip times. Chapter 6 will extend this approach to networks with relatively long round-trip times. The key idea is to run the client process and the server process in parallel, using the client rendering time to cover the network latency and the traversal time on the server. In order to achieve best possible view-dependent rendering results, the two processes running in parallel should be carefully coordinated. We realize the coordination with view-parameters prediction. This predictive parallel approach guarantees the interactive frame rate and provides the best-effort view-dependent refinement on the client. Our experiments show that the predictive parallel approach works well in LAN environments. The low-capacity client keeps only the selective mesh within its triangle-budget, in a format that is optimal for rendering and has little memory overhead (discussed in Chapter 4).

The rest of the chapter is organized as follows. Section 5.2 describes the parallelism of the client process and the server process. Section 5.3 discusses the view-parameters prediction. In section 5.4, we provide the implementation details and the experimental results. Finally, a chapter summary is given in Section 5.5.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

5.2 Parallelism for client-server view-dependent rendering

5.2.1 Overview

In our client-server based view-dependent LOD rendering system, the client maintains and renders the selective mesh locally and the server manages the multiresolution mesh hierarchy. In the initialization step, the client sends a query for a model to the server and the server delivers the base mesh of the model to the client. Then the client interacts with the user and sends the view-parameters to the server. The server traverses the multiresolution hierarchy to produce a stream of mesh simplification and refinement operators according to the received view-parameters. These mesh-update operators are sent to the client and the selective mesh is modified and rendered to the user.

From the perspective of the client, the whole process of displaying one frame to the user includes two main procedures, *the procedure of getting updates* (PGU in short) that involves the network routine and the routine of multiresolution hierarchy traversal on the server, and *the procedure of rendering* (PRD in short) that involves the routine of updating the selective mesh and the actual rendering routine.

In the conventional sequential method (refer to Figure 5.1) used in previous works [34, 35, 30], a typical frame on the client starts from sending the view-parameters to the server, going through the waiting to get the mesh-updates from the server, and ends when the selective mesh is updated and rendered. The problem for this sequential method is that the network overhead plus the traversal time on the server side can be large enough to compromise the interactivity on the client side.

Like the asynchronous view-dependent rendering on shared-memory multiprocessors [87, 38], the procedure of rendering (PRD) and the procedure of getting updates (PGU) in a client-server system also have the potential to run in parallel to boost the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

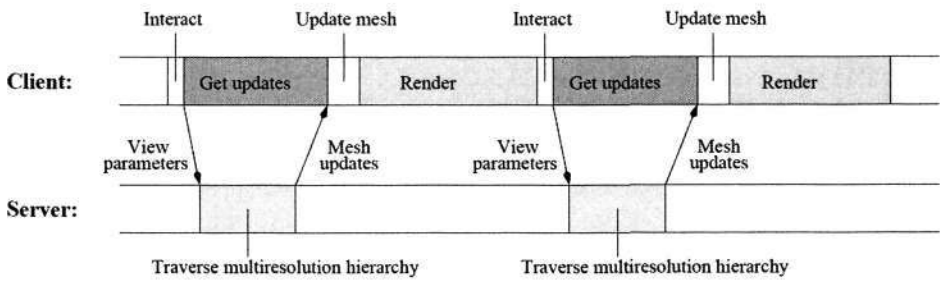


Figure 5.1: The sequential method in a client-server based view-dependent rendering system. Before rendering each frame, the client waits to get the mesh-updates from the server for the current frame.

frame rate, as these two procedures do not have heavy resource competitions. For PRD, the rendering routine mainly relies on the graphics hardware of the client. For PGU, most of the time is spent waiting for the server's response and this portion of time comprises the network latency and the time for multiresolution hierarchy traversal on the server. Thus, the running times of the two procedures can be overlapped to achieve better frame rate. Suppose the time cost for interaction is negligible. Let T_g be the time cost for PGU. The time cost for PRD has two parts, T_u for the time of updating the selective mesh and T_r for the time of the actual rendering routine. In the sequential method, the frame time is $T = T_g + T_u + T_r$. If PGU and PRD can be run in parallel asynchronously as in [87, 38], the frame time can be reduced to $T = T_u + T_r$.

However, just asynchronously parallelizing the two procedures as on the shared-memory multiprocessors [87, 38] is not enough. As we discussed in Chapter 4, we need to regulate the number of mesh-updates applied to the selective mesh per frame to maintain a consistent frame rate. Thus, we require the mesh-updates to be transmitted on batch basis, where one batch is for one frame and each batch has the same maximum limit on the number of mesh-updates. In order to produce good rendering quality, we expect each frame to receive a batch of mesh-updates from the server corresponding to its current view-parameters before the frame is rendered. Therefore, the key issue is

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

a synchronization mechanism to coordinate the two procedures running in parallel, so that each frame can receive its corresponding batch of mesh-updates in time.

5.2.2 Synchronization

Before each frame can be rendered, the client needs the mesh-updates for the current frame to keep the selective mesh up-to-date. In a LAN with very short round-trip time, the procedure of getting updates (PGU) usually takes less time than the procedure of rendering (PRD). Therefore the PGU for the next frame can be parallelized with the PRD for the current frame, so that the time cost of PGU can be hidden and the client can always get the necessary mesh-updates in advance. Figure 5.2 shows the parallel strategy in this case. Before each frame is rendered, instead of sending the current view-parameters to the server as in the sequential method, the client predicts the view-parameters for the next frame and sends the predicted view-parameters to the server as mesh-updates request. Receiving the mesh-updates for the next frame and rendering the current frame are going on simultaneously. When the next frame is to be rendered, the required mesh-updates have already been delivered to the client and are ready to be applied. As this strategy needs to predict the view-parameters for the next frame, we call it *one-frame predictive parallel* strategy.

In the system structure shown in Figure 5.2, the PGU works as a middle layer between the PRD and the server process. The view-parameters are sent from the PRD to the server through the PGU, and also the PRD receives the mesh-updates from the server through the PGU.

We present a batch of mesh-updates as $U(x, y)$, where x and y are the frame indices and $x < y$. This means the batch of mesh-updates is supposed to be received at $frame_y$ and the mesh-update request with the predicted view-parameters for $frame_y$ is sent to the server at $frame_x$. After the mesh-update request with the predicted view-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

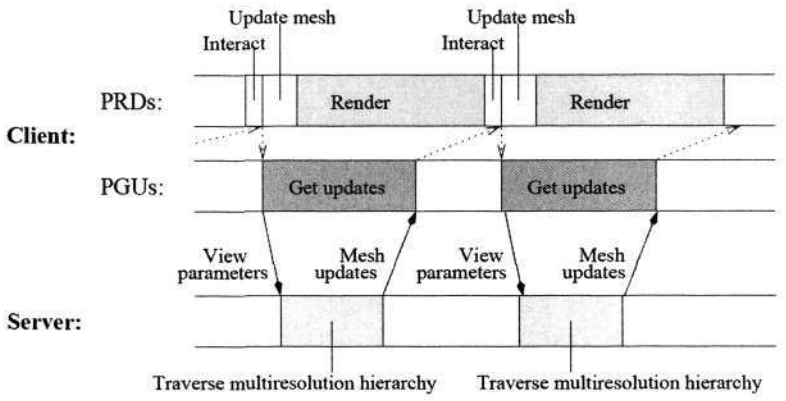


Figure 5.2: The one-frame predictive parallel strategy in a client-server based view-dependent rendering system with short round-trip time. Before rendering each frame, the client has already obtained the mesh-updates for the current frame and starts getting mesh-updates for the next frame.

parameters for $frame_y$ is sent to the server at $frame_x$ and before the corresponding mesh-update $U(x, y)$ can be applied to the selective mesh, we say $U(x, y)$ is *pending*. In the case of one-frame predictive parallelism, a batch of mesh-updates is always in the form of $U(i-1, i)$, where i is the frame index, and the standard pending period of a batch of mesh-updates should be 1 frame.

However, the network performance is influenced by many uncertain factors. Sometimes due to a sudden and transient network slowdown, the batch of mesh-updates $U(i-1,\ i)$ may not be delivered to the client in time for $frame_i$. Like the asynchronous parallelism on shared-memory multiprocessors in [87, 38], we render the old selective mesh from $frame_{(i-1)}$ until the batch $U(i-1,\ i)$ is received and applied at a later frame $frame_{(i+x)}$, where $x \geq 1$. However, unlike the asynchronous parallelism on shared-memory multiprocessors, we also consider the coordination of the batches of mesh-updates and their corresponding frames. As we discussed in Section 5.2.1, in order to keep a smooth frame rate, only one batch of mesh-updates will be applied at each frame. We call frames from $frame_i$ to $frame_{(i+x-1)}$ *unmodified frames*, since

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

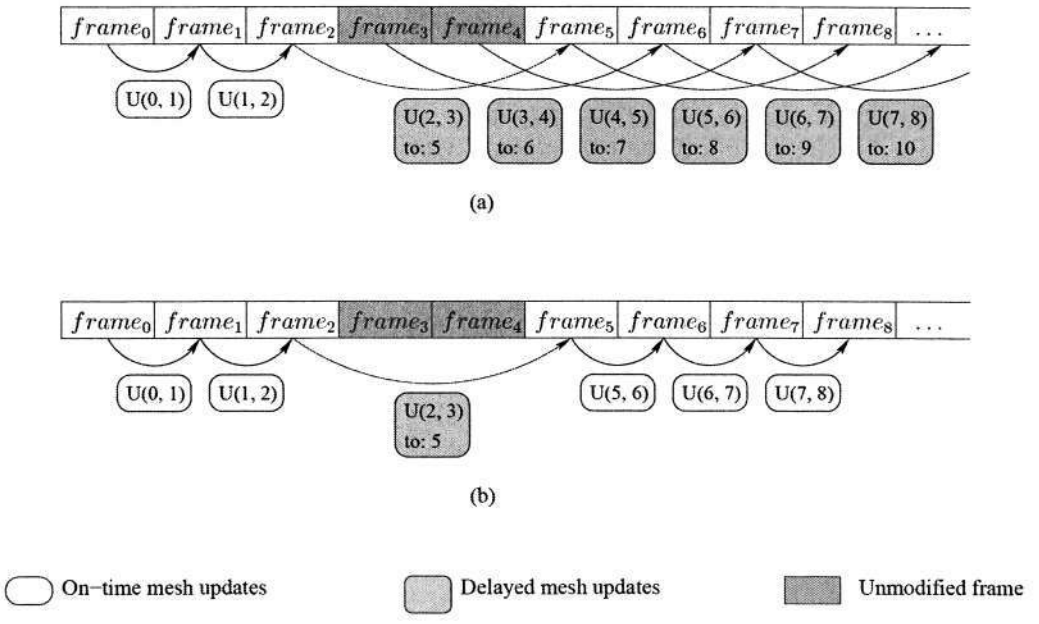


Figure 5.3: An example in one-frame predictive parallelism when $U(2,3)$ is 2-frames late. (a) If each unmodified frame is allowed to send mesh-update request for the next frame, all the subsequent batches of mesh-updates will be applied 2-frames late to the selective mesh. (b) Unmodified frames do not send mesh-update requests for the next frames and the delay will not propagate to the subsequent frames.

no mesh-updates are applied to the selective mesh at these frames. During the period from $frame_i$ to $frame_{(i+x-1)}$, the batch of mesh-updates $U(i-1,\ i)$ is pending, resulting in a pending period of $x+1$ frames. If each unmodified frame of $frame_i$ to $frame_{(i+x-1)}$ sends a mesh-update request for a future frame, the corresponding batch of mesh-updates and all the subsequent batches will also have a pending period of $x+1$ frames. In other words, all the subsequent batches of $U(i-1,\ i)$ will be x-frames late. Thus one batch of mesh-updates outdated will cause the subsequence batches outdated. We show in Figure 5.3(a) an example of a 2-frames late batch for one-frame predictive parallel strategy.

The outdated mesh-updates will reduce the accuracy of the view-dependent rendering. In order to control the number of outdated mesh-updates, we enforce that the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

unmodified frames should not send mesh-update requests for future frames and only those frames that get mesh-updates should send requests for future frames, so that at any point of time there is only one batch of mesh-updates pending in the one-frame predictive parallelism, and the delay will not propagate to the subsequence frames. Figure 5.3(b) illustrates the synchronization of late mesh-updates.

5.2.3 Algorithm details

PROCESS SERVER
1: initialize
2: send *base_mesh*
3: **loop**
4: receive *view_para*
5: *mesh_updates* ← TRAV-MULTI-HIERARCHY(*view_para*)
6: send *mesh_updates*
7: **end loop**

Figure 5.4: The server process.

The server process is detailed in Figure 5.4. The function TRAV-MULTI-HIERARCHY in line 5 traverses the multiresolution hierarchy according to the received view-parameters *view_para* and generates the corresponding batch of mesh-update operators *mesh_updates* for the client. The multiresolution hierarchy traversal is described in Chapter 4. We suppose that the parameters for other traversal constraints introduced in Chapter 4, such as the traversal mode and the constraint on the amount of mesh-updates per traversal, are also included in the parameter *view_para*.

The client process is composed of two independently running procedures, RENDER and GET-UPDATES, synchronized by two mutually exclusive shared data, *view_para* and *mesh_updates*, as described in Figure 5.5. Function CURRENT-VIEW-PARA in line 3 of the procedure RENDER is to initialize *view_para* as the view-parameters of the current frame and function PREDICT-NEXT-VIEW-PARA in line 7 assigns *view_para*

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library


```
PROCEDURE RENDER
 1: initialize
 2: receive base_mesh
 3: view_para ← CURRENT-VIEW-PARA()
 4: loop
 5:    interact
 6:    if NOT-APPLIED(mesh_updates) then
 7:       view_para ← PREDICT-NEXT-VIEW-PARA()
 8:       apply mesh_updates
 9:    end if
10:    render
11: end loop
PROCEDURE GET-UPDATES
 1: loop
 2:    if NOT-SENT(view_para) then
 3:       send view_para
 4:       receive mesh_updates
 5:    end if
 6: end loop
```

Figure 5.5: The client process with one-frame predictive parallelism.

the predicted view-parameters for the next frame. The function NOT-APPLIED in line 6 of the procedure RENDER ensures that each new *mesh_updates* received by the procedure GET-UPDATES will be applied to the selective mesh only once. Similarly, the function NOT-SENT in line 2 of the procedure GET-UPDATES ensures that each new *view_para* assigned by the procedure RENDER will be sent to the server only once. The synchronization of the batches of mesh-updates is realized through the coordination of parameters *view_para* and *mesh_updates*.

5.3 View-parameters prediction

In this section, we discuss how the function PREDICT-NEXT-VIEW-PARA in Figure 5.5 predicts the view-parameters for the next frame. The view-parameters typically include eye position, view direction, the size and shape of the view-frustum, and the screen-space

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

error tolerance. Since the size and shape of the view-frustum and the screen-space error tolerance do not change very often, we only need to consider the constantly changing view-parameters of eye position and view direction. More specifically, the client needs to predict two points in the virtual world space for the next frame, the eye position E and the aim-at position A along the view direction.

The interaction pattern of navigating in a virtual world or viewing a model can be approximated as piece-wise straight lines connected by halts and abrupt changes in speed or direction. Within a very short time (several frames), the interaction pattern can be approximated as uniform motion in a straight line. So, the eye position and the aim-at position for the next frame can be extrapolated based on the actual eye position and aim-at position of the current frame and the previous frame in a way similar to the first-order dead-reckoning algorithm used in distributed simulations [1]:

$$E'_n = E_c + (\overrightarrow{E_c - E_p}),$$

$$A'_n = A_c + (\overrightarrow{A_c - A_p}),$$

where E_c and A_c are the actual eye position and aim-at position of the current frame, E_p and A_p are the actual eye position and aim-at position of the previous frame, and E'_n and A'_n are the predicted eye position and aim-at position for the next frame in the future.

As the view-parameters prediction needs to be performed on client-side, this prediction algorithm is highly efficient for interactive frame rate on low-capacity clients, because of its computational simplicity and its minimal memory requirement. Only 3 additions and 3 subtractions are necessary to predict the next eye position or aim-at position. Also, only 4 points (E_c, A_c, E_p and A_p) need to be stored to perform the prediction. Due to the temporal coherency, this prediction algorithm works well for the regular movement periods during the interaction. Even when abrupt changes in

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

speed and view direction happen, resulting in large prediction error, this error will be corrected in the next few frames as long as the following changes of the view-parameters stay relatively stable for a few frames, which is usually the case in the interaction for viewing models.

Figure 5.6 shows an example of predicting the eye position, with $E'_i = E_{i-1} + (\overrightarrow{E_{i-1} - E_{i-2}})$. An abrupt change happens at E_4 and the predicted eye position E'_4 deviates greatly from the actual eye position E_4. But the prediction accuracy increases in the following frames as the eye position changes steadily.

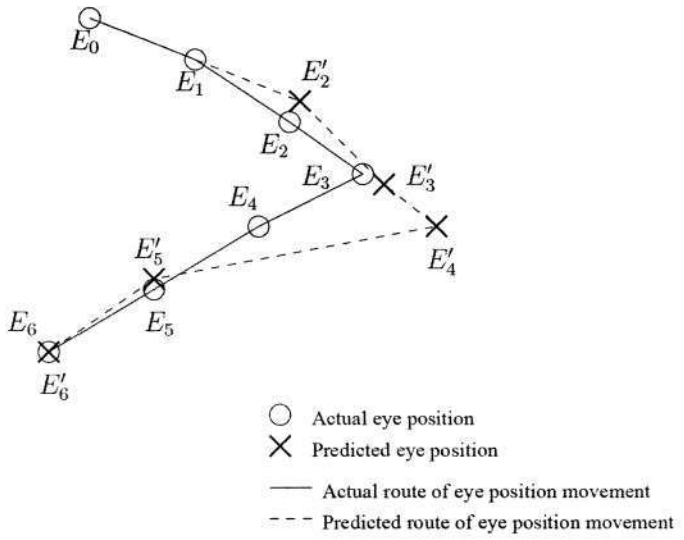


Figure 5.6: One-frame view-parameters prediction. Each predicted eye position E'_i is predicted based on two previous actual eye positions E_{i-1} and E_{i-2}.

5.4 Results

5.4.1 Implementation details

We have implemented the one-frame predictive parallel algorithm and compared it with the conventional sequential algorithm on our client-server based view-dependent LOD

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

rendering system proposed in Chapter 3 and Chapter 4. Same as in Section 4.6, the server runs on a Pentium4 3.6GHz PC and the client runs on an old PC with Pentium4 1.6GHz CPU and an ATI 64MB graphics card. The client and the server are set up on our 100Mbps campus LAN. The round-trip time between the client and the server is $1ms$ on average. We use TCP/IP sockets for network communication. The maximum size of the packet of a batch of mesh-updates for a frame is 64KB, so the maximum transmission time for a packet of mesh-updates is about $6ms$. The packet of the view-parameters is only 33 bytes and its transmission time is negligible.

Each packet of mesh-updates contains a stream of edge-collapse operators and a stream of vertex-split operators. We set the maximum number of updates per frame to 1000 (500 vertex-split operators and 500 edge-collapse operators) to regulate the mesh updating time and transmission time (as discussed in Section 4.3). Also the client-side triangle-budget is assigned to 60000, the same as in Section 4.6. Priority-oriented traversal (Section 4.4.1) is used, so that the 1000 mesh-updates delivered to the client are the most necessary ones according to the predicted view-parameters. The client with the one-frame predictive parallel algorithm is implemented as two threads, one for the procedure of rendering (PRD), and the other for the procedure of getting mesh-updates (PGU).

5.4.2 Experimental results

In Table 5.1, we present the average time costs per frame for getting mesh-updates from the server (T_g), updating the mesh (T_u), rendering the mesh (T_r) and the total frame time (T) for two algorithms, the sequential algorithm and the one-frame predictive parallel algorithm. T_g includes the round-trip time between the client and the server, the transmission time and the time the server spends traversing the multiresolution hierarchy and generating the mesh-updates. Notice that for the one-frame predictive

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

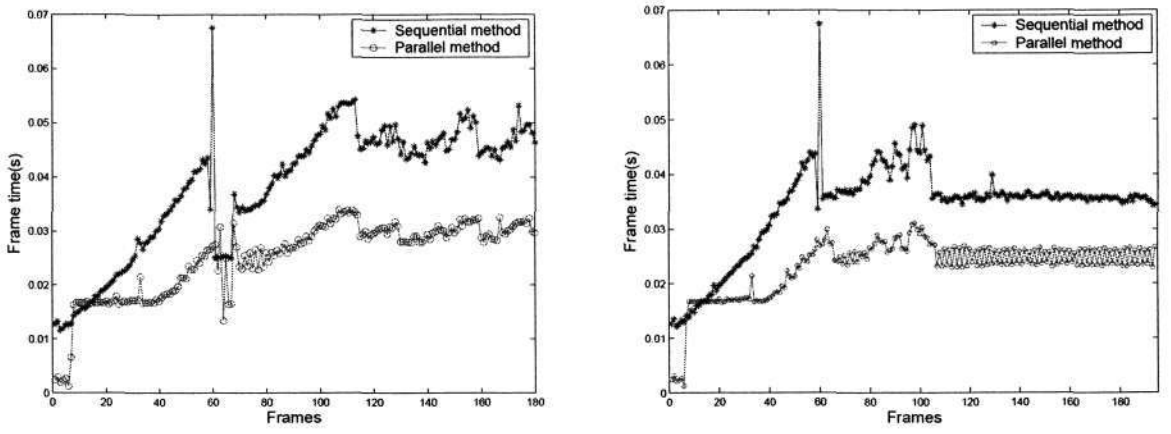
Table 5.1: Comparison of the sequential algorithm (Seq.) and the one-frame predictive parallel algorithm (Par.) on various models. The numbers of triangles of the original models are labelled with the model names. "*Num. Tri.*" is the number of triangles rendered. "*Num. Up.*" is the number of mesh-updates. T_g is time cost for getting the mesh-updates. T_u is the time cost for updating the mesh. T_r is the time cost for rendering. T is the total time for one frame.

	Bunny (69451)		Hand (654666)		Dragon (871306)		Happy (1087474)	
	Seq.	Par.	Seq.	Par.	Seq.	Par.	Seq.	Par.
Num. Tri.	36258	35996	59136	59168	59142	59150	59084	59040
Num. Up.	335	338	691	695	789	787	766	760
T_g (ms)	8.3	8.3	16.3	16.4	16.5	16.6	16.4	16.4
T_u (ms)	2.4	2.4	4.9	5.0	5.2	5.3	5.4	5.4
T_r (ms)	10.4	11.6	22.4	23.9	22.3	23.7	22.1	23.5
T (ms)	21.8	14.2	43.9	29.1	44.2	29.2	44.2	29.1

parallel algorithm over the LAN environment, usually T_g can be covered by T_u and T_r, i. e. $T_g < (T_u + T_r)$. The results in Table 5.1 show that the one-frame predictive parallel algorithm achieves about $15ms$ shorter frame time than the sequential algorithm when T_g is about $16ms$. The actual rendering time T_r of the one-frame predictive parallel algorithm is about $1ms$ to $2ms$ longer because of the multi-threading expense. The large models can be rendered at an interactive frame rate of about $30fps$ by the one-frame predictive parallel algorithm.

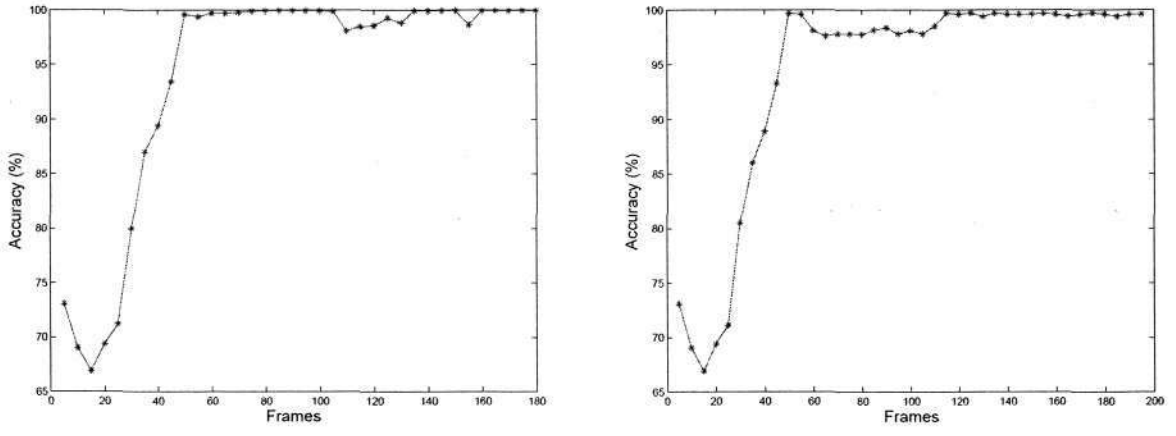
In the predictive parallel algorithm, we have frame time $T = T_u + T_r$. The mesh updating time T_u is affected by the maximum number of mesh-updates per frame. The rendering time T_r is affected by the client-side triangle budget. Given a constant frame rate, these two traversal constraints have a mutually restrictive relationship. Therefore, a larger triangle budget will sacrifice the amount of mesh-updates per frame. On the other hand, increasing the amount of mesh-updates per frame will reduce the triangle budget. Changing the constraint on the amount of mesh-updates will also affect the multiresolution hierarchy traversal time on the server, and hence affect T_g.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



(a) (b)

Figure 5.7: The frame time for the sequential algorithm and the one-frame predictive parallel algorithm on the Dragon model. (a) Over the first interaction path. (b) Over the second interaction path.



(a) (b)

Figure 5.8: The accuracy of the one-frame predictive parallel algorithm on the Dragon model. (a) Over the first interaction path. (b) Over the second interaction path.

Figure 5.7 compares the sequential algorithm and the one-frame predictive parallel algorithm on the Dragon model, over the same two scripted interaction paths used in Chapter 4. The first path has 180 frames, in which the viewpoint is moving close to

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

the model in a straight line, and then scanning the model from the bottom to the top and from the right to the left. The second path has 194 frames, in which the viewpoint is moving close to the model and rotating around the model, first at increasing speed then at uniform speed. At the starting position, the model is at a very low level of detail (below 3000 triangles). So for the parallel algorithm, the rendering time T_r is too short to cover T_g ($T_g > (T_u + T_r)$) in one frame. This leads to the result that the base mesh is rendered for 4 to 5 frames (unmodified frames) before the next batch of mesh-updates for refinement arrives. After about 70 frames, the thread of PRD and the thread of PGU can be nicely parallelized and gain a steadily better performance than the sequential algorithm.

We also test the accuracy of the selective mesh from the one-frame predictive parallel algorithm compared to the sequential algorithm using the same interaction paths. We assume that the selective mesh from the sequential algorithm has the whole set of correct triangles for each frame. To test the accuracy of the parallel algorithm, we measure the percentage of the correct triangles that the parallel algorithm can achieve in its selective mesh every 5 frames. The results on the Dragon model are shown in Figure 5.8 for the two paths. In the first 50 frames, the accuracy is not very high because several unmodified frames occur when the selective mesh still only has a very small number of triangles, as explained above. After the initial warm-up phase, the accuracy increases dramatically to a high level and stays relatively steady. In Figure 5.8(a) of the first interaction path, there are two slight drops of the accuracy at the 110th frame and the 155th frame. In Figure 5.8(b) of the second interaction path, the slight accuracy drop sustains from the 60th frame to the 110th frame. They are caused by the abrupt changes of the view movement. The accuracy increases back to about 100% after the abrupt changes. We have got similar results for frame times and accuracies for other models as for the Dragon model demonstrated in Figure 5.7 and Figure 5.8.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Figure 5.9 visually shows the comparison of the view-dependent rendering results for the two algorithms. Both algorithms start at the low level-of-detail shown in Figure 5.9(a). In the final frame of the interaction path, the one-frame predictive parallel algorithm achieves a refinement/simplification result very close to that of the sequential algorithm, compared in Figure 5.9(b)(d) and Figure 5.9(c)(e) respectively. We can see that for these two algorithms, the portion of the model inside the view-frustum is refined to a very similar extent and also the portion outside of the view-frustum is simplified to a very similar extent. Figure 5.8 and Figure 5.9 confirm the validity of our one-frame view-parameters prediction strategy.

Our predictive parallel algorithm is especially useful for the low-capacity clients, as the clients only need to keep the selective mesh in the compact representation described in Chapter 4 with no extra memory. Also the view-parameters prediction algorithm is efficient enough for low-capacity clients.

5.5 Summary

In this chapter, we have introduced a predictive parallel scheme for client-server view-dependent rendering, which can provide interactive frame rate on low-capacity clients. This chapter is focused on a special case of one-frame predictive parallelism for LAN environments with short round-trip times. We explore the potential of the parallelism of the procedure of rendering and the procedure of getting the new mesh-updates. The latter includes the network routine and the multiresolution hierarchy traversal on the server side. We also present the view-parameters prediction strategy that makes the parallelism feasible. The interactive performance of the predictive parallel algorithm and the validity of the prediction strategy have been shown through a series of experiments. The one-frame predictive parallel algorithm achieves much shorter frame time than the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

sequential algorithm and provides a very close view-dependent refinement/simplification result in the campus LAN experimental environment. This predictive parallel scheme can be extended to networks with long round-trip times, as we will demonstrate in Chapter 6.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

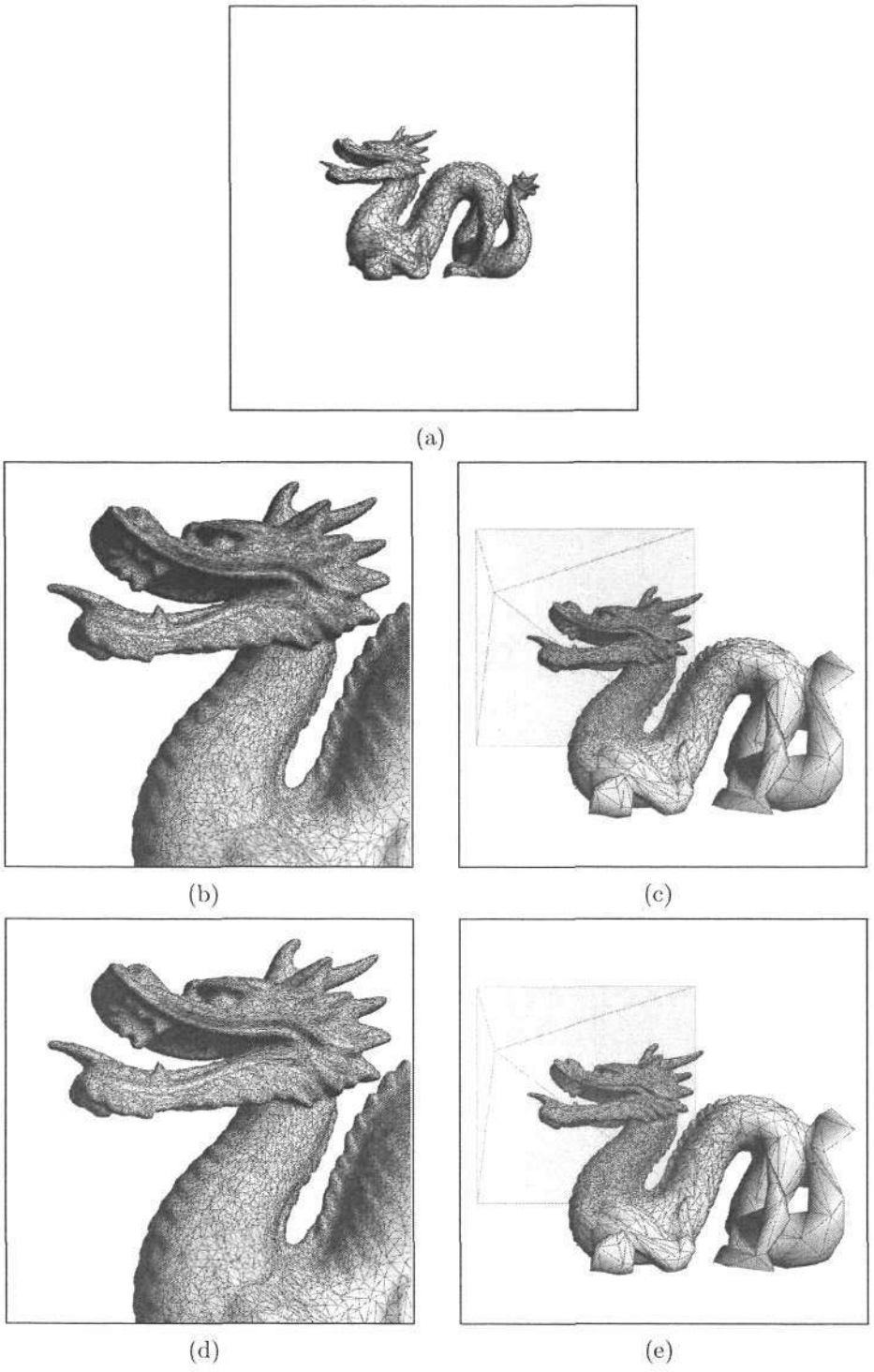


Figure 5.9: Rendering results for the sequential algorithm and the one-frame predictive parallel algorithm. (a) The starting frame for both algorithms. (b)–(c) The final frame for the sequential algorithm, the front view and a different view with view-frustum. (d)–(e) The final frame for the parallel algorithm, the front view and a different view with view-frustum.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Chapter 6

Predictive Parallelism on WAN

6.1 Introduction

In Chapter 5, we proposed a predictive parallel strategy to reduce the frame time for client-server based view-dependent LOD rendering systems. In the conventional sequential method, the total time for displaying one frame is $T = T_g + T_u + T_r$, where T_g is the time cost for the procedure of getting updates (PGU), and $T_u + T_r$ is the time cost for the procedure of rendering (PRD) with T_u for the time of updating the selective mesh and T_r for the time of the actual rendering routine. The one-frame predictive parallel strategy described in Chapter 5 can reduce the frame time to $T = T_u + T_r$. In network environments with very short round-trip times such as LANs, we often have $T_g < (T_u + T_r)$. In this case, the two procedures, PGU and PRD, can be synchronized as described in Chapter 5, so that each frame can receive a batch of mesh-updates for the selective mesh according the current view. However, this one-frame predictive parallel strategy cannot be directly applied to networks with long round-trip times, where $T_g > (T_u + T_r)$, as the PGU will always fall behind, leaving almost all batches of mesh-updates outdated and a large number of frames unmodified. This will severely impair the view-dependent rendering quality.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

In this chapter, we extend the parallel paradigm presented in Chapter 5 to networks with long round-trip times (RTTs) like WANs. Instead of running the PGU for the next frame and the PRD for the current frame in parallel, we manage to cover the PGU for one frame with multiple PRDs for multiple frames, and make the selective mesh for each frame modified with the appropriate batch of mesh-updates. Rather than just predicting the view-parameters for the next frame as in the LAN environments (see Chapter 5), we need to predict the view-parameters multiple frames ahead on networks with long RTTs. We call it *multi-frame predictive parallel* strategy. As the RTT between the client and the server in a WAN environment is constantly changing, we need to dynamically adjust the *prediction span*, which is the range of the view-parameters prediction, according to the current network condition. Our experimental results demonstrate that our approach can achieve an interactive frame rate while keeping an acceptable rendering quality for large triangle models over networks with relatively long RTTs.

The rest of the chapter is organized as follows. Section 6.2 presents the multi-frame predictive parallel strategy. In Section 6.3, we discuss the problem of dynamic adjustment of the prediction span. Section 6.4 shows the experimental results and Section 6.5 summarizes the chapter.

6.2 Multi-frame parallelism

In the network environments with long round-trip times, the time cost for the procedure of getting updates (PGU) can be several times greater than the time cost for the procedure of rendering (PRD). Thus, to keep the interactive frame rate on the client-side, the PGU for one frame has to be covered by the PRDs for several frames. Furthermore, each of these frames should receive an appropriate batch of mesh-updates to modify the selective mesh before it is rendered, in order to maintain a consistent acceptable

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

rendering quality.

As we discussed in Chapter 4, the multiresolution hierarchy traversal time on the server can always be under control. In WAN network environments, a large portion of the time cost for the PGU is the round-trip time. The actual time cost for multiresolution hierarchy traversal on the server is still usually smaller than the rendering time on the client. This property offers the potential to parallelize multiple PGUs for multiple frames and overlap the delays incurred in these PGUs, and hence to produce a batch of mesh-updates for each frame.

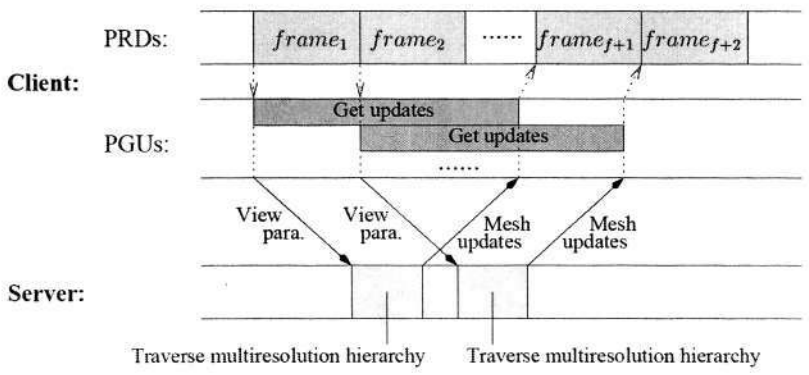


Figure 6.1: The multi-frame predictive parallel method in a client-server based view-dependent rendering system with a long round-trip time. Before each frame, the client has already obtained the mesh-updates for the current frame and starts getting mesh-updates for the f-th frame ahead.

This multi-frame predictive parallel approach is illustrated in Figure 6.1. Each PGU is covered by rendering f frames, and since f PGUs can be overlapped, each frame will receive a batch of mesh-updates from the server. The batch of mesh-updates received at $frame_{(f+i)}$ corresponds to the view-parameters sent at $frame_i$. Therefore, at $frame_i$, the predicted view-parameters for $frame_{(f+i)}$ should be sent to the server. We call f the *prediction span*.

The server process and the GET-UPDATES procedure of the client process remain

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

the same as for the one-frame predictive parallel algorithm described in Figure 5.5. The RENDER procedure of the client process in Figure 5.5 is modified into RENDER-WAN as follows in Figure 6.2 to allow multi-frame view-parameters prediction.

```
PROCEDURE RENDER-WAN(f)
 1: initialize
 2: receive base_mesh
 3: view_para ← CURRENT-VIEW-PARA()
 4: for i ← 1 to f do
 5:    send view_para
 6: end for
 7: loop
 8:    interact
 9:    if NOT-APPLIED(mesh_updates) then
10:       view_para ← PREDICT-FTH-VIEW-PARA()
11:       apply mesh_updates
12:    end if
13:    render
14: end loop
```

Figure 6.2: The rendering procedure for the client process with multi-frame predictive parallelism.

After the client receives the base mesh, it sends beginning view-parameters f times to the server to start the prediction cycle. Since at the beginning, the information for predicting the f-th frame ahead is not adequate, we use the view-parameters of the starting frame as the predicted view-parameters for $frames_f$, $frames_{(f+1)}$, $\ldots$, $frames_{(2f-1)}$. Then the client enters the rendering loop in line 7, the same as in the one-frame predictive parallelism in Figure 5.5, except that the function for view-parameters prediction is modified to be able to predict f-th frame ahead with the function PREDICT-FTH-VIEW-PARA in line 10. We can simply extend the one-frame view-parameters prediction strategy in Chapter 5 to multi-frame view-parameters prediction as follows:

$$E'_f = E_c + f \times (\overrightarrow{E_c - E_p}) \quad ,$$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

$$A'_f = A_c + f \times (\overrightarrow{A_c - A_p}) \quad ,$$

where E_c and A_c are the actual eye position and aim-at position of the current frame, E_p and A_p are the actual eye position and aim-at position of the previous frame, and E'_f and A'_f are the predicted eye position and aim-at position for the f-th frame in the future.

The first batch of mesh-updates is expected to be received just before the f-th frame is rendered, so the first f frames are unmodified frames and the model is approximated by its base mesh in this stage of rendering. After this stage, each frame will receive its batch of mesh-updates and send the predicted view-parameters for the f-th frame ahead. Figure 6.3 shows an example of the rendering procedure when the prediction span is 3 frames.

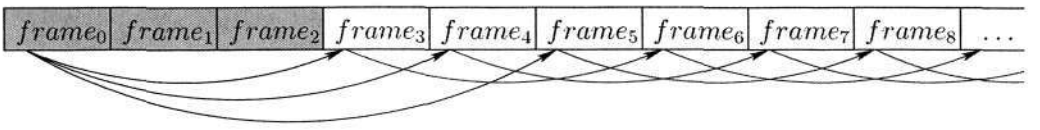


Figure 6.3: An example of the rendering procedure when the prediction span is 3 frames. The first three frames are unmodified frames. The regular prediction cycle begins at $frame_3$.

Similar to the situation discussed in Section 5.2.2, sometimes due to a sudden and transient network slowdown, a batch of mesh-update $U(i, f+i)$ (refer to Section 5.2.2) may not be delivered to the client in time at $frame_{(f+i)}$, but is received at a later frame $frame_{(f+i+x)}$, where $x \geq 1$. The same solution in Section 5.2.2 can be used here. Frames $frame_{(f+i)}$ to $frame_{(f+i+x-1)}$ are unmodified frames and will be rendered using the old selective mesh from $frame_{(f+i-1)}$. The unmodified frames will not send mesh-update requests for future frames. In this way, at any point of time there are only f batches of mesh-updates pending. If the batch of mesh-updates $U(i, f+i)$ is x-frames late, the following $f-1$ batches $U(i+1, f+i+1)$, $U(i+2, f+i+2)$, ..., $U(f+i-1,$

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

$2f + i - 1$), if exist, will also be x-frames late. However, since the x unmodified frames do not add extra pending batches, there will only be f late batches and the delay can be recovered after $U(f + i - 1,\ 2f + i - 1)$ is received at $frame_{(2f+i-1+x)}$. Therefore, the x-frames delay caused by one late batch of mesh-updates will be recovered in $f + x$ frames, and will not propagate to more subsequent frames. Figure 6.4 shows an example of the synchronization for multi-frame predictive parallel algorithm when the prediction span is 2 frames.

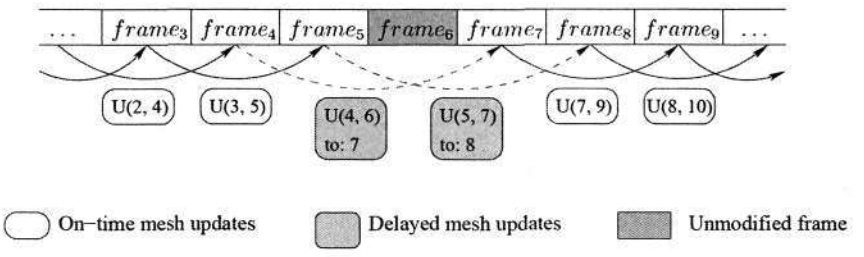


Figure 6.4: An example of the synchronization for multi-frame predictive parallel algorithm when the prediction span is 2 frames and $U(4, 6)$ is one frame late. The delay leads to $U(5, 7)$ one frame late and is recovered at $frame_8$.

If the network slowdown is not transient but lasts for a rather long time, such temporary synchronization will not solve the problem. In this case, we should adjust the prediction span f to adapt to the new network condition. The adjustment of the prediction span is discussed in Section 6.3.

Cares must be taken in dealing with the client-side triangle-budget control for the multi-frame predictive parallel strategy. In Chapter 4, we present the traversal management strategy for traversal control and triangle-budget control. In this strategy, the client sends FREE-MODE to the server when the number of triangles is below the budget and sends BUDGET-MODE when the selective mesh is likely to exceed the triangle-budget. In the case of f-frame prediction, the BUDGET-MODE signal sent to the server will make effects after f frames, so there will be f batches of mesh-updates still gen-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

erated with FREE-MODE. During this transition period, it is highly possible for the selective mesh to exceed the triangle-budget. To solve this problem, we define an alarm level $L = B - (2 \times N \times f)$, where B is the triangle-budget and N is the constraint on the maximum number of mesh-updates per traversal (refer to Chapter 4). We assume the mesh-update operators are manifold, which means each refinement operator introduces 2 triangles to the selective mesh. Whenever the selective mesh is likely to exceed the alarm level, the client sends the BUDGET-MODE signal to the server. In this way, by sending the BUDGET-MODE signal f frames earlier, the selective mesh will not violate the triangle-budget in the transition period.

6.3 Dynamic change of the prediction span

Choosing the appropriate prediction span f is very important for achieving the best possible rendering quality. If the prediction span is longer than need be, the rendering quality will be degraded by the extra unnecessary prediction error, as the prediction error grows with longer prediction span. If the prediction span is not long enough, many batches of mesh-updates will be late for their corresponding frames, causing a large number of unmodified frames. This again will lead to degraded rendering results.

The appropriate prediction span f should be just long enough to cover the time of the procedure of getting updates (PGU) for one frame. As in Section 6.1, we suppose T_g is the time for PGU, and $T_u + T_r$ is the time for the procedure of rendering (PRD), where T_u is the time for updating the mesh, and T_r is the time for rendering the mesh. We can get the appropriate prediction span by

$$f = \lceil T_g/(T_r + T_u) \rceil \quad . \tag{6.3.1}$$

Since the network condition changes from time to time, we can monitor T_r, T_u and T_g during run-time, and update f periodically to adapt to the possible changes of the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

network condition. The picture of parallelizing multiple PGUs for multiple frames is very much like juggling balls, except that the client is tossing update requests and catching the corresponding batches of mesh-updates. If the prediction span is f, at any point of time there will be f batches of mesh-updates pending (like f balls in the air). If we need a longer prediction span, which means the time cost of the PGU needs to be covered by more frames, we have to toss more update requests on the air to increase f, so that each following frames can catch a batch of mesh-updates at a constant rate. If we need a shorter prediction span, we have to take back some pending batches of mesh-updates without tossing further update requests in order to decrease f.

Suppose f is the latest computed prediction span, f' is the previous one used and the prediction span is required to be updated at $frame_i$. If $f > f'$, instead of sending just one update request for $frame_{(i+f')}$, $frame_i$ should send $f - f' + 1$ update requests for $frame_{(i+f')}$, $frame_{(i+f'+1)}$, $\ldots$, $frame_{(i+f)}$, respectively. The prediction span will be switched into f at $frame_{(i+1)}$, as shown in Figure 6.5(a). On the other hand, if $f < f'$, the frames from $frame_i$ to $frame_{(i+f'-f-1)}$ will only receive the corresponding batches of mesh-updates but will not send update requests for future frames. The prediction span will be switched into f at frame $frame_{(i+f'-f)}$ (see the example in Figure 6.5(b)).

6.4 Results

We test our multi-frame predictive parallel strategy on the same client and server computers, with the same four models over the same two scripted interaction paths as the experiments in Section 5.4. Other settings such as the client-side triangle-budget and the traversal management parameters are all the same as in Section 5.4. Also similar to the implementation in Section 5.4, the procedure of rendering (PRD) and the procedure of getting updates (PGU) are implemented as two threads on the client. Since

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

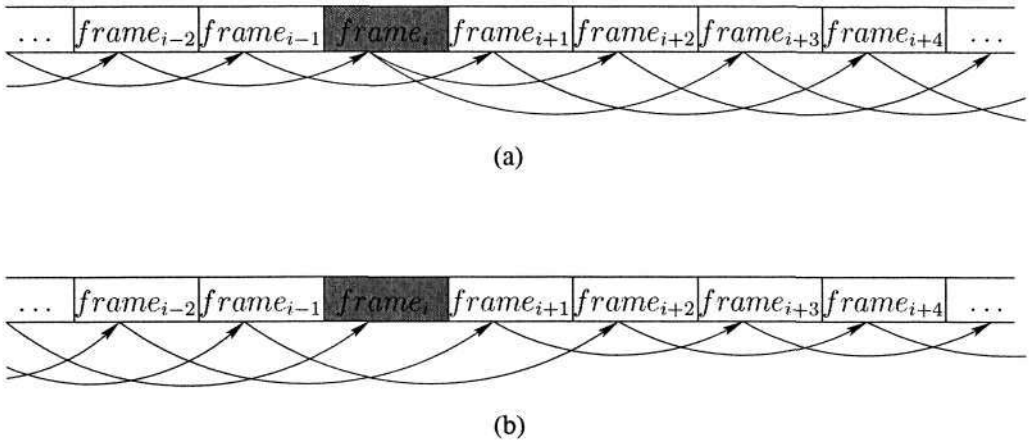


Figure 6.5: Dynamic change of the prediction span. (a) Change the prediction span from 2 to 3 at $frame_i$. (b) Change the prediction span from 3 to 2 at $frame_i$.

our experiment setup is on the campus LAN, we simulate the round-trip times (RTTs) of WAN environments by holding each packet for a specified time on the server. By this means, we can test the performance of our system in environments with different RTTs.

Table 6.1 shows the average frame times and accuracies of the multi-frame predictive parallel algorithm when RTT is $80ms$, $170ms$, $260ms$ and $350ms$ respectively. The corresponding prediction span (PS) used for each RTT is computed by Equation 6.3.1. The accuracy is measured the same way as in Section 5.4, namely the percentage of the correct triangles that the multi-frame predictive parallel algorithm can achieve in the selective mesh every 5 frames. All the data in the table are the averages over the first interaction path.

From Table 6.1, we can see that the multi-frame predictive parallel algorithm can achieve an interactive frame rate ($30fps$) with an acceptable accuracy on networks with relatively long RTTs, where the conventional sequential algorithm is impossible to perform interactively.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Table 6.1: The frame times and accuracies of the multi-frame predictive parallel algorithm for different RTTs.

		Bunny	Hand	Dragon	Happy
RTT: 80ms	Frame time	16.8	28.7	28.4	28.3
PS: 3-frame	Accuracy	89.1%	88.1%	89.3%	90.4%
RTT: 170ms	Frame time	16.7	28.8	29.0	28.0
PS: 6-frame	Accuracy	88.2%	87.7%	88.5%	89.0%
RTT: 260ms	Frame time	16.8	29.1	29.5	28.5
PS: 9-frame	Accuracy	84.3%	84.5%	84.8%	85.9%
RTT: 350ms	Frame time	16.7	29.6	30.0	28.5
PS: 12-frame	Accuracy	77.7%	77.6%	79.6%	80.6%

We illustrate the changes of the accuracy for each RTT over the two interaction paths in Figure 6.6, taking the Happy model as an example. Both Table 6.1 and Figure 6.6 prove that the accuracy decreases with longer RTT and longer prediction span. To analyze Figure 6.6 more deeply, we need to describe the two interaction paths in details. The first interaction path has three pieces of movements, all at uniform speed:

1. From frame 1 to frame 110, the viewpoint is moving towards the model in a straight line;
2. From frame 110 to frame 155, the viewpoint is scanning the model from the bottom to the top;
3. From frame 155 to frame 180, the viewpoint is scanning the model from the right to the left.

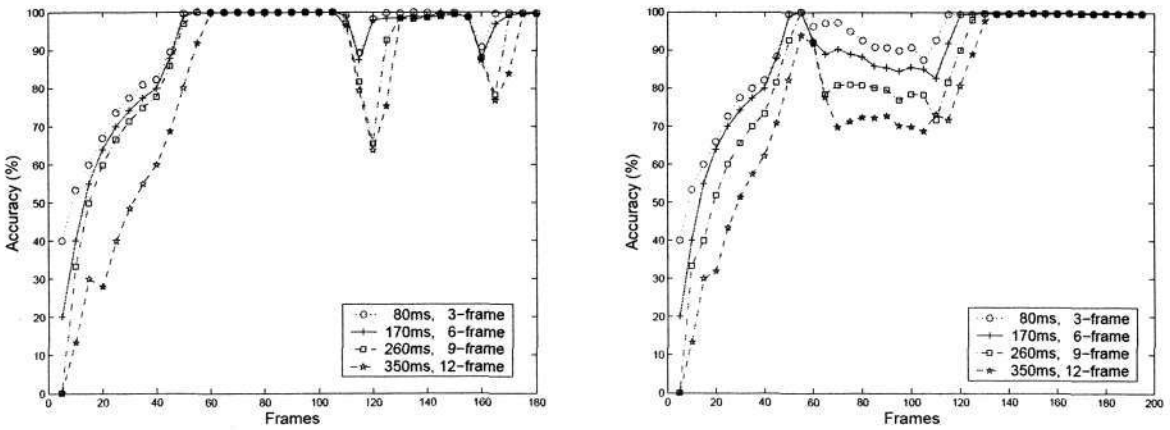
The second interaction path also has three pieces of movements:

1. From frame 1 to frame 60, the viewpoint is moving towards the model in a straight line;

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

2. From frame 60 to frame 110, the viewpoint is rotating around the model at increasing speed;

3. From frame 110 to frame 195, the viewpoint is rotating around the model at uniform speed.



(a) (b)

Figure 6.6: The accuracies for different RTTs and their appropriate prediction spans for the Happy model. (a) Over the first interaction path. (b) Over the second interaction path.

The first 50 frames is the warm-up stage, where the number of triangles is increasing from the base mesh to the triangle-budget. The low accuracy in this stage is due to the considerable number of unmodified frames, as we explained in Section 5.4. An important fact demonstrated by Figure 6.6 is that when the view-parameters are changing at uniform rate, high accuracies can be achieved for all four different RTTs. When there are abrupt changes of view-parameters (frame 110 and frame 155 in the first interaction path, refer to Figure 6.6(a)) or when the view-parameters are changing at variable speed (frame 60 to frame 110 in the second interaction path, refer to Figure 6.6(b)), the accuracies will drop. And when the changes of the view-parameters become steady again, the accuracies will go back to a high level within the time of a prediction span.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

The low accuracies of the selective meshes during the periods of non-uniform view changes are caused by the prediction errors of the predicted view-parameters, as the first-order motion model is used for prediction.

The accuracies of the predicted view-parameters can be improved by applying more sophisticated motion prediction strategies [14, 15], but at the cost of more expensive computation. The first-order motion model is a good trade-off for view-parameters prediction in our client-server based rendering system mainly for three reasons. First is its computational simplicity and its minimal memory requirement. Secondly our traversal constraint on mesh-change amount per frame (refer Chapter 4) can limit the influence of the large prediction error, so that even when the mesh is refined and simplified to the wrong view-parameters, the error is under control. This may lead to not-so-precise view-dependent refinement temporarily, but when the change of the view-parameters becomes more regular, the mesh will be more precisely refined to the current view because of the accumulation of the mesh-updates from several frames in one direction. Finally, when the interaction is fast, the inaccurate view-dependent refinement /simplification caused by the error of the predicted view-parameters will not impair the fidelity of the model since the human vision system cannot resolve much detail in fast moving objects [88].

Figure 6.7 shows the rendering results of the multi-frame predictive parallel algorithm at four different RTTs (Figure 6.7(b)–(e)), compared to the rendering result of the conventional sequential algorithm (Figure 6.7(a)). As we can see, the multi-frame predictive parallel algorithm can achieve an interactive frame rate without much quality degradation in the view-dependent rendering results, even at long RTTs.

We also perform the experiments for each RTT with different prediction spans. Our experiments prove that for a given RTT, the prediction span obtained by Equation 6.3.1 can provide the best overall accuracy. As an example, we show the plots (Figure 6.8)

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

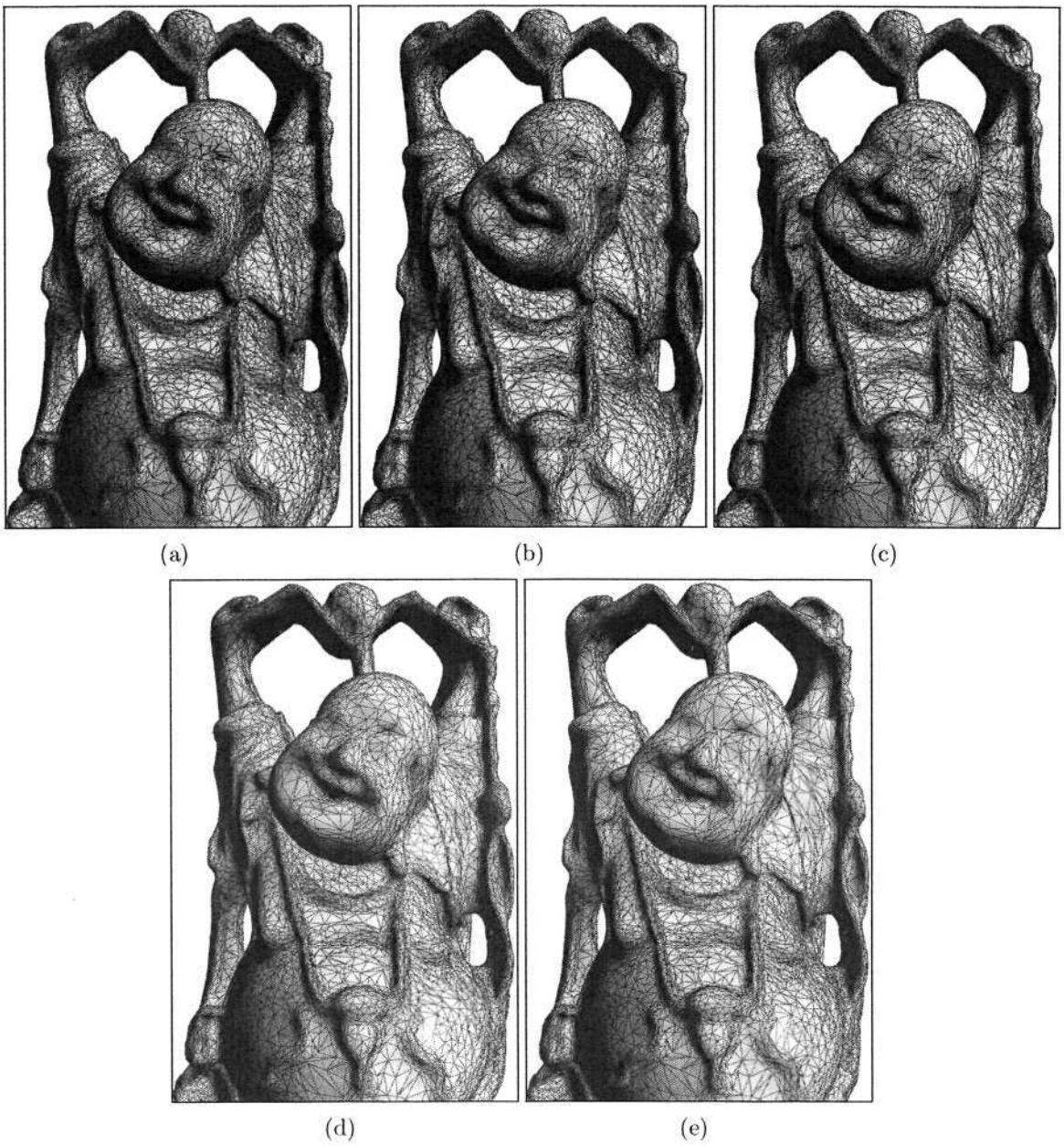
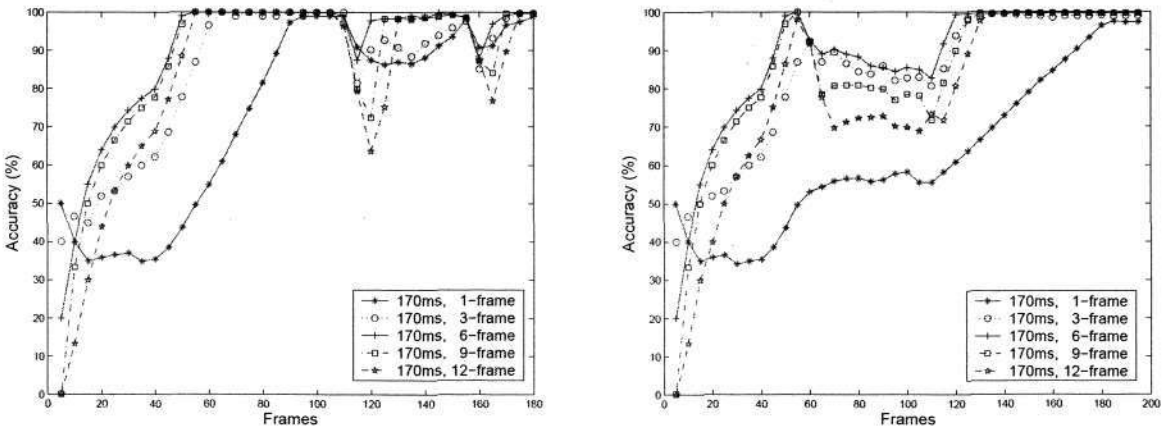


Figure 6.7: The 120th frame of the first interaction path for the Happy model. (a) Result by the conventional sequential method. (b)–(e) Results by the multi-frame predictive parallel method. (b) RTT is $80ms$ and PS is 3 frames. (c) RTT is $170ms$ and PS is 6 frames. (d) RTT is $260ms$ and PS is 9 frames. (e) RTT is $350ms$ and PS is 12 frames.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

for the Happy model with 170ms RTT and 5 different prediction spans. In this case, the sum of T_r and T_u is about 30ms. According to Equation 6.3.1, the appropriate prediction span should be 6 frames. As shown in Figure 6.8, the 6-frame prediction exhibits the best overall accuracy. The lower accuracies of 1-frame prediction and 3-frame prediction for 170ms RTT are caused by the large number of unmodified frames due to the large number of late batches of mesh-updates. When RTT is 170ms, the percentage of the unmodified frames is about 86.6% in 1-frame prediction and about 41.7% in 3-frame prediction. However in 6-frame prediction, the unmodified frames is only about 3.3%. The lower accuracies of 9-frame prediction and 12-frame prediction are due to the fact that the prediction accuracy decreases as the prediction span increases.



(a) (b)

Figure 6.8: The accuracies for different prediction spans when RTT is 170ms for the Happy model. (a) Over the first interaction path. (b) Over the second interaction path.

6.5 Summary

In this chapter, we extend the one-frame predictive parallel strategy in Chapter 5 to multi-frame predictive parallel strategy for WAN environments with longer round-trip times. We also deal with the changing condition of the network, mainly the variable

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

round-trip time, by dynamically adjusting the prediction span. Extensive experiments are performed in a simulated WAN environment. Our experimental results show that the multi-frame predictive parallel strategy can achieve interactive frame rates with acceptable accuracies on networks with relatively long round-trip times. The experiments also demonstrate that the appropriate prediction span can provide the best overall accuracy for a specific round-trip time. Our predictive parallel strategy not only can be used in client-server based view-dependent LOD rendering, but also can be generalized to other client-server based rendering systems where the models rendered on the client need to be updated according to the current view or position of the viewer, such as visibility-based rendering and image-based rendering.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Chapter 7

Visibility-based Scene Transmission

7.1 Introduction

Different from the single large models we investigate in earlier chapters, urban scenes represent another type of scenes, where the objects in the scenes are densely distributed and highly occluded. This means from any given viewpoint, only a small fraction of the scene is visible [94]. For this type of scenes, visibility culling is the most efficient technique to speed up the frame rate. Visibility information from visibility culling is also vital for selecting parts of the scene to be transmitted in a client-server based urban walkthrough system, where the whole city model is only stored on a server, and the client performs interactive walkthrough via network connection to the server, maintaining and rendering only the visible portion of the scene.

As we introduce in Chapter 2, visibility algorithms can be classified into *from-point visibility* and *from-region visibility*. In from-point visibility algorithms, visible sets are computed online with respect to the current viewpoint and have to be re-computed each time the viewpoint changes. From-point visibility algorithms are not suitable for client-server based walkthroughs, as even a slight change of the viewpoint on the client results in the visibility re-computation and transmission of the visible objects, giving

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

rise to unacceptable latency [25]. From-region visibility algorithms usually divide the view space into viewcells and conservatively compute Potentially Visible Set (PVS) [23] for each viewcell. They are more favorable for client-server based walkthroughs, not only because the visibility information is valid for more than one frame and hence alleviates the lag [25, 76, 75], but also in that from-region visibility has the predicting capability for prefetching strategies by using the visibility information from adjacent viewcells [23].

Considering the network latency, the graphical data must be retrieved from the server before the data are actually needed for rendering on the client. Thus the prefetching strategy is an important issue in client-server based walkthroughs. With from-region visibility, when the client is inside a viewcell, the graphical data needed for rendering are only the PVS of that viewcell. To make the walkthrough proceed smoothly, the graphical data of the PVS must be already on the client-side when they are to be rendered. That means, when the client is inside a particular viewcell, the PVS of some adjacent viewcells, which the client may visit next, should be prefetched.

Koltun *et al.* have proposed a *simple neighbor prefetch* (SNP for short) algorithm for their from-region visibility algorithm [76], where the viewcells are uniform grids. When the client is in a certain viewcell, the PVS of all the adjacent viewcells should be prefetched to the client. At the beginning of the walkthrough, the server sends the PVS of the initial viewcell and the PVS of all eight adjacent viewcells. When the client crosses a viewcell border, the PVS of three or five new neighboring viewcells must be transmitted to the client (see Figure 7.1). However this SNP algorithm is not very efficient when the client-side cache is restricted, due to the large amount of cache swaps incurred.

In this chapter, we present a *position-based neighbor prefetch* (PBNP for short) algorithm, which is an improvement of the SNP algorithm. The PBNP algorithm can

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

make more precise prediction about which viewcells will be needed in the near future according to the current position of the viewpoint of the client, and hence can prefetch the PVS of a smaller number of viewcells to the client. In the PBNP algorithm, the size of the required client-side cache is small, and the oscillation effect of continuous cache swaps is avoided. The simulation results show that the PBNP algorithm can significantly reduce the number of cache swaps and the number of transmitted objects, compared to the SNP algorithm.

To take advantage of the visibility coherence between neighboring viewcells, we also provide a *delta-transmission* algorithm to optimize the PVS transmission procedure by avoiding re-transmitting those objects that are already in the client-side cache. The client-side cache and the cache replacement strategy are designed to support the PBNP algorithm and the delta-transmission algorithm. Since our work focuses on PVS prefetch and transmission rather than visibility computation, we assume that the visibility pre-computation stage is finished and each viewcell has been associated with a potentially visible set.

The next section presents the position-based neighbor prefetch (PBNP) algorithm for PVS prefetching. Section 7.3 describes the delta-transmission algorithm. The client-side cache management strategy is provided in Section 7.4. The results of simulation experiments and the summary of this chapter are given in Section 7.5 and Section 7.6.

7.2 PVS prefetch

7.2.1 Problems with the SNP algorithm

In this subsection, we analyze the problems of the simple neighbor prefetch (SNP) algorithm proposed in [76]. The SNP algorithm is depicted in Figure 7.1. Each viewcell is represented by a pair of integers (*row-number, colum-number*). Initially the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

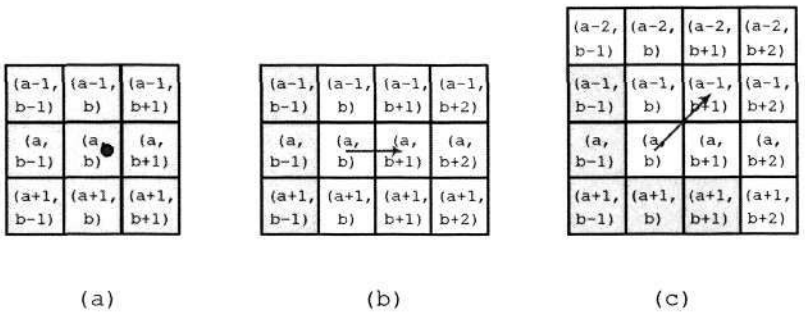


Figure 7.1: The SNP algorithm. (a) Current viewcell is viewcell (a, b). (b) Moving from viewcell (a, b) to viewcell (a, b+1) across the border causes 3 viewcells prefetched. (c) Moving from viewcell (a, b) to viewcell (a-1, b+1) through the corner point causes 5 viewcells prefetched.

viewpoint is in viewcell (a, b), as in Figure 7.1(a). Besides getting the PVS for the current viewcell, the client has to prefetch the PVS of all its eight neighbors, so the minimum size of the client-side cache is 9 entries, each entry for one viewcell. If the client moves into a new viewcell across a side border as in Figure 7.1(b), the PVS of three neighboring viewcells need to be prefetched. If the client moves into a new viewcell through a corner point as in Figure 7.1(c), the PVS of five neighboring viewcells need to be prefetched. For low-end clients, it is important to keep low memory requirement for caching. Therefore, we will restrict the cache size to the minimum size of 9 entries and study PVS prefetch with this cache-size restriction. Under this condition, the SNP algorithm has two problems.

1. Each time the client crosses the border of a viewcell, a number of existing cache entries have to be replaced by the newly prefetched ones. In the case of Figure 7.1(b), three entries have to be replaced, and in Figure 7.1(c) five entries have to be replaced. If the client crosses the same border back and forth continuously, the entries will be swapped out and in continuously, resulting in the *oscillation effect*. To avoid this, the cache size has to be enlarged and more client-side resource

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

will be consumed.

2. Taking Figure 7.1(c) as an example, the chance of the client moving directly from viewcell (a, b) to viewcell (a-1, b+1) is very slim, since they are only adjacent at a point. More likely, the client will move to viewcell (a-1, b+1) through a corner of viewcell (a-1, b) or viewcell (a, b+1). When the client is passing the corner of viewcell (a-1, b) or viewcell (a, b+1), the PVS of the neighbors of viewcell (a-1, b) or viewcell (a, b+1) are prefetched. Then after a very short time, the client moves into viewcell (a-1, b+1). However, the preceding prefetching transaction for viewcell (a-1, b) or viewcell (a, b+1) brings in the PVS of certain viewcell that is not a neighbor of viewcell (a-1, b+1), and therefore should not be in the cache (e.g. viewcell (a-2, b-1) or viewcell (a+1, b+2) in Figure 7.1(c)). The PVS for such viewcell will be swapped out of the cache by the prefetching transaction for viewcell (a-1, b+1) shortly after it is just swapped in and without being used, thus resulting in more unnecessary cache swaps.

7.2.2 The position-based neighbor prefetch algorithm

The reason behind the problems of the SNP is that the predication is too conservative, so that the prefetching area has to encompass all eight neighboring viewcells. This results in many unnecessary viewcells prefetched to the client-side cache. It can be improved by a more refined prediction. In our PBNP algorithm, the strategy is not to prefetch the PVS of a neighboring viewcell until the client shows a tendency to enter this viewcell, and hence to reduce the prefetching area. This tendency can be estimated from the current position of the client. Figure 7.2 illustrates the PBNP algorithm.

In PBNP, each viewcell is divided into 9 regions: middle (M), north (N), south (S), east (E), west (W), northwest (NW), northeast (NE), southeast (SE) and southwest

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

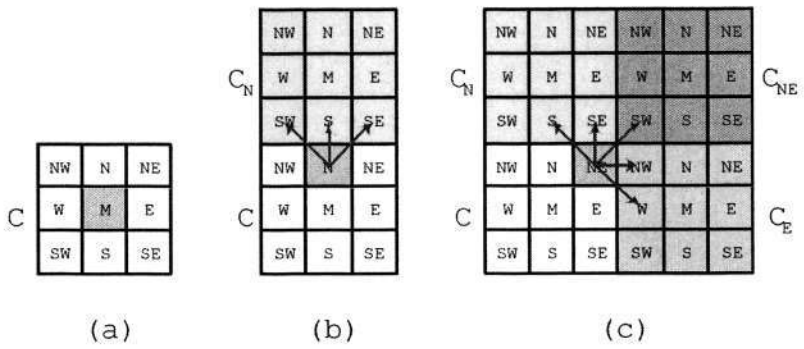


Figure 7.2: The PBNP algorithm. (a) Nine regions in a viewcell C. When the client is in the $C.M$ region, no neighboring viewcell will be prefetched. (b) When the client is in the $C.N$ region, only the viewcell of the north neighbor C_N will be prefetched. (c) When the client is in the $C.NE$ region, the viewcells of C_N, C_E and C_{NE} will be prefetched.

(SW), each region representing a tendency of entering one or more neighboring viewcells. We denote the current viewcell (viewcell (a, b) in Figure 7.1(a)) as C in Figure 7.2(a). Viewcell C also has eight neighboring viewcells, and we refer them as north neighbor (C_N), south neighbor (C_S), east neighbor (C_E), west neighbor (C_W), northwest neighbor (C_{NW}), northeast neighbor (C_{NE}), southeast neighbor (C_{SE}) and southwest neighbor (C_{SW}). In the rest of this chapter, we use notation $x.y$ to present a region y in a viewcell x, where $y \in \{M, N, S, E, W, NW, NE, SE, SW\}$. For example, $C.N$ represents the N region in viewcell C.

Unlike the SNP algorithm that checks for a prefetching transaction when the client is crossing a viewcell border, PBNP checks for a prefetching transaction when the client is crossing a region border inside a viewcell. Table 7.1 shows the relationship of a region and its prefetching area, which is the set of neighboring viewcells (including the current viewcell C) that must be in the cache. This table can also be represented by a function $h(R)$. Given a current region R, $h(R)$ gives the corresponding prefetching area that must be in the cache.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Table 7.1: Regions and their corresponding prefetching areas.

R: Current region in viewcell C	$h(R)$: Prefetching area that must be in cache
$C.M$	$\{C\}$
$C.N$	$\{C, C_N\}$
$C.S$	$\{C, C_S\}$
$C.W$	$\{C, C_W\}$
$C.E$	$\{C, C_E\}$
$C.NW$	$\{C, C_N, C_W, C_{NW}\}$
$C.NE$	$\{C, C_N, C_E, C_{NE}\}$
$C.SE$	$\{C, C_S, C_E, C_{SE}\}$
$C.SW$	$\{C, C_S, C_W, C_{SW}\}$

When the client is in region $C.M$ (Figure 7.2(a)), it shows no tendency of entering any neighboring viewcell in the near future, so no neighbor needs to be prefetched. In Figure 7.2 (b), the client is region $C.N$, which implies that the client would probably enter the north neighbor soon, so we need to prefetch the PVS for C_N. When the client is in region $C.NE$ (Figure 7.2(c)), there are three potential next viewcells: C_N, C_E and C_{NE}. The PVS for these three neighboring viewcells must be prefetched.

We now take a deeper look at how PBNP solves the two problems with SNP.

1. First, PBNP can avoid the oscillation effect with the same minimum cache size of 9 entries as in SNP. The key point to avoid the oscillation effect caused by continuously crossing the same border back and forth is that the cache must be able to contain the union of the prefetching areas for the two regions on both sides of the border. Formally, for any two adjacent regions R_1 and R_2, if $\mid h(R_1) \cup h(R_2) \mid \leq cache_size$, the oscillation effect will not happen. We can distinguish two different adjacency relationships in the PBNP algorithm.

   - In the first case, neither of the two regions is a corner region (region NW,

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

NE, *SW* and *SE*), e.g. $C.N$ and $C_N.S$ in Figure 7.2 (b). In this case, the union of the prefetching areas for this pair of adjacent regions has 2 viewcells. For example, $h(C.N) = \{C, C_N\}$ and $h(C_N.S) = \{C, C_N\}$, so $\mid h(C.N) \cup h(C_N.S) \mid = 2$.

- The other case is that one or both of the two regions are corner regions, e.g. $C.N$ and $C.NE$, or $C.NE$ and $C_N.SE$ (see Figure 7.2). In this case, the union of the prefetching areas for this pair of adjacent regions has 4 viewcells. For example, for $C.N$ and $C.NE$, $h(C.N) = \{C, C_N\}$ and $h(C.NE) = \{C, C_N, C_E, C_{NE}\}$, so $\mid h(C.N) \cup h(C.NE) \mid = 4$. For $C.NE$ and $C_N.SE$, $h(C.NE) = h(C_N.SE) = \{C, C_N, C_E, C_{NE}\}$, so again $\mid h(C.NE) \cup h(C_N.SE) \mid = 4$.

In PBNP, since the maximum number of viewcells in the union of the prefetching areas for two adjacent regions is only 4, no oscillation effect will happen when the cache size is restricted to 9 entries. However, in SNP, the union of the prefetching areas for two adjacent viewcells has 12 (see Figure 7.1(b)) or 14 (see Figure 7.1(c)) viewcells, which exceeds the cache size, and therefore causes the oscillation effect.

2. PBNP can also avoid the unnecessary cache swaps during corner crossings. For example, when the client moves from region $C.M$ to region $C.NE$ through a corner of region $C.N$ or $C.E$, there will be no extra cache swaps, as the PVS transmitted by the prefetching transaction for region $C.N$ or $C.E$ is also part of the prefetching transaction for region $C.NE$, i.e. $h(C.N) \subset h(C.NE)$ and $h(C.E) \subset h(C.NE)$.

7.3 Delta-transmission

Each viewcell can be identified by a unique viewcell ID. Also each object in the scene can be identified by a unique object ID. A convenient way to store PVS information is

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

to use a boolean table T, with each row corresponding to one viewcell and each column corresponding to one object [126]. The element of the table is denoted by T_{ik}, where i and k are the IDs for viewcells and objects respectively. If object k is at least partially visible from some point in viewcell i, we have $T_{ik} = TRUE$; if object k is invisible from any point in viewcell i, we have $T_{ik} = FALSE$. We use S_i to denote the PVS indices for viewcell i, which is the row in the PVS table for viewcell i, i.e. $S_i = (T_{i1}, T_{i2}, \ldots, T_{in})$, where n is the total number of objects in the scene.

Because of the spatial coherence, the PVS for the neighboring viewcells share a lot of local similarity [94, 126]. In other words, the neighboring viewcells tend to have a set of visible objects in common. By taking advantage of this property, we can avoid re-transmitting the overlapped set of visible objects that are already on the client-side. Figure 7.3 illustrates an example. Objects 1, 2, 3, 4 and 8 are the potentially visible objects for viewcell C_1, and objects 1, 2, 3, 4, 5 and 6 are the potentially visible objects for C_1's neighbor C_2. The overlapped set includes objects 1, 2, 3 and 4. When the viewpoint is in C_1 and objects 1, 2, 3, 4 and 8 are in the client-side cache, to prefetch C_2's potentially visible objects, we only need to transmit the non-overlapped set of objects 5 and 6 to the client.

We propose a delta-transmission algorithm to identify and transmit only the non-overlapped set. Suppose viewcell i is the current viewcell, and its potentially visible objects are already on the client. Viewcell j is the viewcell of which the potentially visible objects should be prefetched. We define the non-overlapped visible set ΔS that should be transmitted as $\Delta S = S_j - (S_j \cap S_i)$, where S_i and S_j are the PVS indices for viewcell i and viewcell j respectively. The set ΔS can be computed easily by utilizing the boolean characteristic of the PVS table.

The routine of delta-transmission is accomplished by the coordination of the client and the server, as depicted in the example in Figure 7.4. First, the client sends the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

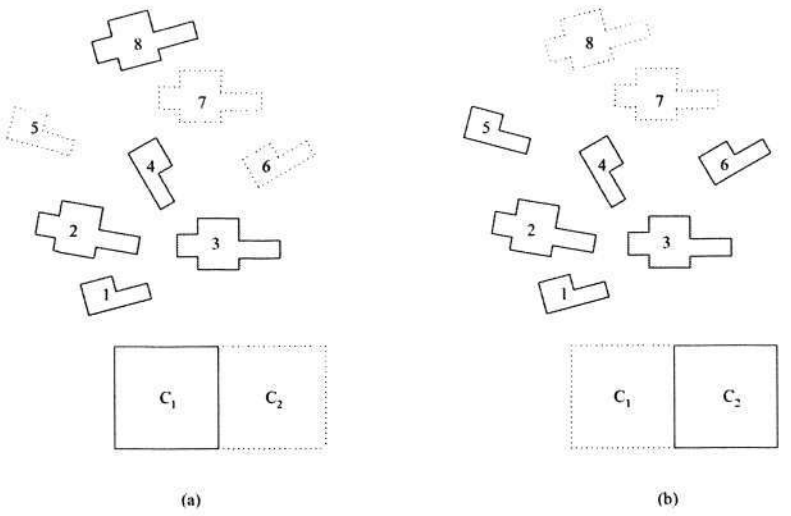


Figure 7.3: The PVS of the neighboring viewcells share a lot of local similarity. (a) PVS for viewcell C_1. (b) PVS for viewcell C_2.

request to the server, asking for the PVS for viewcell j. Then the server sends S_j, which is the row in the boolean PVS table corresponding to viewcell j, to the client. A copy of S_i for the current viewcell i is maintained on the client-side, so that ΔS can be computed by a boolean **AND** operation and a boolean **XOR** operation (see Figure 7.4). Then the client sends ΔS to the server and the server will transmit the graphical data for ΔS to the client.

More commonly, the client will cache the PVS of more viewcells than just the current viewcell, and the PVS of more than one viewcell will be requested in one prefetching transaction. Suppose viewcells i_0, i_1, ..., i_m are the viewcells of which the potentially visible objects are cached on the client besides current viewcell i, and viewcells j_0, j_1, ..., j_l are the viewcells of which the potentially visible objects are requested. In this case, the set ΔS is computed by $\Delta S = S_J - (S_J \cap S_I)$, where S_I is the union of the PVS of all cached viewcells, i.e. $S_I = S_i \cup S_{i_0} \cup S_{i_1} \cup \ldots \cup S_{i_m}$, and S_J is the union of the PVS of all requested viewcells, i.e. $S_J = S_{j_0} \cup S_{j_1} \cup \ldots \cup S_{j_l}$.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

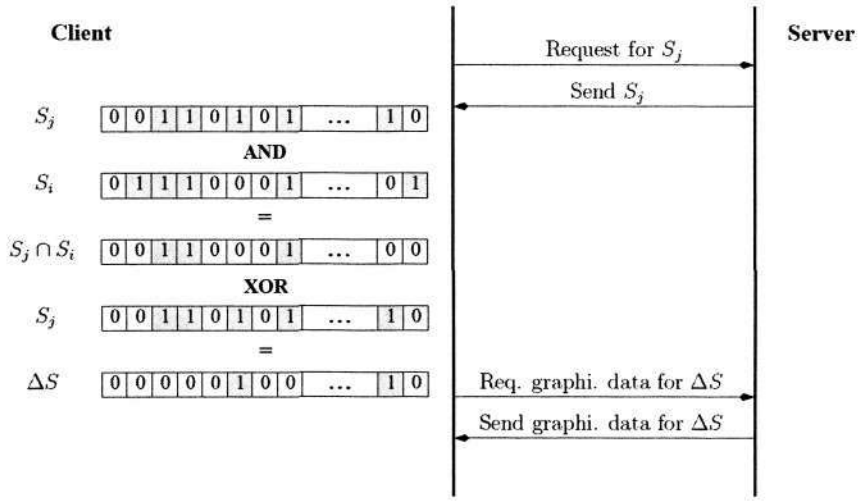


Figure 7.4: Delta-transmission routine for the client and the server.

7.4 Client-side cache management

In this section we discuss the client-side cache that supports the PBNP algorithm and the delta-transmission algorithm. The cache comprises two parts, the local PVS table and the local graphical database. The local PVS table contains the PVS indices for the current viewcell and the prefetched viewcells. As mentioned in Section 7.2, the local PVS table is restricted to 9 entries. The local graphical database stores the graphical information (geometries, textures, etc.) of the objects that are cached on the client. The local graphical database is consistent with the local PVS table.

Before the client renders a new frame, the new position of the viewpoint is checked. If the new position is in a new region, the client must do the following tasks:

1. Compute which neighboring viewcells should be prefetched;

2. Start a prefetching transaction to transmit the graphical data of the PVS for those neighboring viewcells if they are not in client-side cache;

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

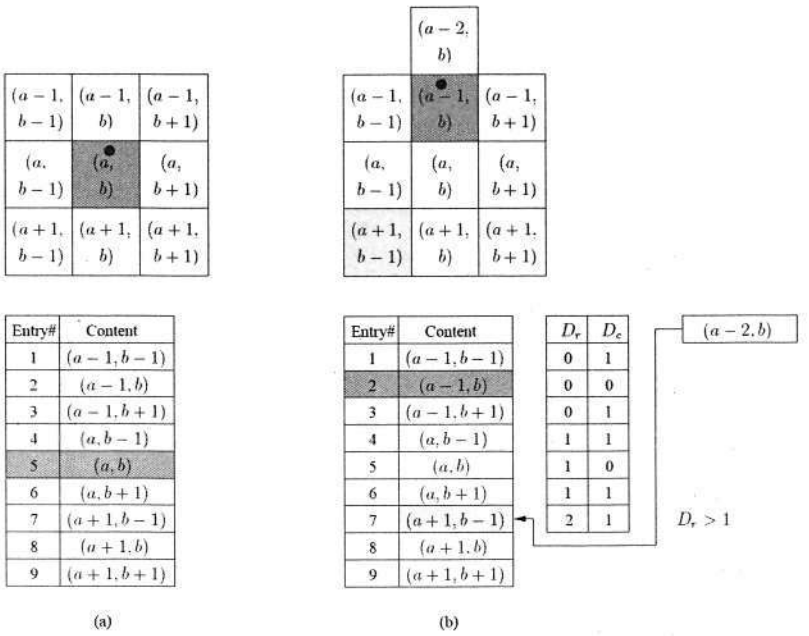


Figure 7.5: Cache entry replacement strategy. (a) The cache is full with 9 entries, one for current viewcell (a, b) and eight for its neighboring viewcells. (b) Current viewcell moves to $(a-1, b)$ and viewcell $(a-2, b)$ is prefetched. The entry with $D_r > 1$ or $D_c > 1$ is replaced by the new viewcell $(a-2, b)$.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

3. Update the cache.

Task 1 and 2 are done by the PBNP algorithm (Section 7.2.2) and delta-transmission algorithm (Section 7.3). Now we look into task 3.

After each delta-transmission transaction, the new graphical data are added to the local graphical database and the new PVS indices for the newly prefetched viewcells should be added to the local PVS table. When the local PVS table is full, some old entries have to be replaced. Our entry replacement strategy for the 9-entry cache is that any viewcell entry that is not current viewcell's neighbor can be replaced. To determine whether a viewcell in an old entry is the current viewcell's neighbor, we can simply check the distance between this viewcell and the current viewcell. In a uniform grid partition, the distance can be expressed by the absolute value of the difference of the row numbers (D_r) or the difference of the column numbers (D_c) of these two viewcells. If $D_r > 1$ or $D_c > 1$, the viewcell is not the current viewcell's neighbor and the entry can be replaced. The strategy is illustrated by an example in Figure 7.5.

When deleting an old cache entry from the local PVS table, we also have to delete the relevant graphical data to make the local PVS table and the local graphical database synchronized, and also to limit the local memory consumption by the cache. Suppose $d_0, d_1, \ldots, d_s$ are the viewcells whose entries should be deleted from the local PVS table and $c_0, c_1, \ldots, c_t$ are the viewcells whose entries should be maintained, then the set of objects ΔS_d that should be deleted from the local graphical database is decided by $\Delta S_d = S_d - (S_d \cap S_c)$, where $S_d = S_{d_0} \cup S_{d_1} \cup \ldots \cup S_{d_s}$ and $S_c = S_{c_0} \cup S_{c_1} \cup \ldots \cup S_{c_t}$.

7.5 Results

We build a simulation system to compare our position-based neighbor prefetch algorithm (PBNP) with the simple neighbor prefetch algorithm (SNP).

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Since the from-region visibility algorithm for generating the PVS table is not a contribution of our work, the PVS table used in our experiments is generated based on the data provided by Cohen-Or *et al.* [24] and Nadler *et al.* [94]. The simulated city model (refer to Figure 7.6) in our experiments is similar to the one in [24], in terms of density and the size of the buildings. It consists 900 buildings. The base of each building is a $100m \times 100m$ square. The streets between the buildings are $60m$ wide, divided into $20m \times 20m$ viewcells. For the similar city model in [24], when the viewcells are $20m \times 20m$, the potentially visibility set for each viewcell is about 5% of the whole scene. Actually, culling off above 95% of the scene for each viewcell on a dense city model can be achieved by many occlusion culling algorithms [32, 134, 75]. Also, according to the cell-to-cell coherence analysis in [94], for a city model like ours, the potentially visible sets of two adjacent viewcells have about 60% overlap. Thus we generate the PVS table for this simulated city model with the assumptions that the potentially visible objects for each viewcell are 5% of the whole scene and that two adjacent viewcells share 60% common visible objects.

Two walking routes, each of which is 3000 steps, are generated with each step $4m$ in length, shown in Figure 7.6. The first route is randomly generated. Many of the movements involve crossing the same border back and forth. The second route is very regular, consisting of seven pieces of straight lines connected with six right-angle turns.

We simulate the walking along the two routes with two different prefetching algorithms, the SNP algorithm and our PBNP algorithm. In both the SNP simulation and PBNP simulation, we use a client-side cache of the same size, i.e. 9 entries for the local PVS table. Delta-transmission algorithm (Section 7.3) and the same cache replacement strategy (Section 7.4) are used in both simulations. The aim is to compare the number of prefetching transactions, the number of cache swaps and the number of objects transmitted throughout the walking routes.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

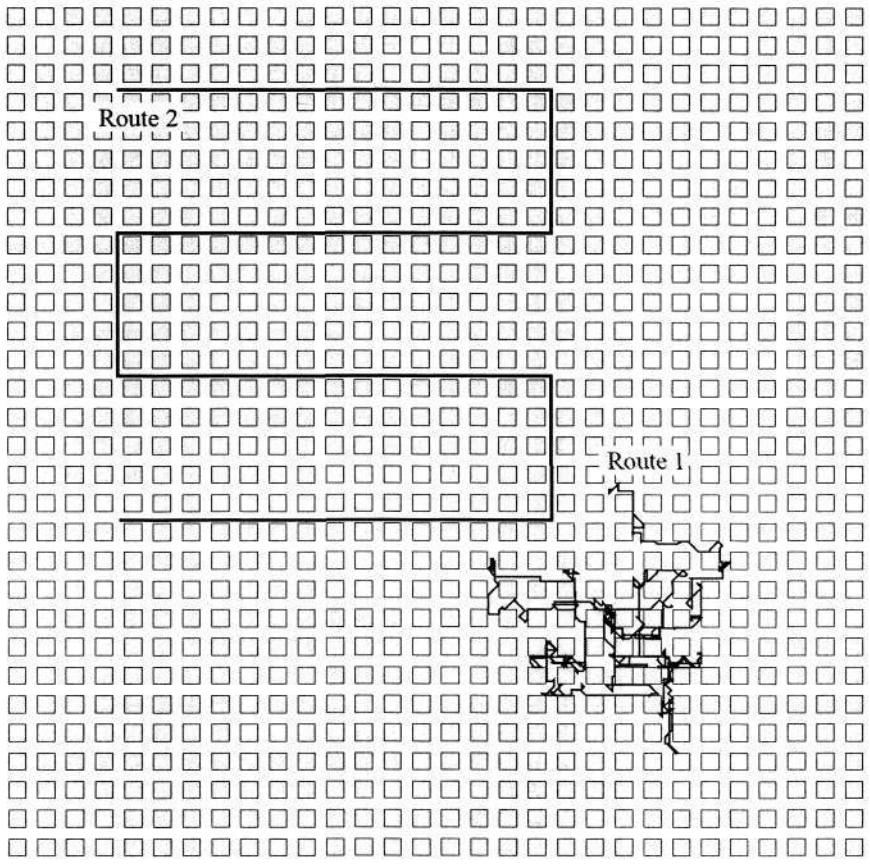


Figure 7.6: The top view of the simulated city model and the two walking routes.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Table 7.2: The number of prefetching transactions and the number of cache swaps for the two algorithms on the two walking routes.

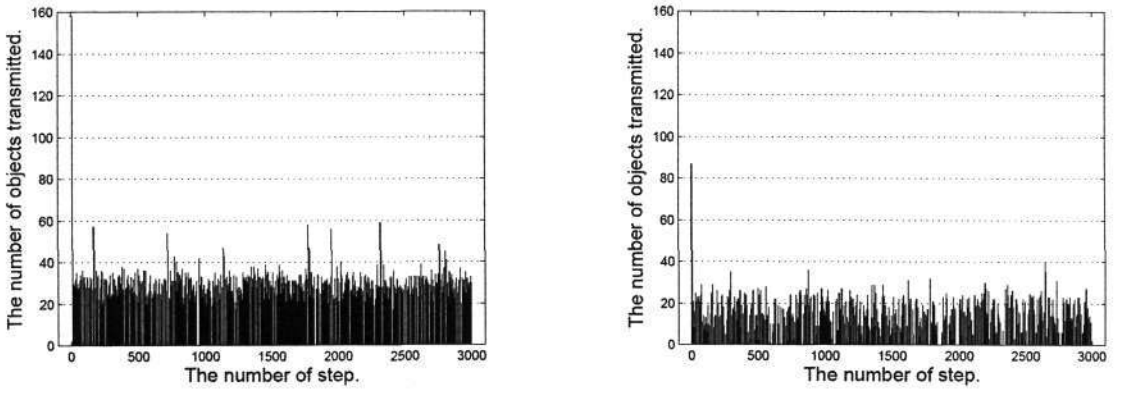
	Route 1		Route 2	
	SNP	PBNP	SNP	PBNP
Num. Pref. Trans.	585	487	600	600
Num. Cache Swaps	1785	876	1797	828

Table 7.2 compares the number of prefetching transactions and the number of cache swaps for the two algorithms. For random walks, which often have many back-and-forth movements, like the first walking route, PBNP can obviously reduce the number of prefetching transactions needed. This is because PBNP can eliminate the oscillation effect described in Section 7.2. For the regular walks without back-and-forth movements, like the second walking route, the PBNP has the same number of prefetching transactions as the SNP. For both walking routes, the number of cache swaps are reduced by half with the PBNP algorithm.

The number of transmitted objects at each step for the two algorithms is depicted in Figure 7.7 and Figure 7.8, for the first route and the second route respectively. In the SNP simulation, the average number of transmitted objects per transaction is 30 for the first route (Figure 7.7(a)) and 29 for the second route (Figure 7.8(a)). By contrast in the PBNP simulation, for both two walking routes, the average number of transmitted objects per transaction is only 16 (Figure 7.7(b) and Figure 7.8(b)). The efficiency of the PBNP algorithm, in terms of smaller number of cache swaps and smaller number of transmitted objects, is due to the fact that PBNP only needs to transmit the PVS for one or three viewcells per prefetching transaction, however the SNP algorithm has to transmit the PVS for three or five viewcells.

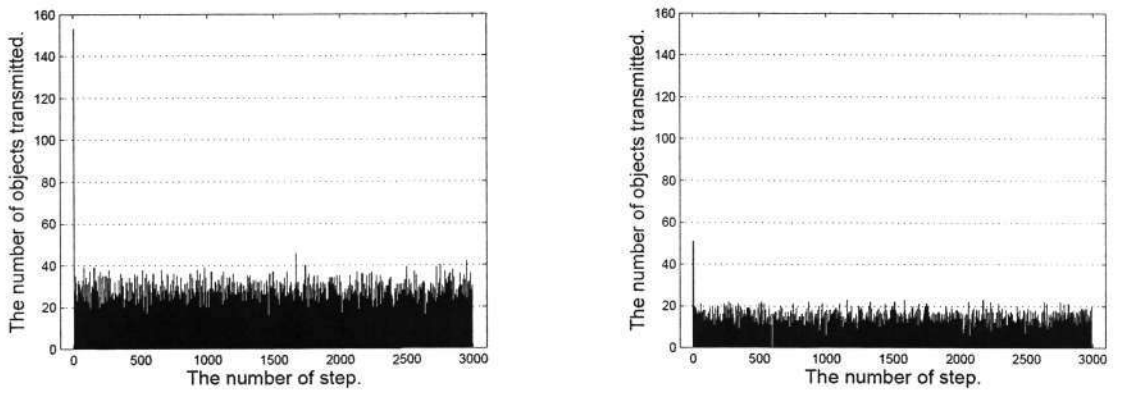
Also, in the SNP, the first prefetching transaction transmits 158 objects on the first route and 153 objects on the second route. In the PBNP, only 87 objects and 51

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library



(a) (b)

Figure 7.7: The number of transmitted objects at each step on the first route. (a) For the SNP algorithm. (b) For the PBNP algorithm.



(a) (b)

Figure 7.8: The number of transmitted objects at each step on the second route. (a) For the SNP algorithm. (b) For the PBNP algorithm.

objects are transmitted in the first prefetches on the first route and the second route respectively. This implies that PBNP algorithm takes a much shorter start-up time.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

7.6 Summary

We have described a position-based neighbor prefetch (PBNP) algorithm, which can utilize the preprocessed from-region visibility information more efficiently than the simple neighbor prefetch (SNP) algorithm in client-server based walkthroughs. We demonstrate that our PBNP algorithm can avoid the problems with the SNP algorithm when the client-side resources are constrained. To optimize the prefetching transmission procedure, we provide a delta-transmission algorithm to decrease the data volume for transmission. We also propose a cache management strategy for the client-side cache with restricted size to support the PBNP algorithm and the delta-transmission algorithm.

Some occlusion culling algorithms [109, 32] use more adaptive space partition methods, such as quadtrees and octrees, to organize the viewcells. One possible improvement of our work in this chapter is to extend the PBNP algorithm to more adaptive viewcells than the uniform grids. Another important factor that needs to be considered in the future works is the navigation speed. More sophisticated prediction strategy based on both current position and speed can provide more accurate and timely PVS prefetch for navigation with highly variable speeds.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Chapter 8

Conclusion

8.1 Summary

Client-server based rendering systems are of great interest in many fields. The big challenges come from two aspects. The first is the fast-increasing complexity of the available models. The high-definition models from the 3D laser scanning and the advanced modeling software are usually way beyond the capabilities of the common PC clients and the common networks. On the one hand, rendering such large models straightforwardly and interactively on the common PCs is almost impossible. It is even more challenging for low-capacity clients such as handheld devices. On the other hand, downloading a large model over the common networks will take a rather long time. We need an approach that supports rendering partially downloaded models, and also allows progressive and interruptible transmission. Many rendering acceleration techniques such as level-of-detail (LOD), image-based rendering and visibility culling can be extended for selective transmission of the scene. More particularly, view-dependent LOD technique provides a good solution for rendering a single highly-detailed object with both fidelity and interactivity in client-server based systems, and from-region visibility culling technique offers the predicting capability for prefetching potentially visible objects in remote walkthroughs of scenes with densely distributed objects.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

The second challenge is to overcome the network latency, especially the latency caused by the round-trip time of the network. The network latency is the main obstacle to achieve interactive frame rate in client-server based systems. Since caching techniques and compression techniques that have been attempted in previous works have their limitations when dealing with low-capacity clients and long round-trip times, we seek alternative solution to the problem of network latency.

In this thesis we demonstrate:

- A decoupled view-dependent LOD scheme for client-server based rendering systems.
- A predictive parallel strategy to overcome the network latency for the client-server based view-dependent LOD rendering.
- A PVS prefetching algorithm for visibility-based scene transmission.

8.1.1 View-dependent LOD for client-server systems

Most of the previous view-dependent LOD schemes are not naturally suitable for client-server based systems, especially not for low-capacity clients. In this thesis, a new view-dependent LOD scheme especially favorable to the client-server systems is proposed.

1. We have managed to decouple the selective mesh and the multiresolution hierarchy by encoding the dependencies among the mesh-updates in a DAG, so that the selective mesh and the multiresolution hierarchy residing on the client and the server respectively are independent of each other. Also, in order to represent the selective mesh in a format that is optimal for rendering and memory-efficient, we define a new legality condition for the vertex-split operators, which ensures

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

the partial order of the mesh-updates so as to remove the need of the adjacency queries on the selective mesh.

2. As part of our view-dependent LOD scheme, we present an efficient algorithm to build the DAG hierarchy of vertex-splits from a progressive mesh (PM) representation. One problem of the DAG-based multiresolution hierarchy is the redundant edges, which will cost extra memory and extra run-time operations. Building a DAG without any redundant edges from a PM requires an $O(n^2)$ time complexity, and thus is not feasible for practical use. Instead we present an algorithm that can eliminate more than 80% redundant edges with an $O(n)$ time complexity.

3. The problem of run-time incremental traversal of the DAG is also addressed. Since the amount of mesh-updates will affect the rendering frame rate, we propose a traversal management strategy for mesh-update control and triangle-budget control. Three different traversal algorithms are compared under this traversal management strategy.

8.1.2 Predictive parallelism to overcome network latency

We present our solution to the problem of network latency in the context of client-server based view-dependent LOD rendering, but this solution can be extended to other client-server based rendering systems.

1. We propose to run the rendering process and the multiresolution hierarchy traversal process in parallel on the client and the server respectively, using the rendering time of one or more frames to cover the round-trip time of the network. Instead of requiring the server to refine the selective mesh according to the current view-parameters, the client predicts the view-parameters for the next frame or several frames ahead and asks the server to transmit the refinement and simplification

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

operators for the future frame. In the LAN environment, where the round-trip time is very short, the rendering time of one frame is usually enough to cover the network round-trip time and the hierarchy traversal time on the server. Our experiments of the one-frame predictive parallel algorithm in the LAN environment have shown promising results. The client can achieve interactive frame rate and the rendering quality is very close to that of the sequential algorithm.

2. In the WAN environments, usually the rendering time of multiple frames is required to cover the relatively long round-trip time. We extend the one-frame predictive parallel algorithm for the LAN environments to the multi-frame predictive parallel algorithm for the WAN environments. Also, since the network condition of a WAN environment is more likely to change over time, we provide a mechanism to dynamically adapt the prediction span according to the current network condition.

8.1.3 Prefetching precomputed PVS

For client-server based walkthroughs in a densely occluded scene with a large collection of objects, e.g. a city model, visibility culling techniques provide the most efficient way of selective scene transmission. To improve the simple neighbor prefetch algorithm (SNP) presented in [76], we propose a position-based neighbor prefetch algorithm (PBNP) to predict and prefetch the precomputed Potentially Visible Sets (PVS) of neighboring viewcells that will be needed in the near future according to the current view position of the client. Our simulation results demonstrate that our PBNP algorithm can significantly reduce the number of transmitted objects from the server and the number of cache swaps on the client, in comparison with the SNP algorithm in [76]. To optimize the prefetching transmission procedure, we provide a delta-transmission algorithm to decrease the data volume of transmission. We also propose a cache man-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

agement strategy for client-side cache with restricted size to support PBNP algorithm and delta-transmission algorithm.

8.2 Discussion

In this section, we discuss some possible improvements for our work and future research directions in the area of client-server based rendering.

8.2.1 View-dependent rendering for multiple clients

The target application of our view-dependent LOD framework is online viewing of large triangle models for education, art appreciation or industry display. It is often desirable for such systems to allow multiple users to view different or the same models concurrently.

The client-server based view-dependent LOD rendering framework presented in this thesis can be easily extended to support multiple clients. Since the selective mesh is decoupled from the multiresolution hierarchy, most of the data in the multiresolution hierarchy are *user-independent* and hence can be shared among multiple concurrent users. The *user-dependent* data, including the counters of the nodes in the DAG hierarchy and the two lists of indices to the nodes in R_c and L_s (see Chapter 4), will be maintained for each user respectively.

When a model is required by the first user, the multiresolution hierarchy of the model is loaded and initialized in a block of shared memory on the server. As user-independent data, the multiresolution hierarchy is read-only. Then a new thread is created for the user, and the user-dependent data are initialized. This thread is in charge of performing the multiresolution hierarchy traversal, maintaining the user-dependent data and network communication for the user. For the subsequent users who also

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

require to view this model, a new thread and its user-dependent data will be created for each.

However, how to increase the space efficiency for the user-dependent data of the multiple threads on the server and how to schedule the multiple threads in order to achieve best overall performance among all the clients will be interesting topics and are worth further research study.

8.2.2 Out-of-core simplification and rendering

In the presented experiments in this thesis, the largest model is the Happy model, which is a little bit more than one million triangles. We do not test larger models because we use the quadric error algorithm [46] to simplify the mesh, which requires the whole original model to be loaded into the main memory. Thus the memory size of the computer limits the size of the model that can be processed. However, many large models available nowadays have several million or even several hundred million triangles. These gigantic polygon models can no longer be completely loaded into the main memory of common computers. To break the constraints of memory capacities, some recent research works are carried out on out-of-core simplification and visualization of very large models.

In the out-of-core simplification algorithms, the entire model to be simplified is stored in the secondary memory, or the hard disk. Different types of approaches can be found in the literature to perform the simplification out-of-core. The first type partitions the model into separate parts that are small enough to be loaded into the main memory and simplified with an in-core simplification algorithm. The boundaries of each part are remained and the simplified parts are stitched together as a new part for further simplification [65, 101]. The second type is based on vertex-clustering, which reads in one triangle of the model at a time, performs clustering using only limited connectivity

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

information and outputs the associated simplification details in files that will be later used to construct the multiresolution hierarchy [80, 83, 81]. Another type of out-of-core simplification externally sorts all the edges of the model and loads only a portion of the model into the main memory at a time for edge-collapse based simplification according to the order of the sorted edges [34].

During out-of-core view-dependent visualization, the entire multiresolution hierarchy is stored on disk. Only the parts that are necessary to render the current frame and those that are likely to be needed in the near future are kept in the main memory. There are two different ways to handle the data-swapping between the main memory and the disk. The first can be called explicit paging. The multiresolution hierarchy is composed of blocks of refinement details, so that data-swapping is on block-basis. The blocks are loaded in or deleted from the main memory explicitly according to the requirement of the view-dependent rendering [34, 28]. The other data-swapping method accesses the external memory through the system memory mapping functions, relying on the operating system to load in the data when needed [101, 82, 81, 20]. This method requires optimized data layout to improve memory coherence.

Currently, the DAG multiresolution hierarchy in our framework is created in a preprocess stage using an in-core algorithm. Adapting the DAG multiresolution hierarchy to out-of-core algorithm will be an important extension of our works. The out-of-core simplification algorithm designed for progressive mesh [66, 101] can be incorporated, as our DAG hierarchy is constructed from a progressive mesh representation. The out-of-core version of our multiresolution hierarchy can be a tree of blocks of refinements, where each block contains a DAG hierarchy of vertex-splits as in our current framework. The degree of the tree is dependent on the spacial division and stitching method used in the out-of-core simplification procedure. Instead of using the granularity of blocks of refinements in the run-time view-dependent rendering as in [101], we can keep the

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

fine granularity of triangles on the client by using the current DAG traversal method. Although the vertex-splits are grouped into blocks of DAGs, the concept of "cut" still holds on the entire hierarchy, as we can consider the entire hierarchy as a big DAG. On the server, the necessary portions of the multiresolution hierarchy to perform the traversal are swapped in and out of the main memory on block-basis.

With out-of-core visualization, the server may scale to a larger number of simultaneous users and accommodate more concurrent models to be viewed. However, the out-of-core algorithm will also make multi-user systems more complicated, especially in that the memory-swapping on the server has to take into consideration the needs of multiple clients.

8.2.3 Integrating view-dependent LOD and visibility culling

Level-of-detail techniques and visibility culling techniques both have their own limitations and show complementary advantages at the same time. LOD techniques alone have difficulty with high-depth-complexity scenes, while visibility culling techniques alone cannot sufficiently reduce the graphics load when a large scale of the scene is actually visible [8]. Therefore, these two techniques should be combined to achieve better performance.

Since the beginning of the view-dependent LOD techniques, view-frustum culling and back-face culling have become the standard view-dependent error criteria for view-dependent LOD schemes [87, 63]. Later, occlusion culling is also integrated into the view-dependent LOD frameworks [36, 50]. On the other hand, static LOD techniques have also been used as tools for occlusion culling algorithms. Simplified versions of occluders have been used to reduce the computation for occlusions [77, 56].

However, integrating view-dependent LOD into rendering complex scenes with densely

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

distributed objects is still a rarely explored area. In many rendering systems of very large and complex environments that use hybrid rendering acceleration techniques [3, 4, 8, 51, 117, 16], only discrete level-of-detail or hierarchical level-of-detail (HLOD) is applied. Continuous LOD has been attempted in multi-object complex scenes [49], but the use of view-dependent LOD still remains in the cases where the whole scene is modeled as a single object.

View-dependent LOD can provide better fidelity than other LOD techniques, and is especially beneficial for the topologically connected single large objects in the scenes. Therefore, in complex scenes with densely distributed objects, applying view-dependent LOD technique on objects with large polygon numbers while also using visibility culling to select the objects that should be rendered can be a solution to compensate the weaknesses of both techniques.

8.2.4 View-dependent LOD with textures

Most existing view-dependent LOD schemes, including ours, only deal with the geometric data of the models. In the practical applications in industry, education and entertainment, the material properties of a model, especially textures, play a very important part in realistic real-time rendering. Textures can add much detail to a scene with only a modest increase in rendering time. Texture mapping techniques have been developed for more than two decades, and they are more and more commonly used to simulate not only the color of the surface, but also the bumpiness, translucency and illumination. Textures have become an inseparable part of 3D models.

Some mesh simplification algorithms take into consideration the attribute errors incurred by the simplification operations. Both geometric errors and attribute errors are used to guide the simplification procedure and the run-time view-dependent visualization. The research works in [47] and [66] provide general frameworks for simplifying

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

models with attributes such as normals, colors and texture coordinates. Methods focused especially on optimizing the attribute of texture coordinates for simplified models have also been proposed in [21, 107].

Our framework can be extended to incorporate the attribute information of the models, such as texture coordinates, without much difficulty. However, the real problem involves the textures themselves, because the distortion of a simplified mesh not only comes from the errors of the geometry and the texture coordinates, but also from the texture representations. It becomes even more complicated for client-server based rendering systems, due to the high data volume of the textures.

Although the standard mip-mapping technique [131] is used in most existing mesh simplification and visualization systems that concern texture attribute [21, 107, 106], it is not suitable for client-server based rendering systems with low-capacity clients. Like discrete LOD technique, mip-maps increase memory consumption and network traffic and do not support progressive transmission for interruptible rendering. There are recent works addressing the problem of progressively transmitting the geometry and textures together to the client [96, 2, 139]. The geometric model is encoded by a conventional local simplification operator (e.g. edge-collapse) and the textures are encoded by standard image compression like JPEG2000 [19]. The geometry bits and the texture bits are combined into one bitstream to transmit to the client, in a way that at any time on the client, the currently received model achieves the best available quality considering both geometry and textures. This type of methods support progressive transmission of combined geometry and textures, but are not flexible enough for view-dependent refinement and simplification.

Thus, view-dependent and progressive refinement/simplification of textures according to the corresponding geometric refinement/simplification operations remains a challenging problem and an interesting topic for future research.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Acronyms

AOI	Area of Interest
CAD	Computer Aided Design
DAG	Directed Acyclic Graph
DCEL	Doubly Connected Edge List
DIS	Distributed Interactive Simulation
EWMA	Exponential Weighted Moving Average
FIFO	First-In First-Out
HDS	Hierarchical Dynamic Simplification
HLOD	Hierarchical Level-Of-Detail
HMD	Head-Mounted Display
HOB	Hierarchical Occlusion Map
HZ	Hierarchical Z-Buffer
IBR	Image-Based Rendering
LAN	Local-Area Network
LOD	Level-Of-Detail
MRA	Multiresolution Analysis
MT	Multi-Triangulation
PBNP	Position-Based Neighbor Prefetch
PGU	Procedure of Getting Updates
PRD	Procedure of Rendering
PM	Progressive Mesh
PS	Prediction Span
PVS	Potentially Visible Set
QE	Quadric Error
RTT	Round-Trip Time
SNP	Simple Neighbor Prefetch
VDPM	View-Dependent Progressive Mesh
VR	Virtual Reality
VE	Virtual Environment
VRML	Virtual Reality Modeling Language
WAN	Wide-Area Network

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Author's Publications

Journal papers:

1. Zhi Zheng and Tony K.Y. Chan, "A Client-server Based View-dependent Multiresolution Mesh Hierarchy," To appear in *International Journal of Computers and Applications*, ACTA Press, 2007.

2. Zhi Zheng, Tony K.Y. Chan, and Edmond C. Prakash, "Interactive View-dependent Rendering Over Networks," Accepted by *IEEE Transactions on Visualization and Computer Graphics*, 2007.

Conference papers:

1. Zhi Zheng, Tony K.Y. Chan, and Edmond C. Prakash "Rendering of Large 3D Models for Online Entertainment," *The 2006 International Conference on Game Research and Development*, pp. 163–170, December 2006, Australia.

2. Zhi Zheng and Tony K.Y. Chan, "Traversal on DAG-based Multiresolution Mesh Hierarchy," *The 3rd International Conference on Computer Graphics, Imaging and Visualisation* (*CGIV '06*), pp. 302–309, July 2006, Australia.

3. Zhi Zheng and Tony K.Y. Chan, "Interactive View-dependent Multiresolution on Low-Capacity Clients," *The Spring Conference on Computer Graphics* (*SCCG '06*, ISBN 80-223-2175-3), pp. 89–96, April 2006, Slovakia.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

4. Zhi Zheng and Tony K. Y. Chan, "View-dependent progressive mesh using non-redundant DAG hierarchy," *GRAPHITE '05*, pp. 417–420, November 2005, New Zealand.

5. Zhi Zheng and Tony K. Y. Chan, "A DAG Hierarchy for View-dependent Multiresolution Mesh," *International Workshop on CyberGames '05*, pp. 57–62, March 2005, Singapore.

6. Zhi Zheng and Tony K. Y. Chan, "Optimized Neighbour Prefetch and Cache for Client-server Based Walkthrough," *CyberWorlds '03*, pp. 143–150, December 2003, Singapore.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Bibliography

[1] *IEEE 1278. Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications.* 1993.

[2] María José Abásolo and Francisco Perales. Geometric-textured bitree: Transmission of a multiresolution terrain across the internet. *Journal of Computer Science & Technology*, 2(7):34–41, 2002.

[3] Daniel Aliaga, Jon Cohen, Andrew Wilson, Eric Baker, Hansong Zhang, Carl Erikson, Kenny Hoff, Tom Hudson, Wolfgang Stuerzlinger, Rui Bastos, Mary Whitton, Fred Brooks, and Dinesh Manocha. MMR: An interactive massive model rendering system using geometric and image-based acceleration. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics (SI3D '99)*, pages 199–206, 1999.

[4] Carlos Andújar, Carlos Saona-Vázquez, Isabel Navazo, and Pere Brunet. Integrating occlusion culling and levels of detail through hardly-visible sets. *Computer Graphics Forum (EUROGRAPHICS '00)*, 19(3):499–506, 2000.

[5] Ronald Tadao Azuma. *Predictive tracking for augmented reality.* PhD thesis, University of North Carolina at Chapel Hill, NC, USA, 1995.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[6] Chandrajit L. Bajaj, Valerio Pascucci, and Guozhong Zhuang. Progressive compression and transmission of arbitrary triangular meshes. In *Proceedings of IEEE Visualization '99*, pages 307–316, 1999.

[7] Bruce G. Baumgart. A polyhedron representation for computer vision. In *Proceedings of 1975 AFIPS National Computer Conference*, volume 44, pages 589–596, 1975.

[8] William V. Baxter-III, Avneesh Sud, Naga K. Govindaraju, and Dinesh Manocha. Gigawalk: Interactive walkthrough of complex environments. In *Proceedings of the 2002 Eurographics Workshop on Rendering (EGRW '02)*, pages 203–214, 2002.

[9] Gavin Bell, Rikk Carey, and Chris Marrin. The virtual reality modeling language specification version 2.0, 1996, ISO/IEC CD 14772, http://www.web3d.org/x3d/specifications/vrml/.

[10] Horst Birthelmer, Ingo Soetebier, and Jörg Sahm. Efficient representation of triangle meshes for simultaneous modification and rendering. In *International Conference on Computational Science*, pages 925–934, 2003.

[11] David Blythe, Brad Grantham, Mark J. Kilgard, Tom McReynolds, and Scott R. Nelson. Advanced graphics programming techniques using OpenGL. In *SIGGRAPH '99 Course notes*, 1999.

[12] Bobert Grover Brown and Patrick Y.C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. John Wiley and Sons, New York, 1992.

[13] Michael Capps, Don McGregor, Don Brutzman, and Michael Zyda. NPSNET-V: A new beginning for dynamically extensible virtual environments. *IEEE Computer Graphics and Applications*, 20(5):12–15, 2000.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[14] Addison Chan, Rynson W. H. Lau, and Beatrice Ng. A hybrid motion prediction method for caching and prefetching in distributed virtual environments. In *Proceedings of the 2001 ACM Symposium on Virtual Reality Software and Technology (VRST '01)*, pages 135–142, 2001.

[15] Addison Chan, Rynson W. H. Lau, and Beatrice Ng. Motion prediction for caching and prefetching in mouse-driven DVE navigation. *ACM Transactions on Internet Technology*, 5(1):70–91, 2005.

[16] Jatin Chhugani, Budirijanto Purnomo, Shankar Krishnan, Jonathan Cohen, Suresh Venkatasubramanian, David S. Johnson, and Subodh Kumar. vLOD: High-fidelity walkthrough of large virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):35–47, 2005.

[17] Jimmy H. P. Chim, Mark Green, Rynson W. H. Lau, Hong Va Leong, and Antonio Si. On caching and prefetching of virtual objects in distributed virtual environments. In *Proceedings of the 6th ACM International Conference on Multimedia (MULTIMEDIA '98)*, pages 171–180, 1998.

[18] Jimmy H. P. Chim, Hong Va Leong, Rynson W. H. Lau, and Antonio Si. Multiresolution cache management in digital virtual library. In *Proceedings of the Advances in Digital Libraries Conference (ADL '98)*, pages 66–75, 1998.

[19] Charilaos Christopoulos, Athanassios Skodras, and Touradj Ebrahimi. The JPEG2000 still image coding system: An overview. *IEEE Transactions on Consumer Electronics*, 46(4):1103–1127, 2000.

[20] Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. Adaptive tetrapuzzles: Efficient out-of-core construc-

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

tion and visualization of gigantic multiresolution polygonal models. In *Proceedings of SIGGRAPH '04*, pages 796–803, 2004.

[21] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. In *Proceedings of SIGGRAPH '98*, pages 115–122, 1998.

[22] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification envelopes. In *Proceedings of SIGGRAPH '96*, pages 119–128, 1996.

[23] Daniel Cohen-Or, Yiorgos Chrysanthou, Cláudio T. Silva, and Frédo Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, 2003.

[24] Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum (EUROGRAPHICS '98)*, 17(3):243–254, 1998.

[25] Daniel Cohen-Or and Eyal Zadicario. Visibility streaming for network-based walkthroughs. In *Proceedings of the 1998 Conference on Graphics Interface (GI '98)*, pages 1–7, 1998.

[26] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics (SI3D '97)*, pages 83–90, 1997.

[27] Wagner T. Corrêa, James T. Klosowski, and Cláudio T. Silva. Visibility-based prefetching for interactive out-of-core rendering. In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG '03)*, pages 1–8, 2003.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[28] Christopher DeCoro and Renato Pajarola. XFastMesh: Fast view-dependent meshing from external memory. In *Proceedings of IEEE Visualization '02*, pages 363–370, 2002.

[29] Leila DeFloriani and Paola Magillo. Multiresolution mesh representation: Models and data structures. In *Multiresolution in Geometric Modelling*, pages 363–418. Springer-Verlag, 2002.

[30] Leila DeFloriani, Paola Magillo, Franco Morando, and Enrico Puppo. Dynamic view-dependent multiresolution on a client-server architecture. *Computer-Aided Design*, 32(13):805–823, 2000.

[31] Leila DeFloriani, Paola Magillo, and Enrico Puppo. Efficient implementation of multi-triangulations. In *Proceedings of IEEE Visualization '98*, pages 43–50, 1998.

[32] Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. In *Proceedings of SIGGRAPH '00*, pages 239–248, 2000.

[33] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proceedings of SIGGRAPH '95*, pages 173–182, 1995.

[34] Jihad El-Sana. Multi-user view-dependent rendering. In *Proceedings of IEEE Visualization '00*, pages 335–342, 2000.

[35] Jihad El-Sana and Neta Sokolovsky. View-dependent rendering on large polygonal models over networks. *International Journal of Image and Graphics*, 3(2):265–290, 2003.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[36] Jihad El-Sana, Neta Sokolovsky, and Cláudio T. Silva. Integrating occlusion culling with view-dependent rendering. In *Proceedings of IEEE Visualization '01*, pages 371–378, 2001.

[37] Jihad El-Sana and Amitabh Varshney. Generalized view-dependent simplification. *Computer Graphics Forum (EUROGRAPHICS '99)*, 18(3):83–94, 1999.

[38] Jihad El-Sana and Amitabh Varshney. Parallel processing for view-dependent polygonal virtual environments. In *Proceedings of SPIE Vol. 3643 (Visual Data Exploration and Analysis VI)*, pages 62–70, 1999.

[39] Carl Erikson, Dinesh Manocha, and William Baxter. HLODs for fast display of large static and dynamic environments. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics (SI3D '01)*, pages 111–120, 2001.

[40] Efi Fogel, Daniel Cohen-Or, Revital Ironi, and Tali Zvi. A web architecture for progressive delivery of 3D content. In *Proceedings of the 6th International Conference on 3D Web Technology (Web3D '01)*, pages 35–41, 2001.

[41] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, and Richard L. Phillips. *Introduction to Computer Graphics*. Addison Wesley, 1993.

[42] Emmanuel Frecon and Marten Stenius. DIVE: A scalable network architecture for distributed virtual environments. *Distributed Systems Engineering Journal (special issue on Distributed Virtual Environments)*, 5(3):91–100, 1998.

[43] Thomas A. Funkhouser. RING: A client-server system for multi-user virtual environments. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics (SI3D '95)*, pages 85–92, 1995.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[44] Thomas A. Funkhouser. Database management for interactive display of large architectural models. In *Proceedings of the 1996 Conference on Graphics Interface (GI '96)*, pages 1–8, 1996.

[45] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of SIGGRAPH '93*, pages 247–254, 1993.

[46] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH '97*, pages 209–216, 1997.

[47] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings of IEEE Visualization '98*, pages 263–269, 1998.

[48] Tran S. Gieng, Bernd Hamann, Kenneth I. Joy, Greg Schussman, and Isaac J. Trotts. Constructing hierarchies for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145–161, 1998.

[49] Enrico Gobbetti and Eric Bouvier. Time-critical multiresolution rendering of large complex models. *Computer-Aided Design*, 32(13):785–803, 2000.

[50] Enrico Gobbetti and Fabio Marton. Far Voxels: A multiresolution framework for interactive rendering of huge complex 3D models on commodity graphics platforms. *ACM Transactions on Graphics*, 24(3):878–885, 2005.

[51] Naga K. Govindaraju, Avneesh Sud, Sung-Eui Yoon, and Dinesh Manocha. Interactive visibility culling in complex environments using occlusion-switches. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics (SI3D '03)*, pages 103–112, 2003.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[52] Markus Grabner. Consistency of the VDPM framework. In *Proceedings of SCCG '00*, pages 147–155, 2000.

[53] Markus Grabner. WebCAME: A light-weight modular client/server multiresolution rendering system. In *Proceeding of the 8th International Conference on 3D Web Technology (Web3D '03)*, pages 63–72, 2003.

[54] Ned Greene, Michael Kass, and Gavin Miller. Hierarchical Z-buffer visibility. In *Proceedings of SIGGRAPH '93*, pages 231–238, 1993.

[55] Chris Greenhalgh, Jim Purbrick, and Dave Snowdon. Inside MASSIVE-3: Flexible support for data consistency and world structuring. In *Proceedings of the 3rd International Conference on Collaborative Virtual Environments (CVE '00)*, pages 119–127, 2000.

[56] Anselm Grundhöfer, Benjamin Brombach, Robert Scheibe, and Bernd Fröhlich. Level of detail based occlusion culling for dynamic scenes. In *Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (GRAPHITE '05)*, pages 37–45, 2005.

[57] Andre Gueziec, Gabriel Taubin, Bill Horn, and Francis Lazarus. A framework for streaming geometry in VRML. *IEEE Computer Graphics and Applications*, 19(2):68–78, 1999.

[58] Andre Gueziec, Gabriel Taubin, Francis Lazarus, and Willian Horn. Simplicial maps for progressive transmission of polygonal surfaces. In *Proceedings of the 3rd Symposium on the Virtual Reality Modeling Language (VRML '98)*, pages 25–31, 1998.

[59] Michael Guthe, Pavel Borodin, and Reinhard Klein. Real-time out-of-core rendering. *International Journal of Image and Graphics*, 2006.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[60] Bernd Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11(2):197–214, 1994.

[61] Gerd Hesina and Dieter Schmalstieg. A network architecture for remote rendering. In *Proceedings of the 2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications (DIS-RT '98)*, pages 88–91, 1998.

[62] Hugues Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH '96*, pages 99–108, 1996.

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

[64] Hugues Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22(1):27–36, 1998.

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

[66] Hugues Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *Proceedings of IEEE Visualization '99*, pages 59–66, 1999.

[67] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings of SIGGRAPH '93*, pages 19–26, 1993.

[68] Wei Hua, Hujun Bao, Qunsheng Peng, and A. R. Forrest. The global occlusion map: A new occlusion culling approach. In *Proceedings of the 2002 ACM Symposium on Virtual Reality Software and Technology (VRST '02)*, pages 155–162, 2002.

[69] Tom Hudson, Dinesh Manocha, Jonathan D. Cohen, Ming C. Lin, Kenneth E. Hoff-III, and Hansong Zhang. Accelerated occlusion culling using shadow frusta.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 1–10, 1997.

[70] R.E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME – Journal of Basic Engineering*, 82D(1):35–46, 1960.

[71] Junho Kim and Seungyong Lee. Truly selective refinement of progressive meshes. In *Proceedings of the 2001 Conference on Graphics Interface (GI '01)*, pages 101–110, 2001.

[72] Junho Kim, Seungyong Lee, and Leif Kobbelt. View-dependent streaming of progressive meshes. In *Proceedings of the 2004 Shape Modeling International (SMI '04)*, pages 209–220, 2004.

[73] Andrew Kiruluta, Moshe Eizenman, and Subbarayan Pasupathy. Predictive head movement tracking using a Kalman filter. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 27(2):326–331, 1997.

[74] David Koller, Michael Turitzin, Marc Levoy, Marco Tarini, Giuseppe Croccia, Paolo Cignoni, and Roberto Scopigno. Protected interactive 3D graphics via remote rendering. *ACM Transactions on Graphics*, 23(3):695–703, 2004.

[75] Vladlen Koltun, Yiorgos Chrysanthou, and Daniel Cohen-Or. Virtual occluders: An efficient intermediate PVS representation. In *Proceedings of the 2000 Eurographics Workshop on Rendering (EGRW '00)*, pages 59–70, 2000.

[76] Vladlen Koltun, Yiorgos Chrysanthou, and Daniel Cohen-Or. Hardware-accelerated from-region visibility using a dual ray space. In *Proceedings of the 2001 Eurographics Workshop on Rendering (EGRW '01)*, pages 205–216, 2001.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[77] Fei-Ah Law and Tiow-Seng Tan. Preprocessing occlusion for real-time selective refinement. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics (SI3D '99)*, pages 47–53, 1999.

[78] Jed Lengyel and John Snyder. Rendering with coherent layers. In *Proceedings of SIGGRAPH '97*, pages 233–242, 1997.

[79] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital Michelangelo project: 3D scanning of large statues. In *Proceedings of SIGGRAPH '00*, pages 131–144, 2000.

[80] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of SIGGRAPH '00*, pages 259–262, 2000.

[81] Peter Lindstrom. Out-of-core construction and visualization of multiresolution surfaces. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics (SI3D '03)*, pages 93–102, 239, 2003.

[82] Peter Lindstrom and Valerio Pascucci. Visualization of large terrains made easy. In *Proceedings of IEEE Visualization '01*, pages 363–371, 2001.

[83] Peter Lindstrom and Cláudio T. Silva. A memory insensitive technique for large model simplification. In *Proceedings of IEEE Visualization '01*, pages 121–126, 2001.

[84] Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *Proceedings of IEEE Visualization '98*, pages 279–286, 1998.

[85] Michael Lounsbery. *Multiresolution analysis for surfaces of arbitrary topological type*. PhD thesis, University of Washington, Seattle, WA, USA, 1994.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[86] Michael Lounsbery, Tony D. DeRose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, 1997.

[87] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of SIGGRAPH '97*, pages 199–208, 1997.

[88] David Luebke, Martin Reddy, Jonathan Cohen, Amitabh Varshney, Benjamin Watson, and Rob Heubner. *Level of Detail for 3D Graphics*. Morgan Kaufmann Publishers, 2002.

[89] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics (SI3D '95)*, pages 95–102, 1995.

[90] Ashton E. W. Mason and Edwin H. Blake. Automatic hierarchical level of detail optimization in computer animation. *Computer Graphics Forum*, 16(3):C191–C199, 1997.

[91] Duncan C. Miller and Jack A. Thorpe. SIMNET: The advent of simulator networking. *Proceedings of the IEEE*, 83(8):1114–1123, 1995.

[92] Tomas Möller and Eric Haines. *Real-time rendering*. A. K. Peters, Ltd., Natick, MA, USA, 2002.

[93] David E. Muller and Franco P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217–236, 1978.

[94] Boaz Nadler, Gadi Fibich, Shuly Lev-Yehudi, and Daniel Cohen-Or. A qualitative and quantitative visibility analysis in urban scenes. *Computers & Graphics*, 23(5):655–666, 1999.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[95] Atul Narkhede and Dinesh Manocha. Fast polygon triangulation based on Seidel's algorithm. In *Graphics Gems 5*, pages 394–397, 1995.

[96] Masahiro Okuda and Tsuhan Chen. Joint geometry/texture progressive coding of 3D models. In *Proceedings of IEEE International Conference on Image Processing*, pages 632–635, 2000.

[97] Renato Pajarola. FastMesh: Efficient view-dependent meshing. In *Proceedings of Pacific Graphics '01*, pages 22–30, 2001.

[98] Renato Pajarola and Christopher DeCoro. Efficient implementation of real-time view-dependent multiresolution meshing. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):353–368, 2004.

[99] George V. Popescu and Zhen Liu. On scheduling 3D model transmission in network virtual environments. In *Proceedings of the 6th IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT '02)*, pages 127–133, 2002.

[100] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *Proceedings of SIGGRAPH '97*, pages 217–224, 1997.

[101] Chris Prince. *Progressive Meshes for Large Models of Arbitrary Topology*. Master thesis, University of Washington, Seattle, WA, USA, 2000.

[102] William T. Reeves. Particle systems – A technique for modeling a class of fuzzy objects. In *Proceedings of SIGGRAPH '83*, pages 359–375, 1983.

[103] Remi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum (EUROGRAPHICS' 96)*, 15(3):C67–C76, 1996.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[104] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. *Geometric Modeling in Computer Graphics*, pages 455–465, 1993.

[105] Jörg Sahm and Ingo Soetebier. A client-server-scenegraph for the visualization of large and dynamic 3D scenes. In *Proceedings of WSCG '04*, pages 387–394, 2004.

[106] Pedro V. Sander and Jason L. Mitchell. Progressive buffers: View-dependent geometry and texture for LOD rendering. In *Eurographics Symposium on Geometry Processing '05*, pages 129–138, 2005.

[107] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *Proceedings of SIGGRAPH '01*, pages 409–416, 2001.

[108] Carlos Saona-Vázquez, Isabel Navazo, and Pere Brunet. The visibility octree: A data structure for 3D navigation. *Computers & Graphics*, 23(5):635–643, 1999.

[109] Gernot Schaufler, Julie Dorsey, Xavier Decoret, and François X. Sillion. Conservative volumetric visibility with occluder fusion. In *Proceedings of SIGGRAPH '00*, pages 229–238, 2000.

[110] Dieter Schmalstieg and Michael Gervautz. Demand-driven geometry transmission for distributed virtual environments. *Computer Graphics Forum (EUROGRAPHICS '96)*, 15(3):421–433, 1996.

[111] Dieter Schmalstieg and Gernot Schaufler. Smooth levels of detail. In *Proceedings of the 1997 Virtual Reality Annual International Symposium (VRAIS '97)*, pages 12–19, 1997.

[112] Bengt-Olaf Schneider and Ioana M. Martin. An adaptive framework for 3D graphics over network. *Computers & Graphics*, 23(6):867–874, 1999.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[113] William J. Schroeder. A topology modifying progressive decimation algorithm. In *Proceedings of IEEE Visualization '97*, pages 205–212, 1997.

[114] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *Proceedings of SIGGRAPH '92*, pages 65–70, 1992.

[115] Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1(1):51–64, 1991.

[116] Leon A. Shirmun and Salim S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. *Computer Graphics Forum*, 12(3):261–272, 1993.

[117] Lidan Shou, Zhiyong Huang, and Kian-Lee Tan. Supporting real-time visualization with the HDoV tree. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC '03)*, pages 966–971, 2003.

[118] Sandeep Singhal and Michael Zyda. *Networked virtual environments: Design and implementation.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[119] Meehae Song, Thomas Elias, Wolfgang Möller-Wittig, and Tony K. Y. Chan. Interacting with the virtually recreated Peranakans. In *Proceedings of the First International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (GRAPHITE '03)*, pages 223–227, 2003.

[120] Alexei Sourin. Nanyang technological university virtual campus. *IEEE Computer Graphics and Applications*, 24(6):6–8, 2004.

[121] Richard Southern, Simon Perkins, Barry Steyn, Alan Muller, Patric Marais, and Edwin Blake. A stateless client for progressive view-dependent transmission. In

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

Proceedings of the 6th International Conference on 3D Web Technology (WEB3D '01), pages 43–50, 2001.

[122] Eyal Teler and Dani Lischinski. Streaming of complex 3D scenes for remote walkthroughs. *Computer Graphics Forum (EUROGRAPHICS '01)*, 20(3):17–25, 2001.

[123] Seth J. Teller. *Visibility Computations in Densely Occluded Environments.* PhD thesis, University of California, Berkeley, CA, USA, 1992.

[124] Seth J. Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Proceedings of SIGGRAPH '91*, pages 61–70, 1991.

[125] Danny S.P. To, Rynson W.H. Lau, and Mark Green. A method for progressive and selective transmission of multi-resolution models. In *Proceedings of the 1999 ACM Symposium on Virtual Reality Software and Technology (VRST '99)*, pages 88–95, 1999.

[126] Michiel van de Panne and James Stewart. Efficient compression techniques for precomputed visibility. In *Proceedings of the 1999 Eurographics Workshop on Rendering (EGRW '99)*, pages 305–316, 1999.

[127] Amitabh Varshney. *Hierarchical geometric approximations.* PhD thesis, University of North Carolina at Chapel Hill, NC, USA, 1994.

[128] Weihua Wang, Qingping Lin, Jim Mee Ng, and Chor Ping Low. SmartCU3D: a collaborative virtual environment system with behavior based interaction management. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '01)*, pages 25–32, 2001.

[129] Richard C. Waters, David B. Anderson, John W. Barrus, David C. Brogan, Michael A. Casey, Stephan G. McKeown, Tohei Nitta, Ilene B. Sterns, and

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

William S. Yerazunis. Diamond Park and Spline: A social virtual reality system with 3D animation, spoken interaction, and runtime modifiability. *Presence: Teleoperators and Virtual Environments*, 6(4):461–481, 1997.

[130] Kevin Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, 1985.

[131] Lance Williams. Pyramidal parametrics. In *Proceedings of SIGGRAPH '83*, pages 1–11, 1983.

[132] Nathaniel Williams, David Luebke, Jonathan D. Cohen, Michael Kelley, and Brenden Schubert. Perceptually guided simplification of lit, textured meshes. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics (SI3D '03)*, pages 113–121, 2003.

[133] Peter Wonka and Dieter Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum (EUROGRAPHICS '99)*, 18(3):51–60, 1999.

[134] Peter Wonka, Michael Wimmer, and Dieter Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Proceedings of the 2000 Eurographics Workshop on Rendering (EGRW '00)*, pages 71–82, 2000.

[135] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[136] Julie C. Xia, Jihad El-Sana, and Amitabh Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):171–183, 1997.

ATTENTION: The Singapore Copyright Act applies to the use of this document. Nanyang Technological University Library

[137] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *Proceedings of IEEE Visualization '96*, pages 327–334, 1996.

[138] Cui Xie, Xiuwen Liu, and Yicheng Jin. 3D scene transmission for web-based shiphandling training. In *Proceedings of the International Conference on Computer Graphics, Imaging and Visualization (CGIV '05)*, pages 291–298, 2005.

[139] Sheng Yang, Chao-Hua Lee, and C.-C. Jay Kuo. Optimized mesh and texture multiplexing for progressive textured model transmission. In *Proceedings of the 12th Annual ACM International Conference on Multimedia (MULTIMEDIA '04)*, pages 676–683, 2004.

[140] Christopher Zach and Konrad Karner. Prefetching policies for remote walkthroughs. In *Proceedings of WSCG '02*, pages 153–160, 2002.

[141] Hansong Zhang, Dinesh Manocha, Tom Hudson, and Kenneth E. Hoff-III. Visibility culling using hierarchical occlusion maps. In *Proceedings of SIGGRAPH '97*, pages 77–88, 1997.